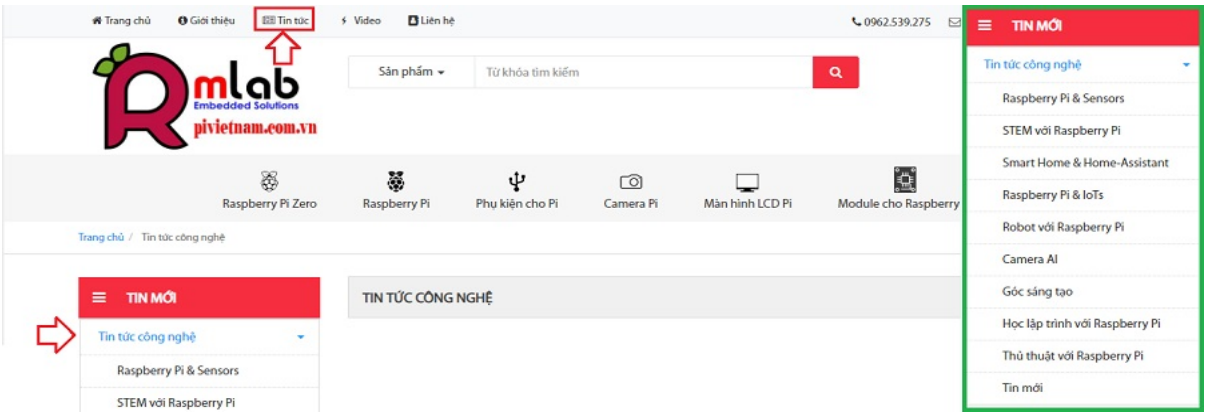
Bài 5 : Lập trình giao tiếp mạng TCP/IP Raspberry Pi phần 1

CHÚ Ý : *Từ 2019 MLAB có thêm một website cho riêng Raspberry Pi và trở thành website chính về Raspberry Pi tại MLAB, các thông tin về sản phẩm - tin tức cập nhật về Raspberry Pi - Bài viết kỹ thuật hỗ trợ cho Raspberry Pi, ... MLAB cập nhật tại website* : *pivietnam.com.vn*

MLAB trân trọng thông báo tới quý khách hàng!!!

Các bạn có thể tham khảo các bài viết hỗ trợ kỹ thuật và các tin tức mới nhất tại phần "tin tức"trên website PVIETNAM.COM.VN



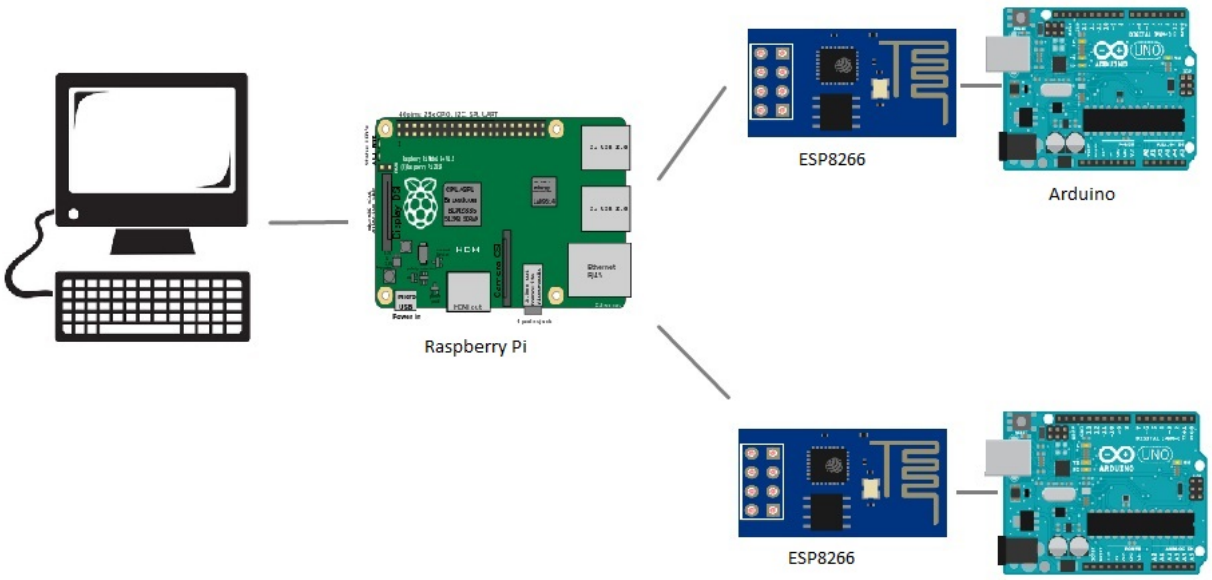
Bài viết hỗ trợ kỹ thuật tại website PIVIETNAM.COM.VN - Bài 5: Lập trình giao tiếp mạng TCP/IP Raspberry Pi Phần 1 (Link here)

