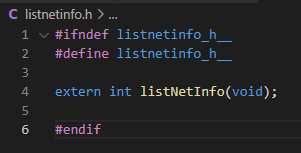
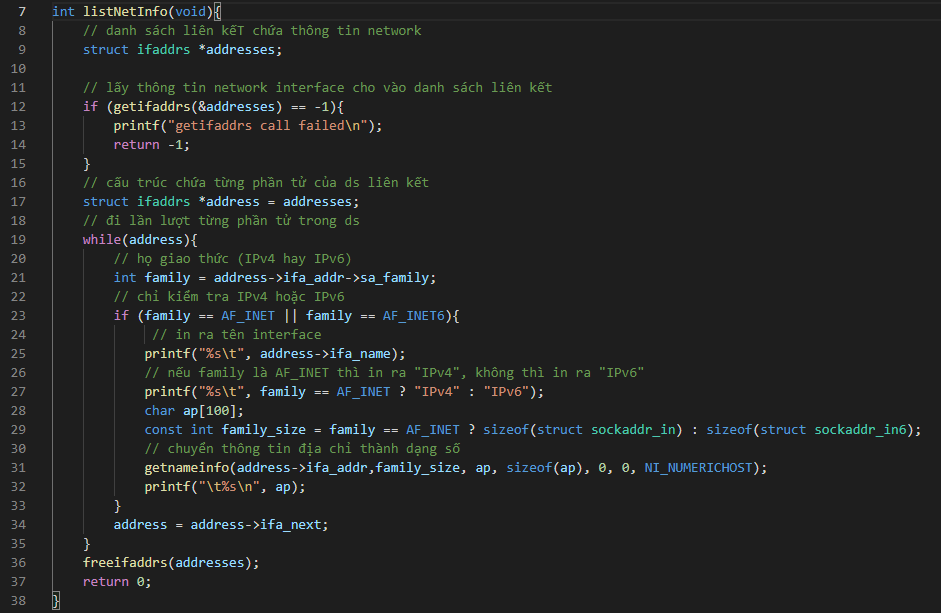
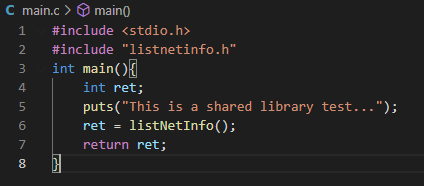
Link chứa toàn bộ mã nguồn của 3 bài: https://drive.google.com/drive/folders/1\_GUFOvmtJ2IeLz6a-n-3Ef0x4Mjunj2S?usp=sharing

Lab 5.1: Tạo dynamic library có chức năng liệt kê thông tin các network interface, chương trình sử dụng dynamic library vừa tạo

1. Tạo 1 file header listnetinfo.h định nghĩa thư viện và nguyên mẫu hàm:  
   
2. Tạo file listnetinfo.c chứa hàm thực thi listNetInfo()  
     
   
3. File chương trình main.c sẽ sử dụng dynamic library đó:  
   
4. Compile mã nguồn thư viện dưới dạng Position Independent Code (PIC) bằng cờ -fpic

gcc -c -Wall -Werror -fpic listnetinfo.c

1. Tạo shared library (file .so) từ file .o vừa compile

gcc -shared -o liblistnetinfo.so listnetinfo.o

1. Cài đặt thư viện:

* Copy file .so vừa tạo vào /usr/lib, và đổi quyền cho nó

sudo cp liblistnetinfo.so /usr/lib

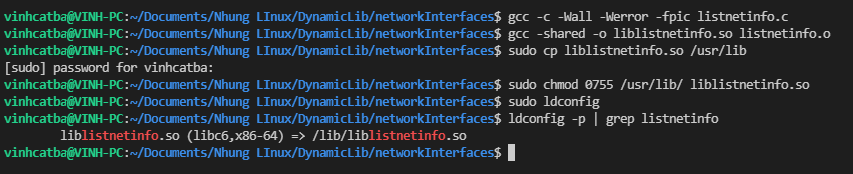
sudo chmod 0755 /usr/lib/ liblistnetinfo.so

