Chương 15: Mẹo và gợi ý

Mọi người đều bắt đầu từ một điểm nào đó, bao gồm cả những chuyên gia. Chương này chứa một số mẹo và gợi ý dành cho các lập trình viên Arduino mới bắt đầu. Arduino được phát triển dành cho sinh viên mới bước chân vào lĩnh vực lập trình nhúng, giúp họ bắt đầu làm quen mà không bị quá tải bởi các chi tiết kỹ thuật. Vì một số người đam mê và những người chơi Arduino có thể thiếu kiến thức đào tạo bài bản về phần mềm, nên có một vài điều đáng lưu ý "một cách tình cờ" mà hầu hết các nhà thực hành dày dạn kinh nghiệm coi là hiển nhiên.

Diễn đàn: Hãy đầu tư chút nỗ lực

Các diễn đàn thường xuyên có các bài đăng nhờ giúp đỡ phát triển một thứ gì đó phức tạp. Yêu cầu thường dưới dạng: "Tôi muốn làm... Tôi là người mới – ai đó có thể giúp tôi không?" Điều này không phải là tội lỗi khi không nhận thức được sự phức tạp và việc là người mới cũng không sao. Nhưng bạn đã mất bao lâu để gõ yêu cầu này? Mười giây? Vậy còn những người trả lời sẽ mất bao nhiêu nỗ lực? Có thể cũng chỉ mười giây, nếu bạn nhận được câu trả lời.

Mọi người sẵn sàng giúp đỡ hơn nếu họ thấy rằng bạn đã đầu tư một chút nỗ lực của riêng mình. Bạn đã trình bày những gì bạn nghĩ là cần thiết và cách bạn nghĩ sẽ thực hiện chưa? Bạn sai cũng không sao, vì điều đó cho thấy rằng bạn không chỉ đơn thuần ném vấn đề qua tường và hy vọng ai đó sẽ làm hết phần việc cho mình.

Bắt đầu nhỏ

Đôi khi, mọi người yêu cầu giúp lập trình những bài tập lớn và phức tạp. Phản hồi thông thường cho những bài đăng này là không có hồi đáp nào cả. Không phải vì không ai biết câu trả lời, mà vì không ai muốn bỏ công sức để hướng dẫn bạn từ đầu về những điều cần phải học trước. Nếu bạn phải tiếp tục với bài tập đó, hãy chia nhỏ nó thành các phần nhỏ hơn. Sau đó, đặt những câu hỏi cụ thể về những khó khăn bạn đang gặp phải.

Tuy nhiên, cách tốt nhất là bắt đầu với những bài tập nhỏ và đơn giản hơn. Ai cũng biết điều này, nhưng những người đam mê thường mất kiên nhẫn. Bạn thuộc kiểu người chỉ muốn "câu trả lời" hay muốn biết cách tự tìm ra câu trả lời? Kinh nghiệm là người thầy tốt nhất. Có lý do vì sao người ta thường bắt đầu với việc làm

nhấp nháy đèn LED. Đèn LED rất đơn giản – chúng chỉ bật hoặc tắt. Nhưng ngay cả khi đơn giản như vậy, vẫn có những bài học cần rút ra.

Chẳng hạn, nếu đèn LED không sáng, nguyên nhân là gì? Nếu bạn chưa từng gặp phải điều đó trước đây, bạn có thể không nhận ra rằng đèn LED đã được đấu nối sai cực. Đừng tự tước đi những khoảnh khắc học hỏi quý giá này của chính mình.

Phương pháp Hợp đồng Chính phủ

Các lập trình viên mới, với kiến thức về ngôn ngữ lập trình của mình, thường lao vào viết toàn bộ chương trình ngay lập tức và sau đó cố gắng sửa lỗi. Tôi thích gọi cách này là **phương pháp hợp đồng chính phủ**. Một khi đã nắm được yêu cầu, phần mềm sẽ được viết ra và thử nghiệm ở giai đoạn cuối. Các phiên sửa lỗi sau đó có thể gây ra sự rối loạn không nhỏ.

Đừng hiểu lầm – các yêu cầu là quan trọng. Nhưng khi yêu cầu đến từ chính bản thân chúng ta (trong các dự án đam mê), chúng thường xuyên thay đổi. Hoặc nếu bạn đang phát triển thứ gì đó chỉ để vui, bạn có thể thậm chí không có những yêu cầu cụ thể. Đây chỉ là một vài lý do để tránh việc viết toàn bộ mã trước khi kiểm tra.

Lý do lớn nhất để tránh phương pháp hợp đồng chính phủ là việc sửa lỗi trên các thiết bị nhúng khó khăn hơn nhiều. Có thể không có trình gỡ lỗi nào sẵn có hoặc khả năng theo dõi của nó bị giới hạn. Bạn, chẳng hạn, không thể bước qua một chương trình xử lý ngắt. Một cách tiếp cận tốt hơn là bắt đầu từ những bước nhỏ và sử dụng phương pháp **basic shell** và **stub**.

Khung cơ bản (Basic Shell)

Thay vì cố gắng viết toàn bộ ứng dụng trước khi sửa lỗi, hãy viết một khung cơ bản trước. Trong khung chương trình này, hãy viết một hàm setup() và loop() tối giản cho mã Arduino của bạn. Đặc biệt khi sử dụng một bảng phát triển như ESP32, hãy tận dụng liên kết USB-to-Serial bằng **Serial Monitor**. Điều này sẽ đơn giản hóa chu trình phát triển và thử nghiệm của bạn.

