Bài 8:
LẬP TRÌNH MẠNG BẰNG SOCKET

I. Khái niệm về socket

1. Socket

- Khi bạn viết ứng dụng và có yêu cầu tương tác với một ứng dụng khác, chúng ta thường dựa vào mô hình khách/chủ (client/server):
+ Ứng dụng chủ (trình chủ hay server): ứng dụng có khả năng phục vụ hoặc cung cấp cho bạn thông tin nào đó.
+ Ứng dụng khách (trình khách hay client): ứng dụng gửi yêu cầu đến trình chủ.
- Trước khi yêu cầu một dịch vụ của trình chủ thực hiện điều gì đó, trình khách (client) phải có khả năng kết nối được với trình chủ. Quá trình kết nối này được thực hiện thông qua một cơ chế trừu tượng hóa gọi là socket. Kết nối giữa trình khách và trình chủ tương tự như việc cắm phích điện vào ổ cắm điện .Trình khách thường được coi như phích cắm điện, còn trình chủ được xem như ổ cắm điện, một ổ cắm có thể cắm vào đó nhiều phích điện khác nhau cũng như một máy chủ có thể kết nối và phục vụ cho rất nhiều máy khách.
Nếu kết nối socket thành công thì trình khách và trình chủ có thể thực hiện các yêu cầu về trao đổi dữ liệu với nhau.

2. Kết nối socket

a) Server

- Trước hết ứng dụng chủ mở một socket . Đây chỉ là quá trình Hệ Điều Hành phân bổ tài nguyên để chuẩn bị kết nối. Bạn gọi là **`socket()`** để tạo "ổ cắm" socket cho trình chủ server.
- Để ứng dụng khách biết đến ổ cắm socket của trình chủ, bạn phải đặt cho socket trình chủ một tên. Nếu trên máy cục bộ và dựa vào hệ thống tập tin của Linux, bạn có thể đặt tên cho socket như là một tên tập tin (với đầy đủ đường dẫn). Bạn chỉ cần đặt tên còn đường dẫn thường đặt trong thư mục **`/tmp`** hay **`/usr/tmp`**. Đối với giao tiếp mạng thông qua giao thức **`TCP/IP`** tên của socket được thay thế bằng khái niệm cổng (port). Cổng là một số nguyên 2 bytes thay thế cho tên tập tin. Nếu trình khách và trình chủ nằm trên hai máy khác nhau, giao thức TCP/IP còn yêu cầu xác định thêm địa chỉ IP để kết nối đến máy chủ ở xa.
- Sau khi đã chỉ định tên hoặc số hiệu port cho socket, bạn gọi là **`bind()`** để ràng buộc hay đặt tên chính thức cho socket của trình chủ. Tiếp đến, trình chủ sẽ gọi hàm **`listen()`** để tạo hàm lắng nghe các kết nối từ trình khách đưa đến. Nếu có yêu cầu kết nối từ trình khách, trình chủ gọi hàm **`accept()`** để tiếp nhận yêu cầu của trình khách.
Hàm **`accept()`** sẽ tạo một socket vô danh khác (unnamed socket), cắm kết nối của trình khách vào socket vô danh này và thực hiện quá trình chuyển dữ liệu trao đổi giữa khách chủ. Socket được đặt tên trước đó vẫn tiếp tục hoạt động để chờ nhận yêu cầu từ trình khách khác.
- Mọi giao tiếp đọc ghi thông qua socket cũng đơn giản như việc bạn dùng lệnh read/write để đọc ghi trên tập tin. Nếu tập tin dựa vào số mô tả (file descriptor) để đọc ghi trên một tập tin xác định thì socket cũng dựa vào số mô tả (socket descriptor) để xác định socket cần đọc ghi cho hàm **`read`**/**`write`**.

b) Client

- Phía trình khách bạn chỉ cần tạo một socket vô danh, chỉ định tên và vị trí socket của trình chủ. Yêu cầu kết nối bằng hàm **`connect()`** và đọc ghi, truy xuất dữ liệu của socket bằng lệnh **`read`**/**`write`**.