* Update cache để có thể sử dụng ngay:

sudo ldconfig

* Kiểm tra xem file .so đó đã có trong hệ thống chưa:

ldconfig -p | grep listnetinfo



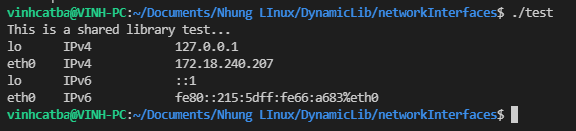
1. Compile chương trình sử dụng thư viện, thêm cờ -llistnetinfo:

gcc -Wall -o test main.c -llistnetinfo

1. chạy chương trình:

./test

Ta thấy có 2 interface là lo và eth0, mỗi interface có 2 family là Ipv4 và Ipv6

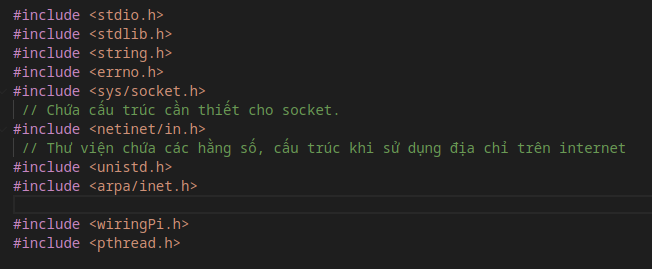


Lab5.2:

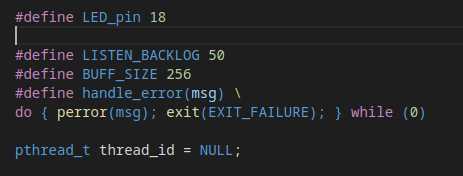
- Ý tưởng: dùng PWM để điều khiển led tăng sáng giảm sáng dần dần. Vì PWM là 1 vòng lặp vô hạn, nên cần dùng pthread để công việc điều khiển PWM có thể chạy song song với chương trình giao tiếp socket. Khi có tín hiệu “on” từ client, thread điều khiển PWM sẽ được tạo. Khi có tín hiệu “off” thread điều khiển PWM sẽ bị hủy bằng hàm pthread\_cancel().

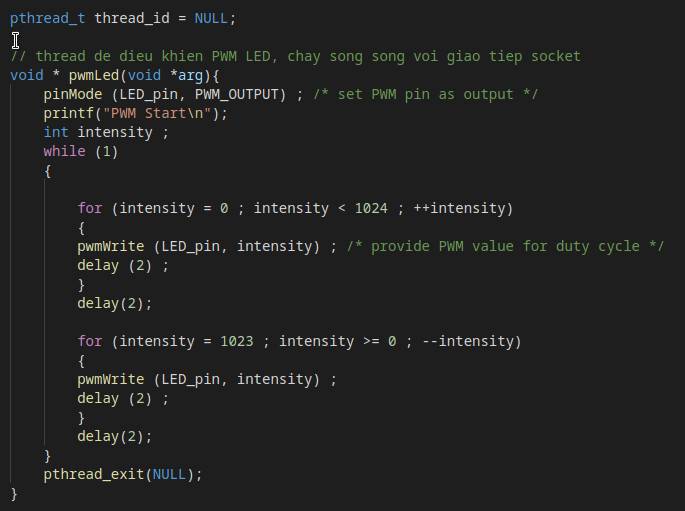
1. Phía server:

- Các thư viện cần thiết

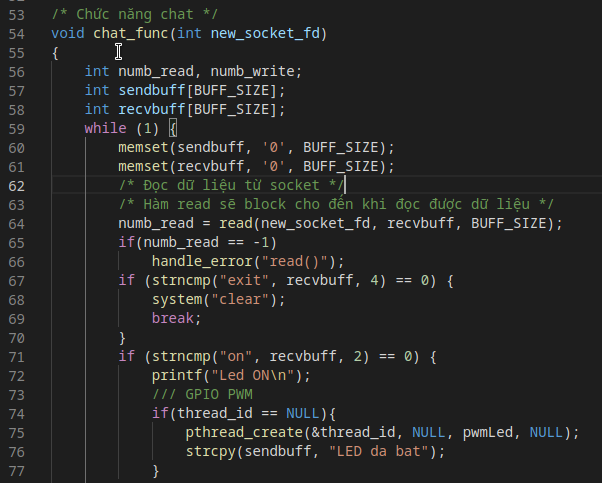


- Định nghĩa chân LED là GPIO18

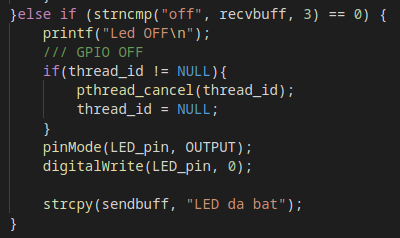


- Hàm điều khiển PWM

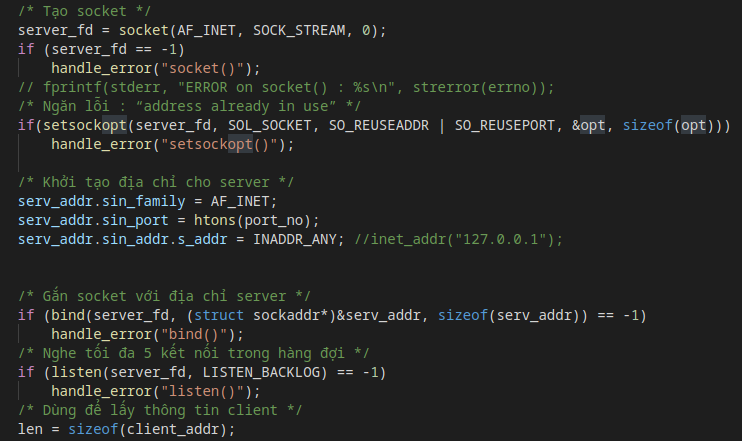
- Chức năng chat: Nếu chuỗi nhận được là “on” thì kiểm tra xem đã có thread được tạo chưa, nếu chưa có thì tạo thread điều khiển PWM.



Nếu chuỗi nhận được là “off” thì kiểm tra xem đã có thread được tạo chưa, nếu có rồi thì hủy thread bằng pthread\_cancel() và đặt lại giá trị thread\_id = NULL. Sau đó đổi chế độ output cho LED\_Pin và đặt 0 để tắt LED.



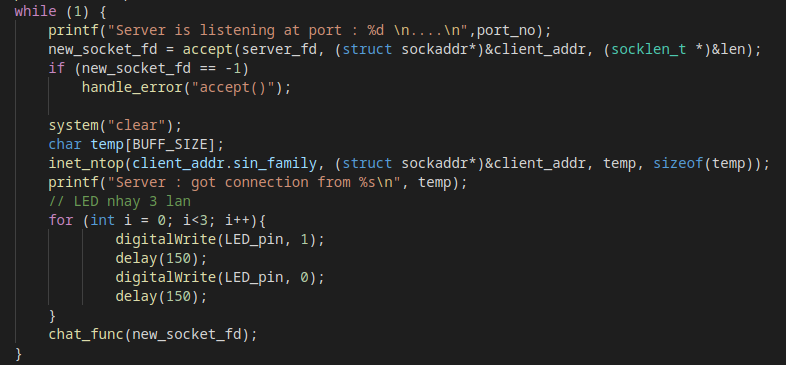
- Trong hàm main(), cấu hình socket ở địa chỉ 0.0.0.0 (INADDR\_ANY) để có thể nhận kết nối từ máy khác cùng dải mạng.