Khung đầu tiên có thể chỉ đơn giản là "Hello from setup()" và "Hello from loop()" trong hàm loop, như minh họa trong Listing 15-1. Hãy nhớ rằng chương trình đầu tiên này không cần phải hoàn hảo. Với việc thử nghiệm tối thiểu như thế này, bạn chứng minh được toàn bộ quy trình **biên dịch**, **nạp mã**, và **kiểm tra chạy**.

Lệnh delay() ở dòng 4 cho phép các thư viện của ESP32 thiết lập liên kết USB-to-Serial trước khi chương trình tiếp tục chạy. Một phần của việc chứng minh tính khả thi là chỉ cần xác nhận rằng liên kết gỡ lỗi qua **Serial Monitor** đã được thiết lập.

```
0001: // basicshell.ino
0002:
0003: void setup() {
0004: delay(2000); // Allow for serial setup
0005: printf("Hello from setup()\n");
0006: }
0007:
0008: void loop() {
0009: printf("Hello from loop()\n");
0010: delay(1000);
0011: }
```

Liệt kê 15-1. Một mẫu shell khởi đầu cơ bản của một chương trình.

Cách tiếp cận sơ khai

Rõ ràng, bạn muốn ứng dụng của mình làm được nhiều hơn là chỉ có khung cơ bản. Hãy bắt đầu xây dựng dựa trên khung đó bằng cách thêm các hàm stub (Danh sách 15-2). Các hàm init_oled() và init_gpio() chỉ là các bản nháp cho các hàm khởi tạo sau này sẽ dành cho thiết bị OLED và GPIO.

Hãy biên dịch, nạp mã, và kiểm tra những gì bạn đã có. Nó có chạy không? Kết quả đầu ra từ Danh sách 15-2 sẽ trông như sau trên **Serial Monitor**: Hello from setup()

```
init_oled() called.
init_gpio() called.
Hello from loop()
Hello from loop()
```

Hello from loop()

...

Bước tiếp theo là mở rộng những gì các hàm stub thực hiện – tức là khởi tạo thiết bị thực sự. Hãy tránh cám dỗ để làm mọi thứ một lúc và giữ cho các bổ sung mã nhỏ gọn và dần dần. Điều này sẽ giúp bạn tránh được những khoảnh khắc vò đầu bứt tai khi xuất hiện vấn đề mới. Thật đáng kinh ngạc khi ngay cả những bổ sung nhỏ, tưởng chừng hiển nhiên cũng có thể gây ra rất nhiều rắc rối.

```
0001: // stubs.ino
0002:
0003: static void init_oled() {
0004: printf("init_oled() called.\n");
0005: }
0006:
0007: static void init_gpio() {
0008: printf("init_gpio() called.\n");
0009: }
0010:
0011: void setup() {
0012: delay(2000); // Allow for serial setup
0013: printf("Hello from setup()\n");
0014: init_oled();
0015: init_gpio();
0016: }
0017:
0018: void loop() {
```

```
0019: printf("Hello from loop()\n");
0020: delay(1000);
0021: }
```

Liệt kê 15-2. Chương trình shell cơ bản được mở rộng với các sơ khai.

Sơ đồ khối

Các ứng dụng lớn hơn có thể hưởng lợi từ việc sử dụng sơ đồ khối để lên kế hoạch cho các tác vụ FreeRTOS cần thiết. Các hàm setup() và loop() bắt đầu từ "loopTask", được cung cấp mặc định. Nếu bạn không thích cách phân bổ stack của loopTask, bạn có thể xóa tác vụ đó bằng cách gọi vTaskDelete(nullptr) từ loop() hoặc từ bên trong setup(). Bạn cần thêm những tác vụ nào? Các chương trình xử lý ngắt (ISR) có cung cấp sự kiện cho tác vụ nào không? Hãy vẽ các đường biểu thị nơi có thể có hàng đợi thông điệp (message queue). Có thể sử dụng các đường nét đứt để thể hiện sự tham gia của các sự kiện, semaphore, hoặc mutex giữa các tác vụ. Sơ đồ không nhất thiết phải được chấp thuận theo UML – chỉ cần sử dụng các quy ước phù hợp với bạn.

Lỗi

Khi bạn xây dựng ứng dụng của mình từng phần một, bạn có thể đột nhiên gặp phải một lỗi chương trình nào đó. Điều này có thể gây khó chịu nhất là trong một ứng dụng đã hoàn thiện. Nhưng vì bạn đang phát triển ứng dụng của mình bằng cách thêm từng phần mã một, bạn đã biết phần mã nào vừa được thêm vào. Lỗi có thể liên quan đến phần mã vừa thêm vào. Các tùy chọn biên dịch của Arduino không cho phép trình biên dịch cảnh báo về mọi thứ mà nó nên cảnh báo. Hoặc có thể là cách tệp header của hỗ trợ printf() của newlib được định nghĩa. Dù sao đi nữa, một ví dụ về mã có thể dẫn đến lỗi là:

```
printf("The name of the task is ‘%s’\n");
```

Thấy vấn đề chưa? Phải có một đối số chuỗi C (C string) sau chuỗi định dạng để đáp ứng tham số định dạng "%s". Hàm printf() sẽ mong đợi điều đó và tìm trong ngăn xếp (stack) để lấy giá trị. Nhưng giá trị mà nó tìm thấy có thể là dữ liệu rác hoặc con trỏ null (nullptr), dẫn đến lỗi. Trình biên dịch biết về những vấn đề này, nhưng vì lý do nào đó, các cảnh báo này không được báo cáo.