- Dưới đây là một ví dụ đơn giản về trình khách **`client1.c`**. Trình khách kết nối với trình chủ thông qua socket mang tên **`server_socket`** và gửi ký tự A xem như lời chào bắt tay đến server.

client1.c

```
/* 1. Tạo các #include cần thiết để gọi hàm socket */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

int main()
{
  int sockfd;                          /* số mô tả socket - socket handle */
  int len;
  struct sockaddr_un address;          /* structure quan trọng, chứa các thông tin về socket */
  int result;
  char ch = 'A';
  /* 2. Tạo socket cho trình khách. Lưu lại số mô tả socket */
  sockfd = socket( AF_UNIX, SOCK_STREAM, 0 );
  address.sun_family = AF_UNIX;
  /* 3. Gán tên của socket trên máy chủ cần kết nối */
  strcpy( address.sun_path, "server_socket" );
  len = sizeof( address );
  /* 4. Thực hiện kết nối */
  result = connect( sockfd, (struct sockaddr*)&address, len );
  if ( result == -1 ) {
    perror( "Oops: client1 problem" );
    exit( 1 );
  }
  /* 5. Sau khi socket kết nối, chúng ta có thể đọc ghi dữ liệu của socket tương tự đọc ghi trên file */
  write( sockfd, &ch, 1 );
  read ( sockfd, &ch, 1 );
  printf( "char from server: %c\n", ch );
  close( sockfd );
  exit( 0 );
}
```

- Chương trình chưa chạy được do phần server (chính xác hơn là socket tên **server-socket** mà trình khách yêu cầu kết nối) chưa được thiếp lập.
- Dưới đây là trình chủ **server1.c** thực hiện mở socket, đặt tên cho socket là **server_socket**, mở hàng đợi lắng nghe kết nối của trình khách bằng **listen()**, chấp nhận kết nối bằng **accept()**. Sau cùng nhận/gửi dữ liệu về trình khách và đóng kết nối.

server1.c

```
/* 1.Tạo các #include cần thiết */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

int main()
{
  int server_sockfd, client_sockfd;
  int server_len, client_len;
  struct sockaddr_un server_address;
  struct sockaddr_un client_address;
  /* 2. Loại bỏ các tên hay liên kết socket khác trước đó nếu có. Đồng thời thực hiện khởi tạo socket mới cho trình chủ */
  unlink( "server_socket" );
  server_sockfd = socket( AF_UNIX, SOCK_STREAM, 0 );
  /* 3. Đặt tên cho socket của trình chủ */
  server_address.sun_family = AF_UNIX;
  strcpy( server_address.sun_path, "server_socket" );
  server_len = sizeof( server_address );
  /* 4. Ràng buộc tên với socket */
  bind( server_sockfd, (struct sockaddr *)&server_address, server_len );
  /* 5. Mở hàng đợi nhận kết nối - cho phép đặt hàng vào hàng đợi tối đa 5 kết nối */
  listen( server_sockfd, 5 );
  /* 6. Lặp vĩnh viễn để chờ và xử lý kết nối của trình khách */
  while ( 1 ) {
    char ch;
    printf( "server waiting...\n" );
    /* Chờ và chấp nhận kết nối */
    client_sockfd = accept( server_sockfd, (struct sockaddr*)&client_address, &client_len );
    /* Đọc dữ liệu do trình khách gửi đến */
    read( client_sockfd, &ch, 1 );
    ch++;
    /* Gửi trả dữ liệu về cho trình khách */
    write( client_sockfd, &ch,1 );
    /* Đóng kết nối */
    close( client_sockfd );
  }
}
```

- Dịch và chạy server dưới nền băng:

```
$ gcc server1.c –o server1
$ ./server1 &
```

- Dùng lệnh **ls** để thấy socket được tạo ra (chú ý ký tự kiểu tập tin là **s** – socket):

```
$ ls –lF
srwx-xr-r      1       s01     users   0   Apr 16:10   server_socket
```

- Dịch và chạy client:

```
$ gcc client1.c –o client1
$ client1
server waiting...
char from server: B
```

- Do server và client dùng chung màn hình nên sẽ thấy hai thông điệp.