- Khởi tạo GPIO với thư viện wiringPi

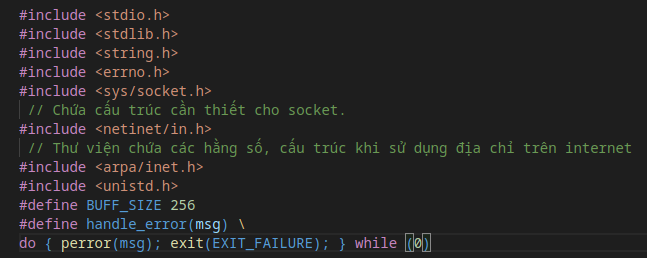


- Sau đó tạo socket, khi có client kết nối thì cho LED bật tắt 3 lần mỗi lần trong 150ms.

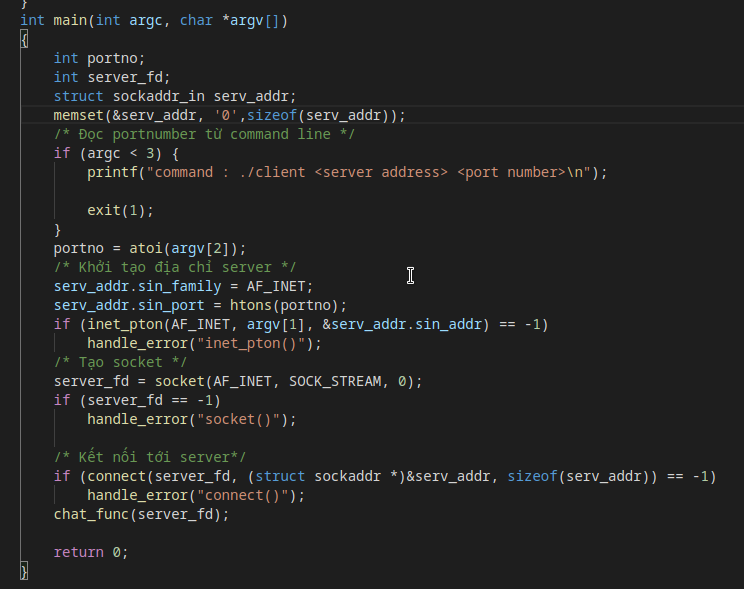


2. Phía client:

- các thư viện và định nghĩa cần thiết:



- Chức năng chat

- hàm main():

3. Nối cực dương của LED với GPIO18, cực âm tới GND

4. Compile server.c trên raspberry pi, với cờ -lpthread và -lwiringPi

gcc -o server server.c -lpthread -lwiringPi

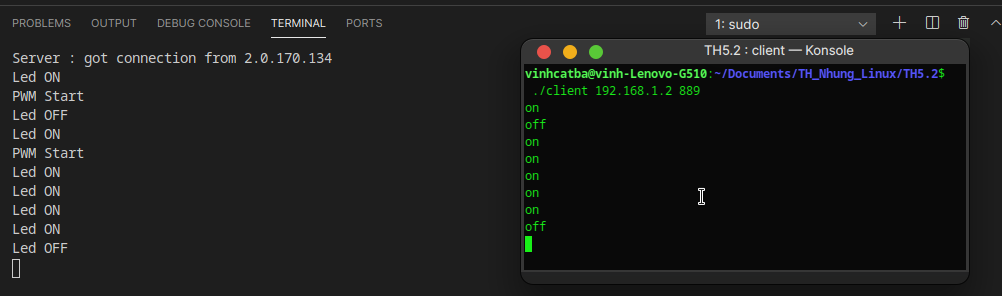
sudo ./server 889

Compile client.c trên máy linux khác và cung cấp địa chỉ ip và port của pi khi chạy

gcc -o client client.c

./client 192.168.1.2 889

- Kết quả: terminal chữ trắng là server trên ras pi, terminal chữ xanh là client trên máy ubuntu. Có thể thấy thread PWM chỉ chạy 1 lần (PWM Start) cho dù có gửi nhiều lệnh “on” liên tiếp.