Một nguồn gây lỗi phổ biến khác là **hết không gian ngăn xếp**. Nếu bạn không thể xác định ngay nguyên nhân gây ra lỗi, hãy phân bổ thêm không gian ngăn xếp cho tất cả các tác vụ được thêm vào. Điều này có thể giúp loại bỏ lỗi. Sau khi hoàn thành thử nghiệm, bạn có thể giảm cẩn thận các phân bổ ngăn xếp khác nhau.

Cũng cần lưu ý đến vấn đề về **thời gian sống của đối tượng (object lifetimes)**. Bạn có gửi một con trỏ (pointer) qua hàng đợi (queue) không? Hãy xem phần "Hiểu rõ vòng đời lưu trữ của bạn". Hoặc đối tượng C++ đã bị hủy (destructed) khi một tác vụ khác cố gắng truy cập vào nó?

Hiểu về Thời gian Sống của Bộ Nhớ

Khi bạn bắt đầu, có vẻ như có rất nhiều thứ cần phải học. Đừng để điều đó làm bạn nản lòng, nhưng hãy xem xét đoạn mã sau:

```
static char area1[25];

void function foo() {

char area2[25];
```

Bộ nhớ cho mảng area1 được tạo ra ở đâu? Nó có giống như area2 không? Mảng area1 được tạo ra trong một khu vực của SRAM được phân bổ vĩnh viễn cho mảng đó. Bộ nhớ này sẽ không bao giờ biến mất.

Tuy nhiên, bộ nhớ cho area2 lại khác vì nó được phân bổ trên stack. Ngay khi hàm foo() trả về, bộ nhớ đó sẽ bị giải phóng. Nếu bạn truyền con trỏ đến area2 qua một message queue, ví dụ, con trỏ đó sẽ không còn hợp lệ ngay khi foo() trả về.

Nếu bạn muốn mảng area2 tồn tại sau khi foo() trả về, bạn có thể khai báo nó là static trong hàm:

```
static char area1[25];

void function foo() {

static char area2[25];
```

Bằng cách thêm từ khóa static vào khai báo area2, chúng ta đã chuyển bộ nhớ phân bổ của nó sang cùng khu vực với area1 (tức là không phải trên stack).

Lưu ý rằng mảng area1 cũng được khai báo với thuộc tính static, nhưng trong trường hợp này, từ khóa static mang một ý nghĩa khác (khi khai báo bên ngoài hàm). Bên ngoài hàm, từ khóa static có nghĩa là không gán một ký hiệu bên ngoài cho nó ("area1"). Việc khai báo các biến này với static giúp tránh xung đột trong bước liên kết.

Tránh Sử Dụng Tên Ngoài

Các hàm và các mục bộ nhớ toàn cục chỉ được tham chiếu trong tệp nguồn hiện tại của bạn nên được khai báo là static. Nếu không có từ khóa static, tên biến hoặc hàm sẽ trở thành "extern" và có thể gây xung đột với các thư viện khác khi liên kết. Trừ khi hàm hoặc biến toàn cục của bạn cần phải là extern, hãy khai báo chúng là static.

Các hàm setup() và loop(), mặt khác, phải là các biểu tượng extern vì liên kết viên (linker) phải gọi chúng từ mô-đun khởi động ứng dụng. Việc sử dụng extern cho phép linker tìm và liên kết với chúng.

Tận Dụng Phạm Vi

Một thực hành tốt trong phần mềm là giới hạn phạm vi của các thực thể để chúng không thể bị nhầm lẫn hoặc tham chiếu từ những nơi không nên. Khai báo tất cả mọi thứ ở phạm vi toàn cục là thuận tiện cho các dự án nhỏ nhưng có thể trở thành cơn ác mộng cho các ứng dụng lớn hơn. Tôi thích gọi đây là phong cách lập trình "cowboy". Những lập trình viên còn nhớ COBOL sẽ hiểu điều này.

Vấn đề với phong cách lập trình cowboy là nếu bạn gặp phải lỗi khi cái gì đó bị sử dụng/thay đổi khi không nên, việc cô lập nguyên nhân sẽ rất khó khăn. Khi thay vào đó bạn sử dụng các quy tắc phạm vi của ngôn ngữ, trình biên dịch sẽ thông báo ngay cho bạn khi bạn cố gắng truy cập vào những thứ không nên. Quyền truy cập hợp pháp sẽ được thi hành.

Một cách để giới hạn phạm vi của các handle trong FreeRTOS và các mục dữ liệu khác là truyền chúng vào các tác vụ dưới dạng các thành viên của một cấu trúc. Ví dụ, nếu một tác vụ cần handle của một hàng đợi và một mutex, thì hãy truyền các mục này trong một cấu trúc đến tác vụ khi tạo tác vụ. Khi đó, chỉ có tác vụ sử dụng mới biết về các handle này.

