

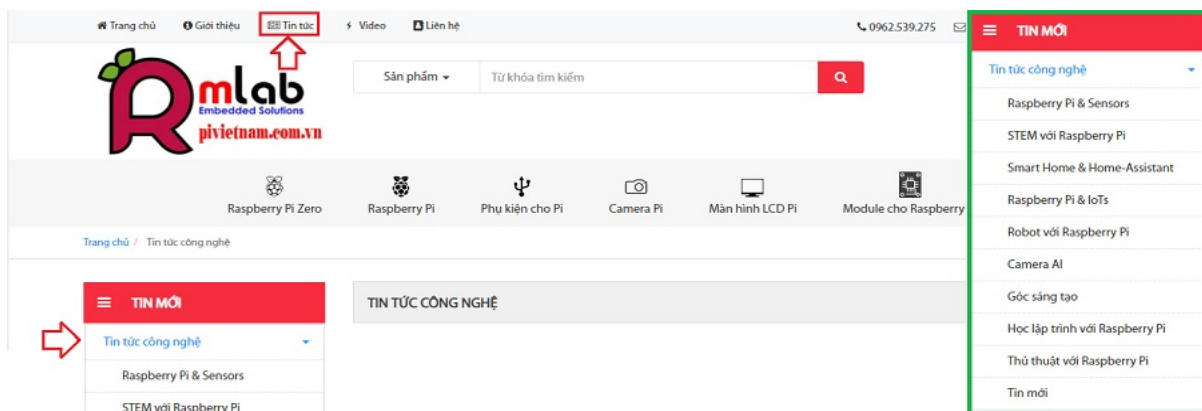
## Bài 6 : Lập trình giao tiếp mạng TCP/IP Raspberry Pi phần 2

**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](http://pivietnam.com.vn)

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



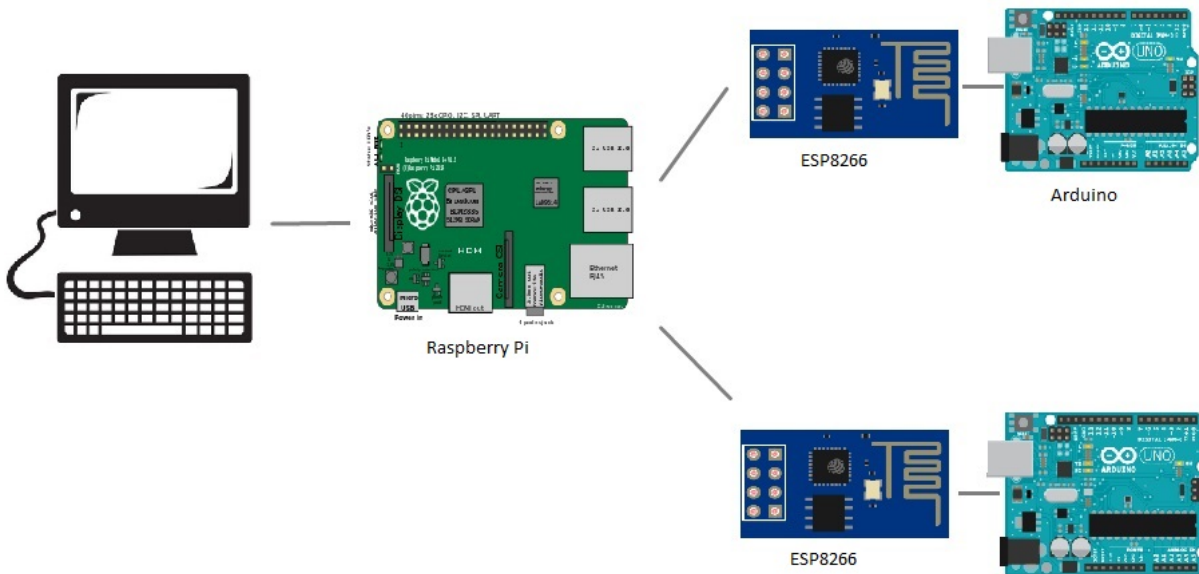
**Các bạn có thể tham khảo các bài viết hỗ trợ kỹ thuật và các tin tức mới nhất tại phần "tin tức" trên website PIVIETNAM.COM.VN**



**Bài viết hỗ trợ kỹ thuật tại website PIVIETNAM.COM.VN - Bài 6: Lập trình giao tiếp mạng TCP/IP Raspberry Pi Phần 2 ([Link here](#))**

Trong phần 1 mình đã trình bày các lý thuyết để xây dựng được mô hình server- client đơn giản với giao thức truyền nhận thông tin là TCP-IP. Các bạn đã có thể dùng Raspberry làm server trung tâm. Máy tính (client) có thể thông gửi và nhận dữ liệu tới Pi mà qua đó có thể gián tiếp điều khiển hoặc xem xét thông tin của các client khác.