Mô hình Server & Client

Mô hình client & server là một mô hình giao tiếp phổ biến trong truyền thông thông tin. Mô hình gồm 2 phía, phía Client và Server. Client có chức năng thực hiện một tác vụ chuyên biệt nào đó, nó có thể truyền thông tin và nhận lệnh từ phía Server. Server thì nhận thông tin từ Client và gửi thông tin điều khiển.

Phía Client sẽ luôn chủ động tạo kết nối tới Server, còn bên phía Server sẽ ở trong chế độ chờ client kết nối tới. Client trong một lúc chỉ có thể kết nối tới duy nhất một Server nhưng ngược lại Server có thể đồng thời giao tiếp với nhiều Client.

Trong bài đọc này sẽ hướng dẫn tạo lập Raspberry Pi làm Server trung tâm :



Hình 1 : mô hình kết nôi máy tính - raspberry pi - các vi điều khiển khác

Client có thể là máy tính mà thông qua đó sẽ gửi các thông tin tới Pi hoặc thông qua Pi đê gián tiếp điều khiển các client khác. Client có thể là các vi điều khiển được hỗ trợ thêm phần cứng kết nối mạng như esp8266.

Bài đọc sẽ được chia làm hai phần :

Phần 1 : Tạo lập chương trình đơn giản server-client dựa trên phương thức TCP-IP. Lấy Pi làm server và máy tính làm client.

Phần 2 : Chuyển đổi server điều khiển được nhiều client cùng lúc. Lấy client là các arduio kết hợp esp8266.

Phần 1 : Tạo lập chương trình Server-client

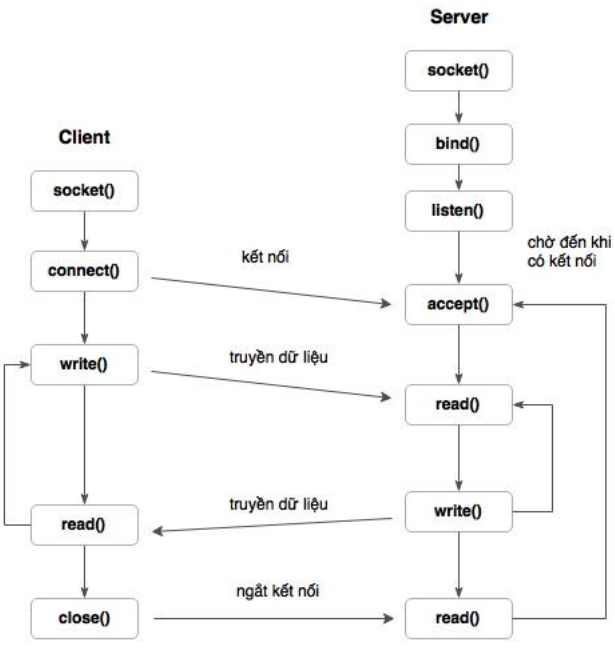
Thuật toán và chương trình được viết dưới đây chạy trên hệ điều hành linux, được test trên Ubuntu – Raspberry. Mình cũng cung cấp thêm chương trình đơn giản trên window dành cho những bạn không quen dùng với linux.

1. Kiến thức nền tảng

Yêu cầu trước cho mọi người :

- Có kiến thức cơ bản về lập trình C.
- Hiểu biết cơ bản về TCP – IP. Bao gồm địa chỉ Ip,port và socket là gì. Giao thức TCP.
- Quen với linux. Nếu mọi người không quen có thể download đoạn code trên window phía bên dưới.

Thuật toán kết nối giữa server và client.



Hình 2 : thuật toán cllient kết nối server

Các bước được thực hiện với **client** :

B1. Tạo socket để kết nói qua hàm socket().