Thư giãn bộ não

Khi nói đến trải nghiệm con người, các nhà tâm lý học cho biết có ít nhất 16 loại tính cách khác nhau (Myers-Briggs). Tuy nhiên, tôi tin rằng hầu hết mọi người sẽ tiếp tục làm

việc với một vấn đề ngay cả khi họ đã ngừng suy nghĩ về nó một cách có ý thức. Vì vậy, khi bạn cảm thấy mất kiên nhẫn trong một phiên gỡ lỗi vào đêm khuya, hãy cho phép bản thân nghỉ ngơi. Điều này cũng có thể giúp bạn tránh được những rủi ro không cần thiết, có thể dẫn đến "khói ma thuật" (lỗi phần cứng).

Một số người có thể ngủ ngon, trong khi những người khác sẽ trằn trọc suốt đêm. Nhưng tâm trí vẫn suy ngẫm về những sự kiện trong ngày và lướt qua tất cả các kịch bản có thể xảy ra. Sáng hôm sau, vợ/chồng bạn có thể phàn nàn về những tiếng lầm bầm bằng hệ thập lục phân của bạn khi ngủ. Nhưng khi bạn thức dậy, bạn sẽ thường có những ý tưởng mới để thử. Nếu đó là một vấn đề đặc biệt khó khăn, khoảnh khắc "eureka" có thể mất vài ngày để phát triển. Bạn sẽ vượt qua.

Sổ tay ghi chú

Khi đầu bạn chạm vào gối vào ban đêm, bạn có thể đột nhiên nhớ ra một hoặc hai điều mà bạn đã quên không giải quyết trong mã nguồn hoặc mạch điện. Một cuốn sổ tay đặt bên giường có thể là một công cụ hỗ trợ trí nhớ hữu ích. Khi còn trẻ, tâm trí bạn ít bị rối loạn, và việc nhớ những thứ cần làm là dễ dàng. Tuy nhiên, khi tuổi tác tăng lên, cuộc sống trở nên phức tạp và những thứ sẽ bắt đầu bị bỏ sót. Một cuốn sổ tay rất hữu ích để ghi lại những gì đã thử hoặc cách bạn giải quyết một vấn đề. Những buổi sáng thứ Hai tại nơi làm việc sẽ thuận tiện hơn khi bạn có thể tiếp tục công việc từ nơi bạn đã dừng lại vào thứ Sáu tuần trước.

Trừ khi bạn sử dụng một kỹ thuật đặc biệt nào đó thường xuyên, bạn sẽ cần phải tra cứu lại. Đây là một cách khác mà việc ghi chép có thể hữu ích. Hãy ghi lại các API đặc biệt, kỹ thuật C++ hoặc các thứ liên quan đến FreeRTOS mà bạn thấy hữu ích. Nếu bạn thích khả năng sao chép và dán, hãy sử dụng các trang web như evernote.com. Những công cụ này có lợi thế là có thể tìm kiếm trực tuyến.

Yêu Cầu Giúp Đỡ

Kể từ khi "mạng lưới không thể kiểm soát" xuất hiện, chúng ta đã có sự may mắn khi có các diễn đàn và công cụ tìm kiếm, cho phép chúng ta "google" để tìm kiếm sự trợ giúp. Tìm kiếm trên web thường là bước đầu tiên hữu ích để có câu trả lời ngay lập tức hoặc manh mối. Tuy nhiên, hãy tiếp nhận những gì bạn đọc với một chút hoài nghi – không phải tất cả lời khuyên đều tốt. Tùy thuộc vào bản chất của vấn đề, bạn sẽ thường xuyên phát hiện ra rằng

những người khác cũng đã gặp phải vấn đề tương tự. Trong trường hợp đó, bạn có thể nhận được một hoặc nhiều câu trả lời đã được đăng để tham khảo.

Khi yêu cầu trợ giúp trên diễn đàn, hãy đặt câu hỏi thông minh. Các bài đăng như "I2C của tôi không hoạt động, bạn có thể giúp không?" thể hiện rất ít sự chủ động. Đây là kiểu nỗ lực "ném vấn đề qua tường và hy vọng điều tốt nhất". Bạn có đưa xe của mình đến gara và chỉ nói rằng xe của bạn bị hỏng không? Các bài đăng trên diễn đàn không nên cần phải chơi trò hỏi đáp 20 câu hỏi.

Hãy đăng câu hỏi của bạn với một số thông tin cụ thể:

- Bản chất chính xác của vấn đề (phần nào của I2C không "hoạt động")
- Bạn đang làm việc với các thiết bị I2C nào?
- Bạn đã thử những gì cho đến nay?
- Có thể là thông tin cụ thể về bảng phát triển ESP32 của bạn.
- Nền tảng phát triển – Arduino hay ESP-IDF?
- Bạn đang sử dụng thư viện nào, nếu có?
- Bất kỳ điều gì kỳ lạ khác mà bạn quan sát được.

Tôi khuyên bạn không nên đăng mã nguồn trong bài đăng đầu tiên, nhưng hãy sử dụng phán đoán tốt nhất của bạn. Một số người đăng rất nhiều mã như thể điều đó sẽ tự giải thích. Tôi tin rằng việc mô tả bản chất vấn đề trước sẽ hiệu quả hơn. Bạn luôn có thể đăng mã sau khi giải thích vấn đề.

Khi đăng mã, đôi khi không cần thiết phải đăng tất cả mã (đặc biệt khi mã dài). Đôi khi chỉ cần đăng những phần mã có khả năng góp phần vào vấn đề. Trong ví dụ của chúng ta, bạn có thể chỉ cần đăng các hàm mã I2C đã sử dụng.