II. Cách socket làm việc

1. Thuộc tính của socket

- Socket được định nghĩa dựa trên ba thuộc tính: vùng (domain), kiểu (type) và giao thức (protocol). Socket còn căn cứ vào một địa chỉ kết hợp với nó, địa chỉ này phụ thuộc vào domain của socket và thường gọi là họ giao thức (protocol family). Ví dụ họ giao thức theo hệ thống tập tin của Sun lấy tên tập tin làm đường dẫn đến địa chỉ, trong khi giao thức TCP/IP lại lấy địa chỉ là số IP 32bits để tham chiếu và thực hiện kết nối vật lý.

a) Vùng giao tiếp của socket (domain)

- Vùng dùng xác định hạ tầng mạng nơi giao tiếp của socket diễn ra. Vùng giao tiếp socket thông dụng nhất hiện nay là **AF_INET** hay giao tiếp socket theo chuẩn mạng Internet. Chuẩn này sử dụng địa chỉ **IP** để xác định nút kết nối vật lý trên mạng. Ngoài ra nếu bạn chỉ cần giao tiếp cục bộ, bạn có thể dùng giao tiếp của Sun **AF_UNIX**, đó là dùng đường dẫn và hệ thống tập tin để đặt tên và xác định kết nối giữa hai hay nhiều ứng dụng. Chẳng hạn như ví dụ trên, chúng ta đã thực hiện bằng cách đặt tên cho socket là server_socket, và Linux đã tạo ra tập tin **server_socket** ngay trên thư mục hiện hành (hay trong thư mục tạm **/tmp** tùy theo phiên bản của Linux). Tuy nhiên vùng giao tiếp **AF_UNIX** lại ít được sử dụng trong thực tế. Ngày nay hầu hết các ứng dụng mạng sử dụng socket đều theo vùng **AF_INET** là chủ yếu.
- Vùng **AF_INET** sử dụng địa chỉ **IP** (Internet Protocol) là một số 32bits để xác định kết nối vật lý. Số này thường được viết ở dạng nhóm như **192.168.1.1**, **203.162.42.1** hay **127.0.0.1** là các địa chỉ **IP** hợp lệ. Địa chỉ **IP** có thể được ánh xạ thành một tên gợi nhớ hơn như **www.yahoo.com** hay **www.microsoft.com** , chúng được gọi là tên vùng (domain name). Việc ánh xạ địa chỉ **IP** thành tên vùng thường do máy chủ **DNS** (Docmain Name Server) thực hiện. Cũng có thể tự ánh xạ tên vùng ngay trên máy cục bộ bằng cách sử dụng tập tin **/etc/host**.
- Socket theo dòng giao thức **IP** sử dụng port (số hiệu cổng) để đặt tên cho một socket. Cổng dùng để phân biệt dữ liệu gửi đến sẽ chuyển cho ứng dụng nào. Bạn hình dung nếu địa chỉ **IP** dùng để xác định được máy hay nơi kết nối vật lý để đưa dữ liệu đến thì cổng là địa chỉ phụ dùng để gửi

chính xác dữ liệu đến nơi ứng dụng cần. Điều này là do trên một máy có thể có nhiều ứng dụng cùng chạy và cùng sử dụng socket để giao tiếp. Các ứng dụng trên cùng một máy không được sử dụng trùng số cổng. Do cổng là một giá trị nguyên 2 bytes nên bạn có thể sử dụng khoảng 65535 cổng để tự do đặt cho socket. Trừ các số hiệu cổng quen thuộc như FTP, Web, …, bạn có thể chọn số cổng >1024 để mở cho socket của ứng dụng.
Mở socket theo **AF_INET** không khác mấy so với **AF_UNIX** ngoài việc đặt tên và chỉ định số hiệu cổng. Ví dụ:

```
/* Mở socket theo kết nối IP */
server_sockfd = socket ( AF_INET, SOCK_STREAM, 0 );
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet_addr( "127.0.0.1" );
server_address.sin_port = 1234;
```

b) Kiểu socket (type)

