FreeRTOS for ESP32 – Arduino

Chương 15 : Tips and Hints

Ai cũng có điểm khởi đầu, kể cả các chuyên gia. Chương này chứa một số mẹo và gợi ý cho những người mới bắt đầu lập trình Arduino. Arduino được phát triển cho những sinh viên đang bước vào lĩnh vực lập trình nhúng, cho phép họ làm quen mà không bị choáng ngợp bởi các chi tiết kỹ thuật. Bởi vì một số người đam mê và nghiệp dư với Arduino có thể thiếu đào tạo phần mềm ,nên có một vài điều đáng nói mà hầu hết những người thực hành dày dạn kinh nghiệm đều coi là điều hiển nhiên

Diễn đàn: Đầu tư sự nỗ lực

Các diễn đàn thường có các bài đăng yêu cầu giúp đỡ để phát triển một thứ gì đó phức tạp. Yêu cầu thường là "Tôi muốn làm... Tôi là người mới - ai đó có thể giúp tôi không?" Không phải là lỗi của họ khi không biết về sự phức tạp và cũng không sao cả khi họ là người mới. Nhưng phải mất bao lâu để nhập yêu cầu này? Mười giây? Những người trả lời sẽ nỗ lực bao nhiêu? Có thể là mười giây nếu bạn nhận được phản hồi

Mọi người sẵn sàng giúp đỡ hơn nhiều nếu họ thấy rằng bạn đã tự mình đầu tư một số nỗ lực. Bạn đã vạch ra những gì bạn nghĩ là cần thiết và cách bạn nghĩ rằng bạn sẽ đạt được điều đó chưa? Không sao cả khi sai về điều đó bởi vì nó chứng tỏ rằng bạn không chỉ ném vấn đề qua tường và hy vọng người khác sẽ làm tất cả công việc này

Hãy bắt đầu từ việc nhỏ

Đôi khi các yêu cầu được đưa ra để giúp lập trình các bài tập lớn và phức tạp. Phản hồi thông thường cho những bài đăng này là không có câu trr lời nào cả. Không phải là không ai biết câu trả lời cho câu hỏi mà là không ai muốn bỏ công sức để hướng dẫn người dùng về nhiều điều mà họ cần phải học trước. Nếu họ phải tiếp tục với bài tập đó, hãy chia nó thành các phần nhỏ hơn. Sau đó, hãy hỏi những câu hỏi cụ thể về những khó khăn đang gặp phải. Tuy nhiên, cách tiếp cận tốt nhất là bắt đầu từ những bài tập đơn giản hơn. Mọi người đều biết điều này nhưng những người đam mê lại thiếu kiên nhẫn. Bạn là kiểu người chỉ muốn "câu trả lời" hay bạn muốn biết cách xác định câu trả lời?

Kinh nghiệm là người thầy tốt nhất.

Có một lý do mà mọi người bắt đầu với đèn LED nhấp nháy. LED rất đơn giản - chúng bật hoặc tắt. Đơn giản như vậy, nhưng vẫn có những bài học cần học. Ví dụ: nếu đèn LED không sáng, nguyên nhân là gì? Nếu bạn chưa bao giờ gặp phải điều đó trước đây, bạn có thể không nhận ra rằng đèn LED đã được nối dây sai cực.

Những lập trình viên mới, được trang bị kiến thức về ngôn ngữ lập trình của họ, thường nhảy vào việc viết toàn bộ chương trình một lần và sau đó cố gắng gỡ lỗi nó. Khi đã được trang bị các yêu cầu, phần mềm được viết lên và sau đó được kiểm tra ở cuối. Các phiên gỡ lỗi tiếp theo có thể diễn ra dồn dập. Đừng hiểu sai - các yêu cầu là quan trọng. Khi các yêu cầu đến từ chính chúng ta (trong sở thích), chúng thường thay đổi thường xuyên. Hoặc nếu bạn đang phát triển một cái gì đó chỉ để vui, bạn có thể thậm chí không có yêu cầu chắc chắn. Đây chỉ là một số lý do để tránh việc lập trình mọi thứ trước khi kiểm tra.