Diễn đàn thường có cách để đăng "mã" trong tin nhắn (như [code]...[/code]). Hãy chắc chắn sử dụng tính năng này khi có thể. Nếu không, với kiểu chữ không có tỷ lệ và thiếu sự tôn trọng đối với thụt lề, mã sẽ trở thành một mớ hỗn độn rất khó đọc. Tôi ghét đọc mã có thụt lề tồi tệ.

Có một tác dụng phụ có lợi khi mô tả chính xác vấn đề, dù là trong bài đăng hay qua email – khi bạn hoàn thành việc mô tả vấn đề, có thể bạn đã nhận ra câu trả lời rồi. Ngoài ra,

khi làm việc với đồng nghiệp hoặc bạn học, chỉ việc giải thích vấn đề cho họ cũng có thể mang lại kết quả tương tự.

Chia Để Chinh Phục

Sinh viên mới có thể gặp khó khăn khi ứng dụng bị treo. Làm sao để bạn cách ly được phần mã gây ra vấn đề? Lập trình viên dày dặn kinh nghiệm sẽ biết kỹ thuật chia để trị.

Khái niệm này đơn giản như trò chơi đoán số. Nếu bạn phải đoán một con số mà tôi đang nghĩ đến trong khoảng từ 1 đến 10, và bạn đoán số 6, tôi trả lời rằng số đó thấp hơn, thì bạn sẽ chia phạm vi đó ra với một phán đoán là 3 chẳng hạn. Cuối cùng, bạn sẽ có thể đoán chính xác con số bằng cách giảm dần phạm vi qua mỗi lần thử. Khi một chương trình bị treo, bạn chia nó thành hai nửa cho đến khi cô lập được phần mã gây lỗi.

Các phương pháp khác nhau có thể được sử dụng làm chỉ báo – có thể là việc in ra Serial Monitor hoặc kích hoạt một LED. LED rất hữu ích cho việc theo dõi ISR, nơi mà bạn không thể in thông báo. Nếu bạn có đủ chân GPIO, bạn thậm chí có thể sử dụng một LED hai màu để báo hiệu các sự kiện khác nhau. Ý tưởng là chỉ ra rằng mã đã được thực thi tại các điểm quan trọng. Nếu bạn cần nhiều hơn từ LED, bạn có thể nhấp nháy mã khi không ở trong ISR. Khi bạn đã thu hẹp được vùng mã nơi xảy ra vấn đề, bạn có thể xem xét kỹ lưỡng hơn để tìm ra nguyên nhân.

Lập Trình Để Có Câu Trả Lời

Tôi đã thấy lập trình viên trong công ty tranh cãi cả nửa giờ về việc điều gì sẽ xảy ra khi một sự kiện nào đó xảy ra. Ngay cả sau đó, tranh cãi cũng thường không có hồi kết. Toàn bộ vấn đề có thể được giải quyết nhanh chóng bằng cách viết một chương trình đơn giản trong một phút để kiểm tra giả thuyết. Dĩ nhiên, hãy sử dụng một chút thông minh với những gì bạn đã quan sát:

- Hành vi quan sát được có được API hỗ trợ không?
- Hay hành vi này là do sử dụng sai API hoặc khai thác một lỗi?

Nếu API là mã nguồn mở, thì mã nguồn thường là câu trả lời cuối cùng. Thường thì mã nguồn và các chú thích sẽ chỉ ra ý định của các giao diện tài liệu kém. Kết luận: đừng ngại viết mã thử nghiệm, dù chỉ là mã tạm thời.

Tận Dụng Lệnh Find

Khi kiểm tra mã nguồn mở, bạn có thể tìm kiếm nó trực tuyến hoặc xem xét những gì bạn đã cài đặt trên hệ thống của mình. Việc xem mã đã cài đặt rất quan trọng khi bạn nghĩ rằng mình đã tìm thấy một lỗi trong thư viện mà bạn đang sử dụng. Một trong những nhược điểm của Arduino là nhiều thứ được thực hiện trong "hậu trường" và vẫn ẩn đi với người học. Nếu bạn đang sử dụng hệ thống POSIX (Linux, FreeBSD, MacOS, v.v.), lệnh find sẽ cực kỳ hữu ích. Người dùng Windows có thể cài đặt WSL (Windows Subsystem for Linux) để thực hiện tương tự hoặc sử dụng phiên bản Windows của lệnh này.

Hãy dành thời gian để làm quen với lệnh find. Nó rất mạnh mẽ và có thể khiến người mới bắt đầu cảm thấy khó khăn, nhưng không có gì là bí ẩn. Chỉ là rất nhiều sự linh hoạt mà bạn có thể tiếp thu theo từng bước nhỏ. Lệnh find hỗ trợ vô số tùy chọn khiến nó có vẻ phức tạp. Hãy cùng xem xét những tùy chọn quan trọng và hữu ích nhất trong số đó. Định dạng cơ bản của lệnh như sau:

find [options] path1 path2 ... [expression]

Các tùy chọn trong dòng lệnh dành cho người dùng nâng cao, và chúng ta có thể bỏ qua chúng ở đây. Một hoặc nhiều tên thư mục là nơi bạn muốn bắt đầu tìm kiếm. Để có kết quả, trước đây cần phải chỉ định tùy chọn -print cho phần biểu thức, nhưng với lệnh find của Gnu, điều này bây giờ được mặc định:

$ find basicshell stubs -print

HJặc chỉ cần:

$ find basicshell stubs

basicshell

basicshell/basicshell.ino

stubs

stubs/stubs.ino

Với hình thức lệnh trên, tất cả các tên thư mục và tệp con từ các thư mục đã cho sẽ được liệt kê, bao gồm cả thư mục và tệp. Hãy giới hạn kết quả chỉ với các tệp bằng cách sử dụng tùy chọn -type với tham số "f" (chỉ tệp):

```
$ find basicshell stubs -type f
basicshell/basicshell.ino
stubs/stubs.ino
```

Bây giờ, kết quả chỉ hiển thị đường dẫn tệp. Tuy nhiên, kết quả này vẫn chưa thực sự hữu ích. Điều chúng ta cần làm là yêu cầu lệnh find thực hiện một hành động với các tên tệp này. Lệnh grep là một ứng cử viên tuyệt vời:

```
$ find basicshell stubs -type f -exec grep ‘setup’ {} \;
void setup() {
delay(2000); // Allow for serial setup
printf("Hello from setup()\n");
void setup() {
delay(2000); // Allow for serial setup
printf("Hello from setup()\n");
```

Chúng ta đã gần hoàn tất, nhưng trước tiên, hãy giải thích một vài điều. Chúng ta đã thêm tùy chọn exec vào lệnh find, theo sau là tên lệnh (grep) và một số cú pháp đặc biệt. Tham số 'setup' là biểu thức chính quy trong grep mà chúng ta đang tìm kiếm (hoặc một chuỗi đơn giản). Thông thường, bạn sẽ phải đặt trong dấu nháy đơn để tránh shell can thiệp vào nó. Tham số "{}" chỉ ra nơi trên dòng lệnh để chuyển đường dẫn tệp (cho lệnh grep). Cuối cùng, ký tự \; đánh dấu sự kết thúc của lệnh. Ký tự này là cần thiết vì bạn có thể thêm nhiều tùy chọn find sau lệnh đã cho.

Để có kết quả hữu ích hơn, chúng ta cần xem tên tệp nơi grep tìm thấy kết quả phù hợp. Trong một số trường hợp, bạn cũng có thể muốn biết số dòng nơi kết quả phù hợp. Cả hai yêu cầu này có thể được grep đáp ứng bằng cách sử dụng tùy chọn -H (hiển thị tên tệp) và -n (hiển thị số dòng):

```
$ find basicshell stubs -type f -exec grep -Hn setup {} \;
basicshell/basicshell.ino:3:void setup() {
basicshell/basicshell.ino:4: delay(2000); // Allow for serial setup
basicshell/basicshell.ino:5: printf("Hello from setup()\n");
```

stubs/stubs.ino:11:void setup() {

stubs/stubs.ino:12: delay(2000); // Allow for serial setup

stubs/stubs.ino:13: printf("Hello from setup()\n");

Bây giờ, kết quả cung cấp tất cả các chi tiết bạn cần.

Đôi khi, bạn chỉ muốn tìm vị trí của tệp header đã được cài đặt. Trong trường hợp này, bạn không muốn sử dụng grep trên nội dung tệp mà chỉ muốn xác định vị trí của tệp đó. Ví dụ, bạn muốn tìm tệp header của thư viện Arduino nRF24L01 đã được cài đặt ở đâu? Tệp header có tên RF24.h. Lumen có thể sử dụng lệnh find sau trên iMac của cô ấy:

$ find ~ -type f 2>/dev/null | grep 'RF24.h'

/Users/lumen/Documents/Arduino/libraries/RF24/RF24.h

/Users/lumen/Downloads/RF24-1.3.4/RF24.h

Dấu ~ đại diện cho thư mục home (trong hầu hết các shell). Bạn cũng có thể chỉ định $HOME hoặc thư mục một cách rõ ràng. Câu lệnh "2>/dev/null" chỉ được thêm vào để ngừng hiển thị các thông báo lỗi về các thư mục mà bạn không có quyền truy cập (điều này thường xảy ra trên Mac). Điều này đặc biệt hữu ích nếu bạn đang tìm kiếm qua toàn bộ hệ thống (bắt đầu từ thư mục gốc "/"). Hãy dành nhiều thời gian khi tìm kiếm từ thư mục gốc (chắc chắn là lúc để pha cà phê).

Kết quả của lệnh find trong ví dụ trước được chuyển qua lệnh grep để chỉ báo cáo những đường dẫn tệp có chứa chuỗi RF24.h. Việc tìm kiếm này cũng có thể được thực hiện bằng cách sử dụng tùy chọn -name của lệnh find:

$ find ~ -type f -name 'RF24.h' 2>/dev/null

/Users/lumen/Documents/Arduino/libraries/RF24/RF24.h

/Users/lumen/Downloads/RF24-1.3.4/RF24.h

Bằng cách này, ta có thể thấy rằng trên iMac của Lumen, thư mục ~/Documents/Arduino/libraries/RF24 chứa tệp header RF24.h (và các tệp liên quan).

Tùy chọn -name cũng chấp nhận tìm kiếm với biểu thức mẫu tệp. Để tìm tất cả các tệp header, bạn có thể thử:

$ find ~ -type f -name '*.h' 2>/dev/null

Điều này chỉ là bước đầu – nó cung cấp một công cụ mạnh mẽ không thể bỏ qua. Hãy tận dụng nó.