- Với mỗi vùng domain của socket, bạn có thể có nhiều cách giao tiếp. Có hai kiểu giao tiếp điển hình là giao tiếp một chiều (**SOCK_DGRAM**) và giao tiếp bảo đảm hai chiều (**SOCK_STREAM**). Dữ liệu khi chuyển đi được đóng thành từng gói (data package hay datagram). **SOCK_STREAM** hay **SOCK_DGRAM** qui định cách chuyển gói dữ liệu theo 2 cách sau:

+ **SOCK_STREAM** luôn giữ kết nối và bảo đảm gói thông tin gửi đi được chuyển đến đích và nhận đầy đủ. Xong khi nhận xong dữ liệu, nơi nhận sẽ gửi thông tin phản hồi cho nơi gửi và quá trình gửi sẽ được xác nhận hoàn tất khi nhận được đầu đủ thông tin gửi nhận ở cả hai phía khách chủ. Kiểu truyền dữ liệu socket này trong vùng giao thức mạng Internet **AF_INET** được gọi là giao thức truyền **TCP**. Khi kết hợp với cơ chế định tuyến theo địa chỉ **IP** chúng được gọi tắt là **TCP/IP**. Kết nối và truyền dữ liệu theo cách này tuy bảo đảm nhưng tốn nhiều tài nguyên của hệ thống do kết nối phải duy trì và theo dõi thường xuyên.

+ **SOCK_DGRAM** cách gửi nhận dữ liệu này chỉ diễn ra một chiều. Trình khách, nơi gửi dữ liệu đi không cần biết trình chủ (nơi nhận) có nhận được gói dữ liệu đầy đủ hay không. Trình chủ khi nhận được dữ liệu cũng không cần thông báo hay xác nhận với trình khách là dữ liệu đã nhận xong. Cách gửi này có vẻ không an toàn nhưng bù lại thực hiện nhanh và không tiêu tốn nhiều tài nguyên để duy trì kết nối và kiểm tra dữ liệu nhận gửi. Kiểu truyền dữ liệu socket này trong vùng giao thức mạng Internet **AF_INET** được gọi là giao thức truyền **UDP** (User Datagram Protocol). Khi kết hợp với cơ chế định tuyến theo địa chỉ **IP** chúng được gọi tắt là **UDP/IP**. Do tính chất không bảo đảm nên **UDP** được dùng nhiều trong mạng với mục đích thông báo hay gửi tin đến một nhóm máy tính nào đó trong mạng. Nó ít khi được dùng trong các dịch vụ cần sự chính xác cao như gửi nhận tập tin hay kết nối cơ sở dữ liệu.

c) Giao thức của socket (protocol)

- Giao thức là cách quy ước gửi nhận dữ liệu giữa hai hay nhiều máy tính trong mạng. Tùy theo kiểu gửi nhận dữ liệu mà ta có các giao thức khác nhau. Hiện nay kiểu gửi nhận theo giao thức **TCP** và **UDP** là sử dụng nhiều nhất. Tuy nhiên điều này không bắt buộc, **TCP** và **UDP** chỉ là cách cài đặt cho vùng socket **AF_INET** theo giao tiếp Internet mà thôi. Ví dụ, bạn có thể cài đặt một giao thức khác thay cho **UDP** áp dụng cho cách chuyển dữ liệu không đảm bảo.Mỗi kiểu socket đều có giao thức mặc định áp dụng cho nó. Trong tất cả các hàm socket sau này, tùy chọn protocol nếu có yêu cầu bạn có thể đặt trị 0 để yêu cầu sử dụng giao thức mặc định.

Trong ví dụ ở phần sau ta chỉ tập trung nghiên cứu về vùng socket **AF_INET** và địa chỉ **IP** thay cho vùng **AF_UNIX** ở ví dụ đầu tiên. Với **AF_INET** mọi ứng dụng mạng của bạn đều có thể giao tiếp được với nhau bất kể chúng đang chạy trên Windows, Linux hay UNIX.