Tuy nhiên phần trước server mới chỉ có thể kết nối với duy nhất một client. Các client khác muốn kết nối tới đều phải chờ đợi client đang kết nối kết thúc. Phần tiếp theo đây sẽ trình bày cách giải quyết để server thực sự là server – có thể cùng lúc làm việc với nhiều client.



Chuẩn bị phần cứng :

- Raspberry Pi 2 model B : **thông tin**
- ESP8266 : **thông tin**
- Arduino Uno R3 : **thông tin**

Nền tảng UNIX hỗ trợ nhiều phương thức xử lý khác nhau và chia làm 2 nhánh:

- Multitasking : nó cũng giống như các phần mềm làm việc độc lập với nhau (multiprocess). Mỗi phần mềm khi làm việc sẽ được gọi là process. Hoặc trong cùng một phần có nhiều tác vụ có thể hoạt động độc lập song song ( multithread ). Mỗi tác vụ bên trong sẽ chạy trên 1 thread. Tương ứng với process là `fork()` và thread là `thread()`
- Multiplexing : Cái này sinh ra dành cho mạng. Nó chỉ cần sử dụng duy nhất một thread để làm việc với nhiều client. Tương ứng là `select()` hoặc `poll()`

Mỗi phương thức có nhiều ưu điểm, nhược điểm khác nhau. Trong bài này mình sẽ trình bày phương thức thread-base. Lý do vì nó đơn giản để thực hiện và có thể thỏa mãn nhu cầu về số lượng client lớn.

## 1. Multithread

Trong linux multithread được hỗ trợ bởi thư viện Pthread được nhúng sẵn trong linux. Các bạn không cần phải quan tâm gì thêm mà chỉ cần uống nhanh một tách trà, bật Raspberry lên và bắt tay vào ngay vào code :)

Công việc luôn được bắt đầu bằng việc khai báo thư viện

```
#include <pthread.h>
```

Hàm thiết lập và hủy bỏ thread :

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine) (void *), void *arg);

void pthread_exit(void *retval);
```

Trong đó :

- Thread (thread\_id) : được dùng để đánh ID cho từng thread.
- Attr : được dùng để set thuộc tính cho thread.
- start\_routine : hàm xử lý tương ứng với từng thread. Chính là hàm xử lý client của chúng ta. Hàm này bắt buộc phải là hàm con trỏ void.
- Arg : tham số truyền cho hàm start\_routine.
- Pthread\_create sẽ trả về giá trị 0 nếu thành công , ngược lại sẽ trả về số tương ứng với loại lỗi (error number)
- Retval : giá trị trả về từ hàm pthread\_exit.

Hãy cùng xem ví dụ dưới đây, các bạn nên copy code vào để chạy thử trước :

```
// thread.c
#include <stdio.h>
#include <unistd.h> // for sleep
#include <pthread.h>

// Hàm xử lý cho thread 1
void *Thread1(void *threadid){
    while(1){
        printf("thread 1 : hello\n");
        sleep(3);
    }
}

// Hàm xử lý cho thread 2
void *Thread2(void *threadid){
    while(1){
        printf("thread 2 : hello\n");
    }
}
```

```

        sleep(4);
    }
}

int main () {
    pthread_t threads[NUM_THREADS];
    int rc, i = 0;
    // Tao thread 1
    printf("Creating thread %d, \n",i);
    if ( (rc = pthread_create(&threads[i], NULL, Thread1, NULL)) ){
        printf("Error:unable to create thread, %d\n",rc );
        return 0;
    }
    i++;
    // Tao thread 2
    printf("Creating thread %d, \n",i);
    if ( (rc = pthread_create(&threads[i], NULL, Thread2, NULL)) ){
        printf("Error:unable to create thread, %d\n",rc );
        return 0;
    }
    // Ham main van se chay nhien vu rieng cua no
    while(1){
        printf("main thread : hello\n");
        sleep(2);
    }