Việc gỡ lỗi khó khăn hơn trên các thiết bị nhúng. Có thể không có trình gỡ lỗi có sẵn hoặc khả năng theo dõi của nó bị giới hạn. Bạn không thể bước qua một quy trình phục vụ ngắt, chẳng hạn. Một cách tiếp cận tốt hơn là bắt đầu từ những điều nhỏ và sử dụng các phương pháp shell và stub cơ bản.

Shell cơ bản

Thay vì cố gắng viết toàn bộ ứng dụng trước khi gỡ lỗi nó, hãy viết một shell cơ bản trước. Trong shell của một chương trình này, hãy mã hóa một hàm setup() và loop() tối thiểu cho mã Arduino của bạn. Đặc biệt khi sử dụng bảng phát triển ESP32, hãy tận dụng liên kết USB với serial bằng cách sử dụng Serial Monitor. Điều này sẽ đơn giản hóa chu kỳ phát triển và kiểm tra của bạn.

Shell đầu tiên có thể chỉ là một "Hello from setup()" và một "Hello from loop()" trong hàm loop(), như được minh họa dưới List 15.1 dưới đây. Hãy nhớ rằng chương trình đầu tiên này không cần phải hoàn hảo. Bằng cách thử điều này ngay từ đầu, bạn chứng tỏ được một chu kỳ biên dịch đầy đủ, tải lên flash và kiểm tra chạy. Lệnh delay() ở dòng 4 cho phép các thư viện ESP32 thiết lập liên kết USB serial trước khi tiếp tục.

```
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: }
```

List 15.1 : Một shell khởi động cơ bản của một chương trình

Cách tiếp cận với Stub

Rõ ràng, bạn muốn ứng dụng của mình làm nhiều hơn là shell cơ bản đó. Hãy bắt đầu xây dựng trên khung đó bằng cách thêm các hàm stub (List 15.2). Các hàm init_oled() và init_gpio() chỉ là các stub cho những gì sẽ trở thành các hàm khởi tạo cho các thiết bị OLED và GPIO