Tiếp Tục Cải Tiến

Một số lập trình viên chỉ phát triển phần mềm của họ cho đến khi "có vẻ như hoạt động". Khi họ thấy kết quả mà họ mong đợi, họ cho rằng công việc của mình đã xong. Ngay lập tức rửa tay gác lại, họ đưa nó vào sản xuất và sau đó lại phải quay lại để sửa lỗi, đôi khi là nhiều lần.

Đối với công việc sở thích, bạn có thể phản đối rằng điều này là chấp nhận được. Tuy nhiên, những người làm sở thích này cũng sẽ chia sẻ mã nguồn của mình. Bạn có muốn truyền bá mã nguồn tồi hoặc gây xấu hổ cho người khác? Đặt ra một ví dụ xấu? Khác với một mạch điện hàn, phần mềm có thể thay đổi vô hạn, vì vậy đừng ngại cải thiện nó. Phần mềm thường cần được đánh bóng.

Hãy tự hỏi bản thân:

- Liệu có cách tốt hơn để viết ứng dụng này không?
    - Thay thế các thủ tục macro bằng các hàm inline?
    - Sử dụng template trong C++ cho mã tổng quát?
    - Việc sử dụng Thư viện Mẫu Chuẩn (STL) có cải thiện ứng dụng không?
    - Có cách hiệu quả hơn để thực hiện một số phép toán không?
- Mã có dễ hiểu không?
- Dễ duy trì, mở rộng không?
- Ứng dụng có dễ bị tấn công hoặc bị lạm dụng không? Không có lỗi không?
- Mã có sử dụng tốt các quy tắc phạm vi của C/C++ để hạn chế quyền truy cập vào các handle và tài nguyên khác trong chương trình không?
- Có rò rỉ bộ nhớ không?
- Có điều kiện đua không?
- Có sự cố bộ nhớ dưới một số điều kiện không?

Hãy tự hào về công việc của bạn và làm cho nó trở nên tốt nhất có thể. Nếu làm đúng, nó có thể sẽ hữu ích cho bạn trong một lần ứng tuyển vào công việc. Các kỹ sư luôn tìm cách hoàn thiện nghề nghiệp của mình.

Làm Quen Với Các Bit

Tôi thường cảm thấy không thoải mái khi thấy những macro như BIT(x), được thiết kế để thiết lập một bit cụ thể trong một từ. Là một lập trình viên, tôi thích thấy biểu thức thực tế (1 << x) hơn là một macro BIT(x). Việc sử dụng một macro đòi hỏi sự tin tưởng rằng nó được triển khai như bạn nghĩ. Tôi ghét việc phải giả định. Vâng, tôi có thể tra cứu định nghĩa của macro, nhưng cuộc sống thì ngắn ngủi. Liệu việc thiết lập một bit có khó đến mức cần phải có sự gián tiếp này không? Tôi khuyến khích tất cả những người mới bắt đầu nên làm quen với việc thao tác với các bit trong C/C++. Lời cảnh báo duy nhất của tôi là phải chú ý đến thứ tự các phép toán. Nhưng điều này có thể dễ dàng khắc phục bằng cách thêm dấu ngoặc quanh biểu thức.

Điều này dẫn đến việc bạn cần dành thời gian để học thứ tự ưu tiên của các toán tử trong C và sự khác biệt giữa & và &&. Nếu bạn không chắc về những điều này, hãy đầu tư vào bản thân. Học chúng một cách vững vàng để có thể áp dụng suốt đời. Tôi bắt đầu sự nghiệp của mình với một bảng tóm tắt nhỏ dán trên màn hình. Nó cực kỳ hữu ích.

Hiệu Quả

Có vẻ như hầu hết các lập trình viên mới bắt đầu đều rất chú trọng đến hiệu quả. Đối với họ, việc mã hóa phiên bản hiệu quả nhất của một phép gán là một huy chương danh dự. Đừng hiểu lầm tôi – hiệu quả là quan trọng, như trong một MPU cần xử lý mã hóa và giải mã video, với tài nguyên ít ỏi. Tuy nhiên, nói chung, nhu cầu về hiệu quả không lớn như bạn nghĩ.

Một lập trình viên Linux mới từng than phiền với tôi về việc một truy vấn MySQL trong chương trình C++ mà anh ta đang làm việc rất không hiệu quả. Anh ta đã dành nhiều thời gian hơn để tìm cách giảm thiểu chi phí này so với thời gian mà anh ta có thể tiết kiệm được từ việc tối ưu hóa mã. Truy vấn đó chỉ chạy một lần,

hoặc có thể vài lần mỗi ngày. Trong bức tranh lớn hơn, hiệu quả của thành phần này là không quan trọng.

Khi bạn bắt tay vào một thay đổi vì lý do hiệu quả, hãy tự hỏi liệu nó có quan trọng trong bức tranh tổng thể không. Người dùng cuối có nhận thấy sự khác biệt không? Liệu nó có làm mã của ứng dụng trở nên khó hiểu và bảo trì hơn không? Liệu mã đó có an toàn hơn không? Đã có một thời điểm mà thời gian máy tính rất quý giá. Trong thế giới ngày nay, chi phí của lập trình viên mới là yếu tố quyết định. Nếu tất cả những gì bạn làm được chỉ là làm cho tác vụ nhàn rỗi FreeRTOS lâu hơn, vậy bạn đã đạt được gì?