    pthread_exit(NULL);
    return 0;
}

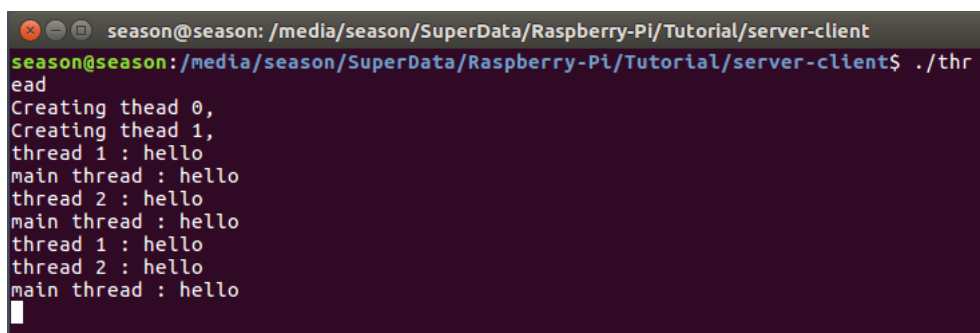
```

Các bạn có thể down code trực tiếp từ [github](#) của mình.

Biên dịch chương trình với lệnh

```
gcc -o thread thread.c -lpthread # -lpthread khai báo dùng thư viện pthread
```

Kết quả sẽ hiển thị như sau



```

season@season: /media/season/SuperData/Raspberry-Pi/Tutorial/server-client
season@season: /media/season/SuperData/Raspberry-Pi/Tutorial/server-client$ ./thread
Creating thread 0,
Creating thread 1,
thread 1 : hello
main thread : hello
thread 2 : hello
main thread : hello
thread 1 : hello
thread 2 : hello
main thread : hello

```

Chương trình sẽ chạy trên 3 luồng riêng biệt là main thread, thread1 và thread2. Mỗi thread sẽ gọi hàm xử lý riêng và sẽ hiển thị câu chào lên màn hình. Tất nhiên là các thread có thể gọi chung một hàm xử lý. Lưu ý rằng hàm xử lý luôn làm hàm con trở dạng void.

Với ứng dụng của thread chương trình có thể hoạt động đa nhiệm không chỉ với server-client mà với bất cứ chương trình nào bạn mong muốn thực hiện đa nhiệm.

## 2. Server – multitasking

Từ ví dụ của phần trước là client trên máy tính kết nối tới Pi. Bây giờ mình sẽ tận dụng luôn từ ví dụ đó để có thể test với server mới. Server mới có thể cùng một lúc nhận nhiều tin nhắn của client và hiển thị lên màn hình, server mới cũng sẽ biết chính xác là client nào gửi thông tin cho mình. Các bạn có thể sửa từ bản server.c cũ như sau :

```

// cau truc struct cho thread, them vao phan dau trong server.c
struct ThreadArgs{
    int cIntSock; /* Socket descriptor for client */
};

// Ham xu ly client, them vao phan dau trong server.c
void *HandleClient(void *threadArgs){
    int cIntSock;
    int recvMsgSize;
    char buffer[BUFFSIZE];
    bzero(buffer,BUFFSIZE);

    cIntSock = ((struct ThreadArgs *) threadArgs) -> cIntSock;

    while(1){
        recvMsgSize = recv(cIntSock,buffer,BUFFSIZE,0);
    }
}

```

```

        if (recvMsgSize < 0)
            error("ERROR reading from socket");
        else if(recvMsgSize>0){
            printf(". Client[%d]: %s\n",cIntSock,buffer);
            bzero(buffer,strlen(buffer));
        }
        else{
            printf("- Client[%d]: disconnected !\n",cIntSock);
            break;
        }
    }
    close(cIntSock);
}

// Thay the tu doan accept den het ham while(1) trong server.c, nhien vu cua phan nay la tao thread khi co thread moi
while(1){

    // Wait for a client to connect
    if( (cIntSock = accept(servSock, (struct sockaddr*) &cli_addr, &cIntLen)) < 0)
        error("accept() failed !");

    getpeername(cIntSock, (struct sockaddr *) &cli_addr, &cIntLen);