Biên dịch, tải lên và kiểm tra những gì bạn có. Nó có chạy không? Đầu ra từ List 15.2 nên giống như sau trong 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 - việc khởi tạo thiết bị thực tế. Tránh các sai lầm khi thực hiện và giữ cho các bổ sung mã nhỏ và từng bước. Điều này sẽ giúp tiết kiệm rất nhiều thời gian suy nghĩ khi các vấn đề mới phát sinh. Thật đáng ngạc nhiên là ngay cả những bổ sung nhỏ rõ ràng 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: }
```

List 15.2 : Chương trinh shell cơ bản được mở rộng với các stub

Sơ đồ khối

Các ứng dụng lớn hơn có thể có lợi khi có một sơ đồ khối để lập kế hoạch cho các tác vụ FreeRTOS cần thiết. Các hàm setup() và loop() bắt đầu từ "loopTask", được cung cấp theo mặc định. Nếu bạn không thích việc cấp phát stack cho 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? Có phải các routine ISR sẽ cung cấp sự kiện cho bất kỳ tác vụ nào trong số đó không? Vẽ các đường có thể là các hàng đợi chưa message. Có thể sử dụng các đường chấm để chỉ ra nơi có các sự kiện, semaphore hoặc mutex liên quan giữa các tác vụ. Nó không nhất thiết phải là một sơ đồ được phê duyệt bởi UML - chỉ cần sử dụng các quy ước mà bạn thấy hợp lý.

Là một phần của việc tạo mẫu cho ứng dụng của bạn, hãy tạo mỗi tác vụ ban đầu dưới dạng một hàm mẫu. Tất cả những gì hàm đó cần làm là thông báo về việc bắt đầu của nó. Một tác vụ không dược phép trả về trong FreeRTOS, vì vậy để phục vụ cho mục đích mẫu, tác vụ có thể xóa chính nó sau khi thông báo. Sau này, bạn có thể hoàn thiện tác vụ với mã cuối cùng.

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 rất khó chịu 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 mã vừa thêm vào. Lỗi có thể liên quan đến mã vừa thêm. 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 mà tệp tiêu đề cho hỗ trợ printf() của newlib được định nghĩa. Một ví dụ về mã có thể dẫn đến lỗi là

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

Bạn có thấy vấn đề không? Phải có một đối số chuỗi C sau chuỗi định dạng, để thỏa mãn mục định dạng "%s". Hàm printf() sẽ yêu cầu điều đó và sẽ truy cập vào stack để lấy nó. Nhưng giá trị mà nó tìm thấy có thể là rỗng hoặc nullptr và gây ra lỗi. Trình biên dịch biết về những vấn đề này nhưng những cảnh báo nãy không được đưa ra vì một lý do nào đó.

Một nguồn lỗi phổ biến khác là hết không gian stack. Nếu bạn không thể ngay lập tức xác định nguyên nhân của lỗi, hãy cấp phát thêm không gian stack cho tất cả các tác vụ đã thêm. Điều này có thể loại bỏ lỗi. Khi bạn hoàn thành việc kiểm tra, các phân bổ stack khác nhau có thể được giảm

Cũng có vấn đề về vòng đời của đối tượng mà bạn cần lưu ý. Bạn đã gửi một con trỏ qua hàng đợi chưa? Hãy xem phần Hiểu Thêm Về Thời Gian Lưu Trữ Của Bạn. Hay là đối tượng C++ đã bị hủy trước khi một tác vụ khác cố gắng truy cập vào nó?

Hiểu Thêm Về Thời Gian Lưu Trữ Của Bạn

Khi bạn mới bắt đầu, có vẻ như có rất nhiều điều để học, đừng để điều đó làm bạn nản lòng . Hãy xem xét đoạn mã sau:

```
static char area1[25];

void function foo() {
  mảng ký tự area2[25];
```

Khu vực lưu trữ cho mảng area1 được tạo ra ở đâu? Nó có giống với area2 không? Mảng area1 được tạo ra trong một vùng SRAM được phân bổ vĩnh viễn cho mảng đó. Khu vực lưu trữ đó không bao giờ biến mất.

Bộ nhớ cho area2 thì khác vì nó được cấp phát trên stack. Ngay khi hàm foo() trả về, bộ nhớ đó sẽ được giải phóng. Nếu bạn truyền con trỏ đến area2 qua một hàng đợi tin nhắn, ví dụ, con trỏ đó sẽ không còn hợp lệ ngay khi foo() trả về.

Nếu bạn muốn mảng array2 tồn tại sau khi foo() trả về, bạn có thể khai báo nó là tĩnh trong hàm.

```
static char area1[25];