B2. Kết nối tới server nhờ hàm connect(). Hàm này thông thường sẽ block cho tới khi nào kết nối được.

B3. Gửi và nhận thông tin qua hàm send/recv.

Các bước được thực hiện với **server** :

B1. Tạo socket để kết nói qua hàm socket()

B2. Gắn socket vừa tạo với địa chỉ của server. Dùng hàm bind().

B3. Thiết lập socket để chờ kết nối. Dùng hàm listen().

B4. Chấp nhận kết nối từ client. Hàm accept() thông thường sẽ bị block tới khi nào có kết nối tới.

B5. Gửi và nhận thông tin qua hàm send()/recv().

Lưu ý : Client khi kết nối tới server cần phải biết địa chỉ của server và cổng kết nối. Ngược lại server không cần phải biết địa chỉ của client mà chỉ cần đợi kết nối từ client đến. Nhất thiết trên client và server phải có chung cổng kết nối.

Toàn bộ thư viện và các hàm API trong bài đọc được cài đặt sẵn trên linux.

1.1 Thiết lập socket

Ngay khi bắt đầu thực thi chương trình. Chương trình cần yêu cầu hệ thống tạo ra socket mới để chương trình có thể sử dụng.

```
int socket(int protocolFamily, int  type, int protocol)
```

Trong đó :

- protocolFamily : Tham số chỉ ra phương thức protocol family cho socket. Có nhiều phương thức khác nhau như AF_INET dùng cho Ipv4 Internet protocols. AF_INET6 cho Ipv6 Internet protocols. Ngoài ra còn có phương thức dùng chung cho đa số protocols khác nhau, đó là PF_INET.

- type : chỉ ra thể loại socket được dùng. SOCK_STREAM chỉ ra phương thức truyền gửi đáng tin cậy (cũng tức là dùng với TCP), ngoài ra còn có SOCK_DGRAM dành riêng cho việc truyền nhanh datagram socket (trong UDP).

- protocol : Chỉ ra protocol được sử dụng như là TCP hay UDP. Với protolFamily là PF_INET, tham số tương ứng với TCP là IPPROTO_TCP còn UDP sẽ là IPPROTO_UDP. Bạn có thể dùng số 0 để chương trình tự chọn phương thức phù hợp như TCP cho strem socket (SOCK_STREAM) hay UDP cho datagram socket (SOCK_DGRAM).

- Kết quả trả về là là một file-descriptor. Nó thực ra là một số kiểu int. Nếu có lỗi xảy ra kết quả sẽ là -1.

Để hủy đi một socket có thể sử dụng hàm :

```
int close(int socket)
```

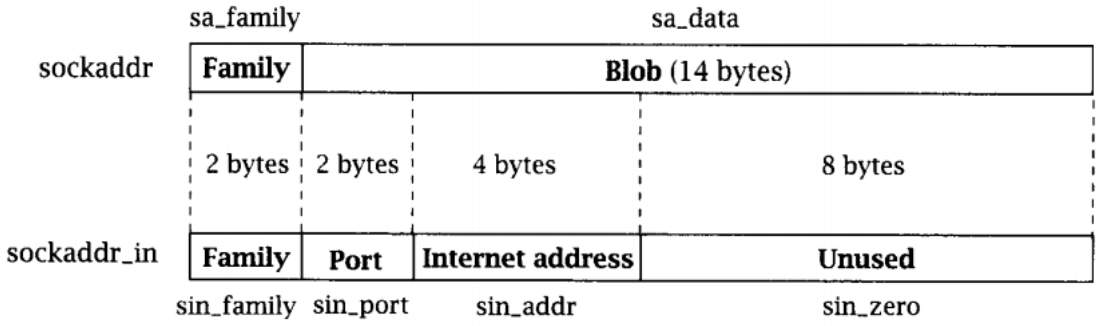
Trong đó tham số socket sẽ chính là socket được trả về từ hàm socket().