    /* Create separate memory for client argument */
    if ((threadArgs = (struct ThreadArgs *) malloc(sizeof(struct ThreadArgs))) == NULL)
        error("malloc() failed");

    threadArgs -> cIntSock = cIntSock;

    if (pthread_create(&threadID, NULL, HandleClient, (void *) threadArgs) != 0)
        error("pthread_create() failed");

    printf("\n+ New client[%d][Addr:%s][Port:%d]\n\n",
        cIntSock, inet_ntoa(cli_addr.sin_addr), ntohs(cli_addr.sin_port));
}

```

Các bạn hãy xem kỹ code trên [github](#) nhé.

Giải thích thêm :

- struct ThreadArgs : Được dùng để lưu thông tin cho thread
- HandleClient() : hàm xử lý tương ứng với từng client
- Chương trình sẽ tạo ra thread tương ứng với từng client mỗi khi có client mới kết nối tới.
- Hàm getpeername() : dùng để lấy thông tin địa chỉ của client.

Biên dịch chương trình với lệnh

```
gcc -o server-thread server-thread.c -lpthread
```

Bây giờ các bạn hãy chạy thử ./server-thread, và bật nhiều client cùng một lúc trên nhiều terminal khác nhau để kết nối tới server.

### 3. Chương trình client với Esp8266

Mình sẽ lấy esp8266 làm client. Các bạn xem thêm phần cài đặt thư viện và lập trình cho esp8266 trên arduino [tại đây](#).

Chương trình cho esp8266 cũng sẽ thực hiện công việc giống hệt client trên máy tính. Nó sẽ gửi tin nhắn đến cho server.

Chương trình như sau :

```

// WifiClientBasic.ino
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>

ESP8266WiFiMulti WiFiMulti;

const uint16_t port = 8888; // port tương ứng với server
const char * host = "192.168.1.23"; // ip của raspberry

void setup() {
    Serial.begin(115200);
    delay(10);

    // Can thay ten wifi và mật khẩu nha ban vào
    WiFiMulti.addAP("ten wifi", "mật khẩu la");

    Serial.println();
    Serial.println();
    Serial.print("Wait for WiFi... ");

    while(WiFiMulti.run() != WL_CONNECTED) {

```

```

        Serial.print(".");
        delay(500);
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());

    delay(500);
}

void loop() {

    Serial.print("connecting to ");
    Serial.println(host);

    // Use WiFiClient class to create TCP connections
    WiFiClient client;

    if (!client.connect(host, port)) {
        Serial.println("connection failed");
        Serial.println("wait 5 sec...");
        delay(5000);
        return;
    }
    while(1){
        if (Serial.available() > 0) {
            String str = Serial.readString();
            // gửi cho server
            client.print(str);

            Serial.print("message: ");
            Serial.println(str);
        }

    }

    //read back one line from server
    //String line = client.readStringUntil('\r');
    //client.println(line);

    //Serial.println("closing connection");
    client.stop();
}

```

Các bạn phải chú ý tới những thiết lập :

- const uint16\_t port = 8888; // thiết lập port tương ứng với server
- const char \* host = "192.168.1.23"; // địa chỉ của Pi
- WiFiMulti.addAP("ten wifi", "mat khau la"); // tên wifi và mật khẩu wifi

#### Kết quả

#### Test mô hình server-client trên Raspberry



Ashe |   
05/04/2019

minh viet server la .c còn client minh code = python thì có được không

trieu ngoc xuan |   
03/15/2018

Hi mlab ! bài viết các bạn rất hay, mình đã chạy trên linux thấy rất OK ! Mình có thắc mắc mong bạn giúp mình : Mình muốn dùng socket client trên C# để điều khiển Rapsberry PI, mọi việc rất OK , nhưng khi mình đóng ứng dụng C# thì trên linux báo server cũng bị ngắt theo: "ERROR reading from socket: Connection reset by peer" Mong bạn có thể hỗ trợ giúp mình ! Xin chân thành cảm ơn bạn

Hiện thị từ 1 đến 2 trong tổng số 2 (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:

8216e8

Tiếp tục

- TRANG CHỦ
- LIÊN HỆ
- CHÍNH SÁCH BẢO HÀNH
- CHÍNH SÁCH BẢO MẬT THÔNG TIN
- CHÍNH SÁCH VẬN CHUYỂN VÀ GIAO NHẬN
- CHÍNH SÁCH ĐỔI TR

Công ty TNHH MLAB  
Số chứng nhận kinh doanh: 0106356768. Nơi cấp: Sở kế hoạch và đầu tư Thành Phố Hà Nội. Ngày cấp: 07/11/2013  
Trụ sở : Số 30F9 - Ngõ 104 Lê Thanh Nghị - Hai Bà Trưng - Hà Nội  
Email mua bán hàng: smarttechn.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