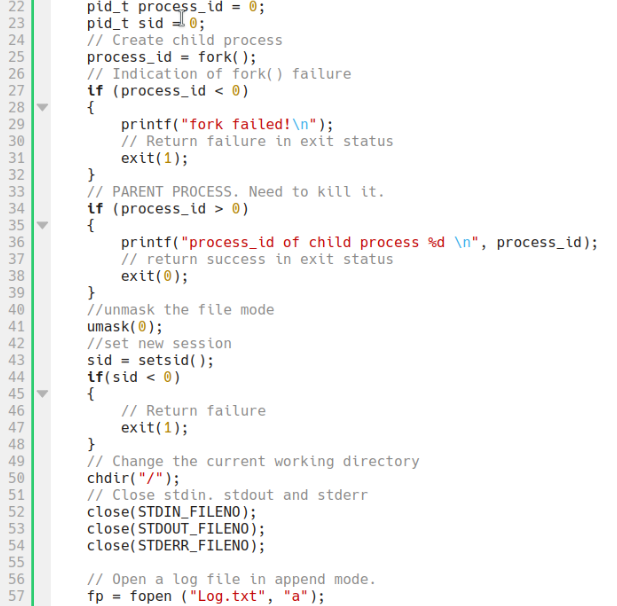
video demo: https://drive.google.com/file/d/11EJYeIe\_6OKQxmXMAewLkojllDEavcL5/view?usp=sharing

TH5.3:

\*Về phía server

Tạo một daemon process bao gồm các bước:

* *Tách khỏi tiến trình gốc (dùng hàm fork())*
* *Đổi quyền tệp bằng umask*
* *Mở file log để ghi nhật ký*
* *Tạo session ID mới (SID)*
* *Đổi thư mục làm việc tới nơi an toàn (thường là thư mục root)*
* *Đóng tất cả các luồng file đang mở từ tiến trình gốc*



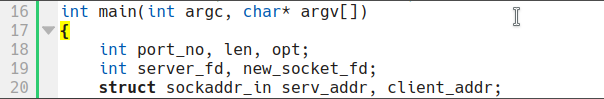
* + Vì daemon process là một tiến trình ngầm nên ở đây cần sử dụng file Log để ghi lại các sự kiện xảy ra. Hàm fopen để mở file Log.txt và quyền a cho phép ghi vào cuối file. File được tạo nếu nó chưa tồn tại.
  + Ở đây chúng em thay đổi thư mục hiện tới / nên file Log.txt được tạo ra tại đó.

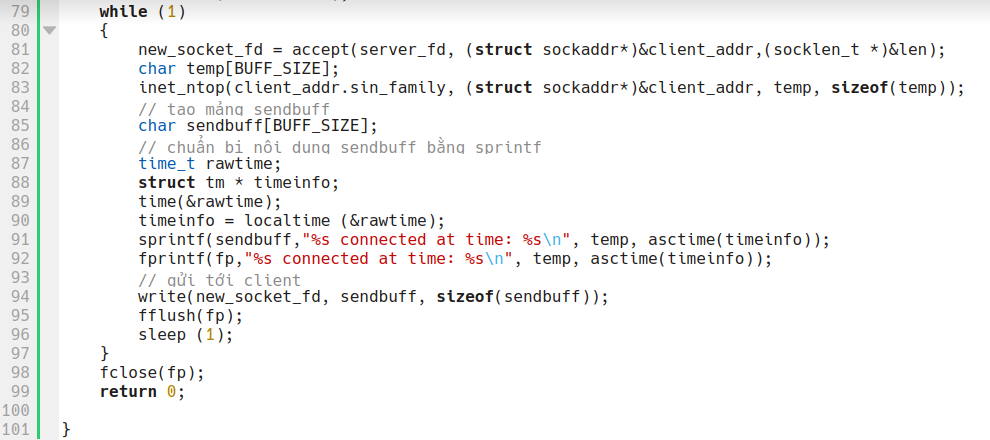
Tiếp theo trong hàm while(1) chứa các function để server và client có thể kết nối được với nhau:

* + *Cố định port kết nối là 888*



* + *Các hàm còn lại:*



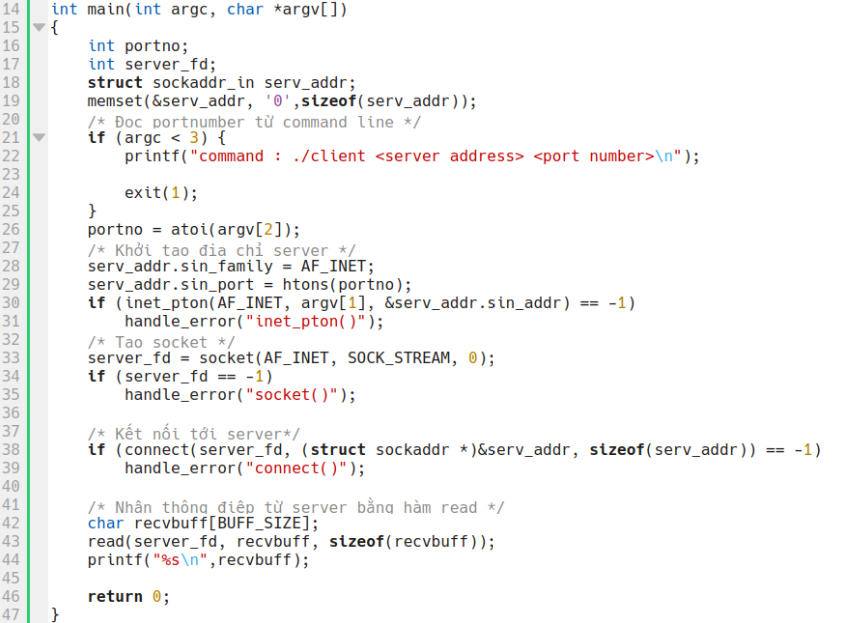


* + Hàm fprintf để ghi thông tin vào file Log.txt, sử dụng hàm này để lại thời gian mỗi khi client kết nối với server.
  + Sử dụng sendbuff và hàm write để gửi chuỗi thời gian hiện tại tới client
  + Cuối cùng là hàm fclose để đóng file Log.txt



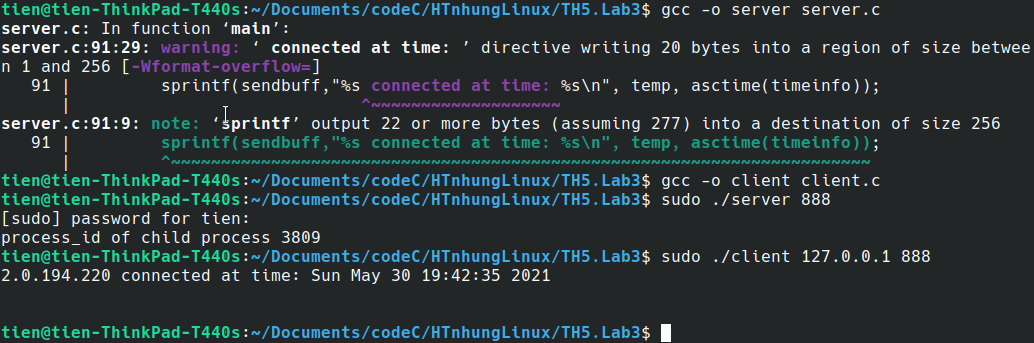
\*Về phía client

Tương tự như bên server, client cũng có các hàm để kết nối vào port của server

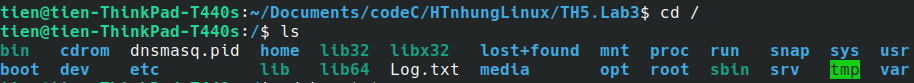


* + Sử dụng hàm read và recvbuff để đọc và nhận thông tin thời gian kết nối từ bên server gửi.

\*Quá trình kết nối và kết quả phía client



*File Log.txt được tạo ra tại thư mục /*



*Thời gian kết nối giữa client và server được in ra trong file*