1.2 Thiết lập địa chỉ

Trong API có cung cấp các cấu trúc địa chỉ dưới đây..

```
struct sockaddr
{
    unsigned short sa_family; /* Address family (e.g. AF_INET) */
    char  sa_data[14] ; /* Family-specific address information */
};
struct in_addr
{
    unsigned long s_addr;
};
struct sockaddr_in
{
    unsigned short sin_family; /* Internet protocol (AF_INET) */
    unsigned short sin_port; /* Address port (16 bits) */
    struct in_addr sin_addr; /* Internet address (32 bits) */
    char    sin_zero[8]; /* Not used, must be zero */
};
```

Hai cấu trúc sockaddr và sockaddr_in có thể dùng thay thế được cho nhau để thiết lập các thông số địa chỉ cho server và client. Cấu trúc sockaddr_in được sử dụng để thể hiện chi tiết hơn về các thông số. Các hàm trong API sẽ dùng các cấu trúc địa chỉ này như tham biến chứa thông tin (nhìn xuống phần code hoàn thiện để thấy rõ hơn) .



Trong đó :

- sin_family, sa_family : chính là protocolFamily.
- sin_port : port địa chỉ kết nối socket.
- sin_addr : địa chỉ kết nối tới.
- sin_zero : không được sử dụng, đặt bằng 0.

2. Chương trình Client – Server

Chương trình Client – server sẽ thực hiện giao tiếp đơn giản như sau. Client sẽ nhận dữ liệu là chuỗi ký tự được bạn viết trên terminal và gửi dữ liệu đó cho Server. Server sẽ nhận dữ liệu và hiển thị lên terminal.

2.1 TCP Client

Bước 1 : Thiết lập socket với các tham số sau :

```
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)
```

Bước 2 : Khi thiết lập xong socket, cần phải thực hiện kết nối với các socket khác để có thể truyền nhận thông tin. Ở đây chính là kết nối tới server.

```
int connect(int socket,struct sockaddr *addr,unsigned int addrLength)
```

Trong đó :

- Socket : là socket vừa được tạo từ hàm socket()
- Addr : là địa chỉ server muốn kết nối tới.
- Addrlength : là độ dài của địa chỉ server (tính ra byte). 4 byte cho IPV4, 16 byte cho IPV6.

Bước 3 : Hàm gửi thông tin là send() và nhận thông tin là recv()

```
int send(int socket,const void *msg,unsigned int len,int flag)
int recv(int socket,void *Buff,unsigned int len,int flag)
```

Trong đó :

- *msg, *buff : con trỏ tới mảng dữ liệu để gửi hoặc nhận.
- len : chiều dài của mảng dữ liệu, cũng là số byte tối đa gửi hoặc nhận trong một lần gọi hàm.
- flag : flag giúp thiết lập chế độ truyền/nhận. Thiết lập là 0 để chọn chế độ mặc định.
- Dữ liệu trả về là số byte được gửi hoặc nhận. Trả về -1 nếu có lỗi xảy ra.

Chương trình client

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), send(), and recv() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define BUFFSIZE 256
#define PORT 8888

void error(const char *msg){
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[]){
    int sockfd, n;
```

```
    struct sockaddr_in serv_addr;
    char buffer[BUFFSIZE];
    char *servIP;

    if(argc < 2){
        fprintf(stderr, "Hay nhap: %s <Server IP> \n",argv[0]);
        exit(1);
    }
    servIP = argv[1];       // localhost : "127.0.0.1"

    // Tao socket
    if ((sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        error("socket() failed");

    // Ghi cau truc dia chi cho server
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family      = PF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(servIP);
    serv_addr.sin_port        = htons(PORT);

    // Ket noi toi server
    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))<0)
        error("ERROR connecting");

    // Gui du lieu
    while(1){
        printf("message: ");
        bzero(buffer,BUFFSIZE);
        fgets(buffer,BUFFSIZE,stdin);
        if ( send(sockfd,buffer,strlen(buffer)-1,0) < 0)
            error("ERROR writing to socket");
    }
    // Dong client
    close(sockfd);
    return 0;
}
```

Giải thích thêm :

- Vùng code “Ghi cấu trúc địa chỉ cho server ” : sẽ thiết lập thông số server mà bạn muốn kết nối tới, gồm có protocolFamily, địa chỉ và cổng kết nối.

- Vùng code “Gui du lieu” : Sẽ gửi dữ liệu mà bạn gõ trên terminal. Lưu ý khi dùng hàm fgets(), chuỗi ký tự bạn nhập sẽ tự đông được thêm ký tự “\n” ở cuối, và ở đây mình không muốn gửi nó tới server nên thiết lập trong hàm send() “strlen(buffer)-1”.

Chương trình Client trên window.

client-win được sửa lại từ example của MSDN và chạy trên visual studio.

2.2 TCP server

Bước 1 : Tương tự như client, thiết lập socket với thông số sau :

```
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)
```

Bước 2 : Gắn socket vừa tạo với địa chỉ của server

Để client kết nối được tới server thì nhất thiết server phải gắn địa chỉ và port kết nối.