Mã Nguồn Đẹp

Khi tôi còn trẻ và đầy nhiệt huyết, tôi nhận được bài tập đầu tiên có điểm thấp từ giáo sư trong học kỳ đó, mặc dù chương trình hoạt động hoàn hảo. Tôi khá là bị xúc phạm vì chương trình chạy tốt mà lại không được điểm tuyệt đối. Vậy vấn đề là gì? Vấn đề là mã không đủ đẹp.

Giờ tôi đã quên những chi tiết về vẻ đẹp đó, nhưng bài học vẫn còn đọng lại trong tôi. Bạn có thể nói rằng bài học này đã để lại dấu ấn tích cực trong tôi. Khi tôi phản đối, thầy đã trả lời cả lớp rằng mã chỉ được viết một lần nhưng phải đọc nhiều lần. Nếu mã xấu hoặc lộn xộn, nó có thể khó bảo trì và ít người muốn đảm nhận việc duy trì nó. Thầy khuyến khích chúng tôi làm mã dễ đọc và trở thành một tác phẩm đẹp. Điều này bao gồm việc định dạng mã đẹp mắt, chú thích rõ ràng nhưng không quá nhiều. Quá nhiều chú thích có thể làm mờ đi mã và dễ bị bỏ quên trong quá trình bảo trì chương trình.

Fritzing So Với Sơ Đồ Mạch

Tôi tin rằng việc làm việc từ các sơ đồ Fritzing là một thói quen không tốt. Một người muốn trở thành họa sĩ không tiếp tục vẽ trên các bức tranh theo kiểu tô màu theo số. Tuy nhiên, đó chính xác là những gì các sơ đồ Fritzing mang lại. Giống như một họa sĩ nghiệp dư chỉ biết tô màu theo số, nó có thể phù hợp với một số người chỉ muốn tái tạo lại bản dựng. Tuy nhiên, những người muốn theo đuổi sự nghiệp trong lĩnh vực này nên tránh xa.

Mặt khác, sơ đồ mạch là đại diện hình ảnh của một mạch điện. Nó cung cấp cái nhìn tổng quan mà sơ đồ dây dẫn không thể có. Bạn có thể hiểu được mạch bằng cách nhìn vào một đống dây không? Tôi khuyến khích những người đam mê dành thời gian học các ký hiệu và quy ước sơ đồ mạch. Hãy học cách nối dây cho các dự án của bạn từ sơ đồ mạch thay vì từ sơ đồ dây dẫn.

Trả Ngay Hay Trả Sau

Dưới đây là một vài lời khuyên chung dành cho các sinh viên dự định theo đuổi sự nghiệp lập trình, dù là trong lập trình nhúng hay các lĩnh vực khác. Trong sự phát triển nghề nghiệp lành mạnh, bạn sẽ bắt đầu với các công việc ở mức độ junior và tiến tới các công việc senior hơn khi tài năng của bạn phát triển. Hãy dành thời gian và kinh nghiệm để phát triển tài năng đó. Đừng quá tham vọng và vội vã.

Trước đây, vào những năm 1970, có một quảng cáo của Midas Muffler ở Bắc Mỹ với thông điệp kiểu như "bạn có thể trả ngay hoặc trả sau." Thông điệp là về việc bảo dưỡng sớm. Sự nghiệp của bạn cũng cần bảo dưỡng sớm. Nếu bạn làm việc ngay từ bây giờ, bạn sẽ nhận lại những lợi ích cho sự nghiệp của mình sau này. Đừng ngại dành thời gian và sự hy sinh trong những năm đầu tiên.

Lập Trình Viên Không Thể Thiếu

Lời khuyên cuối cùng của tôi liên quan đến thái độ của nhân viên. Sau những năm đầu tiên trong sự nghiệp, một số lập trình viên chuyển sang chế độ "bảo mật công việc." Họ sẽ xây dựng các hệ thống khó theo dõi và giữ lại thông tin cho riêng mình. Họ không thích chia sẻ với các đồng nghiệp khác. Động lực cho điều này là muốn trở thành người không thể thiếu đối với công ty.

Bạn không muốn trở thành một nhân viên không thể thiếu. Các đồng nghiệp sẽ không thích bạn và ban quản lý sẽ không chịu đựng mãi được điều đó. Họ sẽ tìm cách phá vỡ sự phụ thuộc đó nếu cần thiết. Các công ty không thích bị bắt làm con tin.

Có một lý do khác để tránh việc trở thành người không thể thiếu – bạn sẽ muốn chuyển sang những thử thách mới và bỏ lại các công việc cũ (cho một nhân viên junior). Ban quản lý sẽ không giao cho bạn những thử thách mới và thú vị nếu bạn cần hỗ trợ những công việc cũ đó. Nếu những công việc cũ quá khó để giao cho một

nhân viên junior, thì chính nhân viên đó có thể sẽ nhận được cơ hội mới thay vì bạn. Trong công việc, bạn muốn sẵn sàng đón nhận những thử thách mới.

Lời Cuối

Chúng ta đã đến màn hạ cuối – kết thúc của cuốn sách này. Nhưng đây không phải là kết thúc với bạn, vì bạn sẽ lấy những gì đã thực hành và áp dụng các khái niệm FreeRTOS vào các ứng dụng của riêng mình. Tôi hy vọng bạn đã tận hưởng hành trình này. Cảm ơn bạn đã cho phép tôi được làm người hướng dẫn cho bạn.