2. Tạo socket

- Hệ thống cung cấp cho bạn hàm **socket()** để tạo mới một socket. Hàm **socket()** trả về số nguyên **int** cho biết số mô tả hay định danh dùng để truy cập socket sau này, còn gõi là socket handle.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket( inte domain, int type, int protocol );
```

+ Tham số **domain** chỉ định vùng hay họ địa chỉ áp đặt cho socket. **domain** có thể nhận một trong các giá trị sau:

AF_UNIX	Mở socket kết nối theo giao thức tập tin (xuất nhập socket dựa trên xuất nhập tập tin) của UNIX/Linux.
AF_INET	Mở socket theo giao thức Internet (sử dụng địa chỉ IP để kết nối).
AF_IPX	Vùng giao thức IPX (Mạng Novell).
AF_ISO	Chuẩn giao thức ISO.
AF_NS	Giao thức Xerox Network System.

Hầu như bạn chỉ sử dụng **AF_UNIX** và **AF_INET** là chính. Các vùng giao tiếp khác đã lỗi thời và hiện nay ít còn được sử dụng.

+ Tham số **type** trong hàm **socket()** dùng chỉ định kiểu giao tiếp hay truyền dữ liệu của socket. Bạn có thể chỉ định hằng **SOCK_STREAM** dùng cho truyền dữ liệu bảo đảm hoặc **SOCK_DGRAM** dùng cho kiểu truyền không bảo đảm.

+ Tham số protocol dùng để chọn giao thức áp dụng cho kiểu socket (trong trường hợp có nhiều giao thức áp dụng cho một kiểu truyền). Tuy nhiên bạn chỉ cần đặt giá trị 0 (lấy giao thức mặc định). **AF_INET** chỉ cài đặt một giao thức duy nhất cho các kiểu truyền **SOCK_STREAM** và **SOCK_DGRAM**, đó là **TCP** và **UDP**.

Nếu tạo socket thành công, hàm sẽ trả về số định danh socket (socket handle). Bạn sử dụng số định danh này trong tất cả các lời gọi truy xuất socket khác như **read**/**write**. Đọc/ghi vào socket cũng đồng nghĩa với gửi và nhận dữ liệu giữa trình khách và trình chủ.

Để đóng socket đã mở trước đó, bạn có thể gọi hàm **close()**.

3. Định địa chỉ socket

- Một socket được tạo ra phải cần có địa chỉ và tên để các trình khách có thể tham chiếu đến. Tùy domain mà cách đánh địa chỉ có thể khác nhau:

+ Khi dùng domain **AF_UNIX** dựa trên hệ thống tập tin (ví dụ trên) , địa chỉ socket được định nghĩa trong structure **sockaddr_un** của tập tin **sys/un.h** như sau:

```
struct sockaddr_un {
  sa_family_t sun_family;     /* AF_UNIX */
  char sun_path;              /* đường dẫn */
```

+ Khi dùng **AF_INET**, structure địa chỉ được định nghĩa trong **sockaddr_in** của tập tin **netinet/in.h** như sau:

```
struct sockadd-in {
  short int sin_ family;      /* AF_INET */
  unsigned short int sin_port /* số hiệu cổng */
  struct in_addr sin_addr;    /* địa chỉ IP */
```