```
int bind(int socket, struct sockaddr *localAddr, unsigned int addrLength)
```

Trong đó :

- LocalAddress : là địa chỉ của server.

- addrLength : là kích thước địa chỉ của server.

Bước 3 : Thiết lập socket chờ kết nối

```
int listen(int socket, int queueLimit)
```

Trong đó

- queueLimit : giới hạn số lượng client đợi kết nối. Nghĩa là khi server đang bận làm việc với clients và không thể làm việc thêm với client nào khác thì client muốn kết nối đó sẽ nằm trong danh sách đợi. Khi server có thể kết nối thêm client sẽ tiếp nhận client trong danh sách đợi đó.

- Trả về -1 nếu có lỗi

Bước 4 : Chấp nhận client

```
int accept(int socket,struct sockaddr *clientAddr,unsigned int *addrLength)
```

Trong đó :

- clientAddr : khi server accept thành công sẽ điền địa chỉ vào clientAddr.
- addrLength : độ dài tối đa của địa chỉ client.
- Trả về là file-descriptor.

Thay vì dùng chính socket do socket() tạo ra để kết nối với client, nó sẽ tạo ra socket mới để kết nối. Đơn giản vì trong nhiều trường hợp kết nối với nhiều client cần những socket khác nhau.

Bước 5 : tương tự như với client.

Chương trình server

```
#include <stdio.h>          /* for printf() and fprintf() */
#include <sys/socket.h>     /* for socket(),connect(),send() and recv() */
#include <stdlib.h>         /* for atoi() and exit() */
#include <string.h>         /* for memset() */
#include <unistd.h>         /* for close() */
#include <sys/types.h>
#include <netinet/in.h>
#define MAXPENDING 5
#define BUFFSIZE 256
#define PORT 8888

void error(const char *msg)
{
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[]){
    int servSock, clntfd, n;
    socklen_t clntLen;
    char buffer[BUFFSIZE];
    struct sockaddr_in serv_addr, cli_addr;

    // Tao socket server
    if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        error("ERROR opening socket");

    // Ghi cau truc dia chi
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(PORT);

    // Bind to the local address
    if (bind(servSock, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR on binding");

    // thiet lap che do cho ket noi cho socket server
    if (listen(servSock, MAXPENDING) < 0)
        error("ERROR on binding");

    // do kich thuoc dia chi cua client
    clntLen = sizeof(cli_addr);

    // cho doi client ket noi, tra ve socket moi ket noi voi client
    if ((clntfd = accept(servSock, (struct sockaddr *) &cli_addr, &clntLen)) < 0)
        error("accept() failed");

    // xoa buffer
    bzero(buffer,BUFFSIZE);
    // xu ly voi client
    while(1){
        n = recv(clntfd,buffer,BUFFSIZE,0);
        if ( n<0 ) error("ERROR reading from socket")
```

Dat |
05/24/2019

Code server cuoi bai bi thieu ha ad?

Hiển thị từ 1 đến 1 trong tổng số 1 (1 trang)

Viết đánh giá

Họ và tên:

Đánh giá của bạn:

Lưu ý: Không hỗ trợ HTML!

Bình chọn: Dở ○ ○ ○ ○ ○ Hay

Nhập mã bảo vệ:

Tiếp tục

Công ty TNHH MLAB
Số chứng nhận kinh doanh: 0106356768. Nơi cấp: Sở kế hoạch và đầu tư Thành Phố Hà Nội. N
Trụ sở : Số 30F9 - Ngõ 104 Lê Thanh Nghị - Hai Bà Trưng - Hà Nội
Email mua bán hàng: smarttechvn.group@gmail.com
Email hỗ trợ kỹ thuật : mlab.services.tech@gmail.com
website:https://mlab.vn
Số điện thoại: 02436231170 hoặc 0984058846 hoặc 0866828846