```
};
```
Trong đó:
```
struct in_addr {
  unsigned long int s_addr;
};
```

4. Đặt tên cho socket

- Sau khi định địa chỉ, cần đặt tên cho socket:

+ Với **AF_UNIX**, địa chỉ là đường dẫn, tên là tên tập tin để tạo socket.

+ Với **AF_INET**, địa chỉ chính là **IP**, tê là số hiệu cổng **sin_port**.

Sau khi gán thông tin đầy đủ và các structure, gọi hàm **bind()** ràng buộc thông tin này cho socket đã mở ra trước đó.
```
#include <sys/socket.h>
int bind( int socketd, const struct sockaddr *address, size_t address_len );
```
socketd: handle của socket, có được do hàm **socket()** trả về.

address: struct **sockaddr**, tổng quát cho **sockaddr_un** và **sockaddr_in**, chứa thông tin địa chỉ socket.

address_len: chiều dài struct **sockaddr**.

Nếu thành công, **bind()** trả về trị **0**, nếu không hàm trả về trị **-1** và **errno** sẽ chứa mã lỗi.

5. Tạo hàng đợi cho socket

- Để đón nhận các kết nối chuyển đến, trình chủ phải tạo hàng đợi (queue) bằng hàm **listen()**:
```
#include <sys/socket.h>
int listen( int socketd, int backlog );
```
socketd: handle của socket.

backlog: số kết nối tối đa được phép đưa vào hàng đợi, thông thường là **5**.

Nếu thành công, **listen()** trả về trị **0**, nếu không hàm trả về trị **-1** và **errno** sẽ chứa mã lỗi.

6. Chờ và chấp nhận kết nối

- Công việc sau cùng là chờ kết nối của trình khách gửi đến bằng hàm **accept()**:
```
#include <sys/socket.h>
int accept( int socketd, struct sockaddr *address, size_t address_len );
```
socketd là handle của socket máy chủ đang lắng nghe. Khi **accept()** phát hiện có socket trong hàng đợi, nó tự động lấy địa chỉ máy khách đặt vào cấu trúc **address**, chiều dài thật sự của cấu trúc địa chỉ này đặt trong **address_len**. Tiếp đó, **accept()** tạo ra một socket vô danh, kết quả trả về của **accept()** là handle của socket vô danh này, ta dùng nó để liên lạc với trình khách. Socket vô danh có cùng kiểu với socket đặt tên đang lắng nghe trên hàng đợi. Nếu chưa có kết nối nào trong hàng đợi, **accept()** sẽ dừng lại chờ (block).

7. Yêu cầu kết nối

- Trình chủ thực hiện chức năng tạo ⇒ đặt tên ⇒ chờ kết nối bằng các hàm: **socket()**, **bind()**, **accept()**. Trình khách đơn giản hơn, chỉ gọi hàm **connect()** để yêu cầu kết nối với trình chủ:
```
#include <sys/socket.h>
int connect( int socketd, struct sockaddr *address, size_t address_len );
```
Nếu thành công, **connect()** trả về trị 0, nếu không hàm trả về trị **-1** và **errno** sẽ chứa mã lỗi. Nếu kết nối không thành công **connect()** sẽ cố gắng chờ kết nối lại. Sau một số lần kết nối không thành công, **connect()** trả về mã lỗi **ETIMEOUT**.

8. Đóng kết nối

- Đóng kết nối với cả hai trình khách và chủ, sẽ giải phóng tài nguyên và bảo đảm dữ liệu chuyển tải hoàn tất. Dùng hàm **close()**.
```
#include <sys/socket.h>
int close( int socketd );
```
- Dưới đây là một ví dụ minh họa dùng **AF_INET** thay cho **AF_UNIX**:

client2.c
```
/* 1. Tạo các #include cần thiết để gọi hàm socket */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
/* dành riêng cho AF_INET */
#include <netinet/in.h>
#include <arpa/inet.h>

int main()
{
  int sockfd;                          /* số mô tả socket - socket handle */
  int len;
  struct sockaddr_in address;          /* structure sockaddr_in, chứa các thông tin về socket AF_INET */
  int result;
  char ch = 'A';
  /* 2. Tạo socket cho trình khách. Lưu lại số mô tả socket */
  sockfd = socket( AF_INET, SOCK_STREAM, 0 );
  /* 3. Đặt tên và gán địa chỉ kết nối cho socket theo giao thức Internet */
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = inet_addr( "127.0.0.1" );
  address.sin_port = htons( 9734 );
  len = sizeof( address );
  /* 4. Thực hiện kết nối */
```

```
  result = connect( sockfd, (struct sockaddr*)&address, len );
  if ( result == -1 ) {
    perror( "Oops: client1 problem" );
    exit( 1 );
  }
  /* 5. Sau khi socket kết nối, chúng ta có thể đọc ghi dữ liệu của socket tương tự đọc ghi trên file */
  write( sockfd, &ch, 1 );
  read ( sockfd, &ch, 1 );
  printf( "char from server: %c\n", ch );
  close( sockfd );
  exit( 0 );
}
```

server2.c

```
/* 1.Tạo các #include cần thiết */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
/* dành riêng cho AF_INET */
#include <netinet/in.h>
#include <arpa/inet.h>

int main()
{
  int server_sockfd, client_sockfd;
  int server_len, client_len;
  struct sockaddr_in server_address;
  struct sockaddr_in client_address;
  /* 2. Thực hiện khởi tạo socket mới cho trình chủ */
  server_sockfd = socket( AF_INET, SOCK_STREAM, 0 );
  /* 3. Đặt tên và gán địa chỉ kết nối cho socket theo giao thức Internet */
  server_address.sin_family = AF_INET;
  server_address.sin_addr.s_addr = inet_addr( "127.0.0.1" );
  server_address.sin_port = htons( 9734 );
  server_len = sizeof( server_address );
  /* 4. Ràng buộc tên với socket */
  bind( server_sockfd, (struct sockaddr *)&server_address, server_len );
  /* 5. Mở hàng đợi nhận kết nối - cho phép đặt hàng vào hàng đợi tối đa 5 kết nối */
  listen( server_sockfd, 5 );
  /* 6. Lặp vĩnh viễn để chờ và xử lý kết nối của trình khách */
  While ( 1 ) {
    char ch;
    printf( "server waiting...\n" );
    /* Chờ và chấp nhận kết nối */
    client_sockfd = accept( server_scockfd, (struct sockaddr*)&client_address, &client_len );
    /* Đọc dữ liệu do trình khách gửi đến */
    read( client_sockfd, &ch, 1 );
    ch++;
    /* Gửi trả dữ liệu về cho trình khách */
    write( client_sockdf, ch,1 );
    /* Đóng kết nối */
    close( client_sockfd );
  }
}
```

- Tập tin văn bản **/etc/host** chứa các cặp địa chỉ **IP** - tên gợi nhớ. Trong chương trình ta có thể dùng tên gợi nhớ (ví dụ "**localhost**") hoặc địa chỉ **IP** (ví dụ "**127.0.0.1**"). Nếu dùng địa chỉ **IP** khác, cần kiểm tra trước xem địa chỉ đó có tồn tại không:

```
$ ping 192.168.2.250
PING 192.168.2.250 (192.168.2.250) from 192.168.2.250: 56(84) bytes of data.
64 bytes from 192.168.2.250: icmp_seq=1 ttl=255 time=0.083ms
64 bytes from 192.168.2.250: icmp_seq=2 ttl=255 time=0.089ms
64 bytes from 192.168.2.250: icmp_seq=3 ttl=255 time=0.103ms
64 bytes from 192.168.2.250: icmp_seq=4 ttl=255 time=0.088ms

--- 192.168.2.250 ping statistics ---
4 packets transmitted, 4 received, 0% loss, time 3009ms
rtt min/avg/max/mdev = 0.083/0.090/0.103/0.113 ms
```

- Việc chọn số hiệu cổng cho chương trình phải loại trừ những cổng đã được các ứng dụng khác sử dụng. Tham khảo tập tin **/etc/services** liệt kê danh sách các dịch vụ và cổng đã sử dụng. Số cổng không được nhỏ hơn **1024** (dành cho các dịch vụ của hệ thống).

III. Xử lý kết nối đồng thời của nhiều trình khách

- Trong mô hình client/server một trình chủ có thể phục vụ đồng thời cho nhiều trình khách. Ở ví dụ trên, trình chủ gọi hàm accept() chờ kết nối đến, xử lý xong kết nối rồi mới quay lại nhận kết nối tiếp theo. Đây là cách xử lý tuần tự và thường không phù hợp với việc nhiều trình khách yêu cầu phục vụ cùng lúc.
- Bạn có thể sử dụng lệnh fork () để kiến tạo tiến trình con mới. Tiến trình con mới này hoạt động độc lập với trình chủ và chịu trách nhiệm phục vụ trình khách theo cách riêng của nó. Trình chủ hoàn toàn tự do để tiếp nhận ngay kết nối khác. Ngoài cách tạo lập tiến trình con mới bạn có thể sử dụng cách tạo tuyến (thread). Tuy nhiên tuyến không thường được sử dụng trong UNIX và LINUX bằng tiến trình (process).
- Dưới đây là chương trình cho thấy cách sử dụng mô hình client/server phục vụ kết nối đồng thời từ nhiều trình khách.

server3.c

```
/*
Như thường lệ, phần đầu đầu là nơi khai báo các tập tin header cần thiết, đồng thời koiwr tạo các biến dùng
cho chương trình. Bạn lưu ý, ta thêm vào signal.h để sử dụng các hằng khai báo xử lý tín hiệu.
*/

#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>

int main()
{
        int server_sockfd, client_sockfd;
        int server_len, client_len;
        struct sockaddr_in server_address;
        struct sockaddr_in client_address;

        server_sockfd = socket (AF_INET, SOCK_STREAM, 0)
        server_address.sin_family = AF_INET;
        server_address.sin_addr.s_addr = htonl (INADDR_ANY);
        server_address.sin_port = htons (9734);
        server_len = sizeof (server_address);

        bind(server_sockfd, (struct sockaddr*) &server_address, server_len);

/*
Tạo hàng đợi để nhận kết nối, yêu cầu bỏ qua tín hiệu kết thúc của các tiến trình con gởi đến tiến trình cha.
Tạo vòng lặp chờ kết nối từ trình khách.
*/
        listen (server_sockfd, 5);
        signal (SIGCHLD, SIG_IGN);
        while (1)
        {
                char ch;
                printf("Server waiting...\n");

/*
Chờ kết nói và chấp nhận kết nối từ trình khách.
*/
                client_len = sizeof (client_address);
                client_sockfd = accept (server_sockfd, (struct sockaddr*) &client_address, &client_len);

/*
Gọi hàm fork () tạo tiến trình con để xử lý kết nối, kiểm tra xem hiện ta đang là tiến trình cha hay tiến
trình con.
*/
                if (fork() == 0)
                {
/*
Nếu hiện là tiến trình con, ta hoàn toàn có thể đọc và ghi vào socket client_sockfd. Chúng ta gọi hàm sleep()
để dừng lại 3 giây để mô phỏng quá trình xử lý thực tế của tiến trình con như kết nối cơ sở dữ liệu, xử lý
nhập xuất…
*/
                        read (client_sockfd, &ch, 1);
                        sleep (3);
                        ch++;
                        write (client_sockfd, &ch, 1);
                        close (client_sockfd);
                        exit (0);
                }
                else
                {
/*
Nếu không, hiện chúng ta đã ở tiến tình cha, quá trình xử lý kết nối đã hoàn tất. socket dành cho client có
thể đóng lại.
*/
                        close (client_sockfd);
```

```
            }
        }
}
```

- Sử dụng client2 để thực hiện kết nối và tương tác với server3 trên đây. Dưới đây là kết quả xuất khi ta cho chạy ngầm server3 ở hậu cảnh và liên tục thực hiện kết nối đến server3 bằng client2.

```
$./server3
[7] 1571
Server waiting…

$client2 & client2 & client2 & ps –ax
[8] 1572
[9] 1573
[10] 1574
Server waiting…
Server waiting…
Server waiting…
PID     TTY       STAT     TIME         COMMAND
1577    ppo       S        0:00:00      server3
1572    ppo       S        0:00:00      client2
1573    ppo       S        0:00:00      client2
1574    ppo       S        0:00:00      client2
1575    ppo       S        0:00:00      ps -ax
1576    ppo       S        0:00:00      server3
1577    ppo       S        0:00:00      server3
1578    ppo       S        0:00:00      server3
char from server = B
char from server = B
char from server = B

$ps –ax
PID     TTY       STAT     TIME         COMMAND
1577    ppo       S        0:00:00      server3
1580    ppo       S        0:00:00      ps –ax
[8] Done       client2
[9] –Done      client2
[10] +Done     client2
```