void function foo() {
  static char area2[25];
```

Bằng cách thêm từ khóa static vào khai báo của area2, chúng ta đã di chuyển việc cấp phát bộ nhớ của nó đến cùng một vùng với area1 (tức là không nằm trên stack).

Lưu ý rằng mảng array1 cũng được khai báo với thuộc tính tĩnh, nhưng trong trường hợp đó, từ khóa tĩnh có ý nghĩa khác (khi được khai báo bên ngoài hàm). Bên ngoài một hàm, từ khóa tĩnh chỉ có nghĩa là không gán một ký hiệu bên ngoài cho nó ("area1"). Việc khai báo các biến này với tĩnh giúp tránh xung đột trong bước liên kết

Tránh External Names

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

Các hàm setup() và loop() ngược lại, phải là các ký hiệu extern vì trình liên kết phải gọi chúng từ một mô-đun khởi động ứng dụng. Việc là extern cho phép trình liên kết xác định và liên kết với chúng.

Tận Dụng Phạm Vi

Một thực tiễn tốt nhất trong phần mềm cần được áp dụng 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 mọi thứ toàn cục là tiện lợi cho các dự án nhỏ nhưng có thể trở thành cơn đau đầu cho các ứng dụng lớn hơn. Tôi thích gọi đây là phong cách lập trình cowboy. Các lập trình viên nhớ đến COBOL sẽ có liên quan.

Vấn đề với phong cách cowboy là nếu bạn tìm thấy một lỗi nơi mà một cái gì đó đang được sử dụng/ sửa đổi khi không nên, sẽ rất khó để cô lập. Khi các quy tắc phạm vi của ngôn ngữ được sử dụng thay vào đó, trình biên dịch sẽ thông báo cho bạn ngay lập tức khi bạn cố gắng truy cập vào một cái gì đó không nên. Quyền truy cập được phép sẽ được thực thi.

Một cách để giới hạn phạm vi của các handle 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 đến một hàng đợi và một mutex, thì hãy truyền những mục này trong một cấu trúc đến tác vụ tại thời điểm tạo tác vụ. Sau đó, chỉ có tác vụ đang sử dụng mới nắm được về những handle này

Thư Giãn Đầu Óc

Khi nói đến trải nghiệm con người, các nhà tâm lý học cho chúng ta biết rằng có ít nhất 16 loại tính cách khác nhau (Myers-Briggs). Nhưng tôi tin rằng hầu hết mọi người sẽ tiếp tục làm việc trên một vấn đề sau khi họ đã từ bỏ việc suy nghĩ một cách có ý thức về nó. Vì vậy, khi bạn thấy mình mất kiên nhẫn trong một phiên gỡ lỗi, hãy cho phép bản thân nghỉ ngơi. Điều này cũng có thể giúp bạn khỏi việc thực hiện những rủi ro không cần thiết, có thể kết thúc bằng magic smoke.

Một số người có thể ngủ ngon, trong khi những người khác sẽ trằn trọc suốt đêm. Nhưng tâm trí sẽ nghiền ngẫm những sự kiện trong ngày và qua tất cả các kịch bản có thể. Vào buổi sáng, vợ/chồng của bạn có thể phàn nàn về những lời lẩm bẩm của bạn trong giấc ngủ. Nhưng khi bạn tỉnh dậy, bạn thường sẽ có một số 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.

Sử dụng Notebooks

Khi đầu bạn chạm vào gối vào ban đêm, bạn có thể đột nhiên nhớ ra một hoặc hai điều mà bạn đã quên chú ý trong mã hoặc mạch. Một cuốn sổ tay bên cạnh giường có thể là một công cụ ghi nhớ hữu ích. Khi bạn còn trẻ, tâm trí không bị rối loạn, và việc nhớ mọi thứ là dễ dàng. Tuy nhiên, khi bạn già đi, cuộc sống trở nên đầy rẫy những phức tạp và sau đó mọi thứ sẽ bắt đầu trải qua các kẽ hở. Một cuốn sổ tay rất hữu ích để ghi lại những gì đã thử hoặc cách bạn đã giải quyết một vấn đề. Những buổi sáng thứ hai tại nơi

làm việc sẽ được hưởng lợi từ việc có thể tiếp tục từ nơi bạn đã dừng lại vào thứ Sáu trước đó.

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

Hỏi Để Được Giúp Đỡ

Kể từ khi sự xuất hiện của internet, chúng ta có được sự may mắn của các diễn đàn và công cụ tìm kiếm, cho phép chúng ta "goggle" để tìm kiếm sự giúp đỡ. Một tìm kiếm trên web thường là bước đầu tiên hiệu quả để có được câu trả lời hoặc manh mối ngay lập tức. Nhưng 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 phát hiện rằng người khác đã gặp phải những vấn đề tương tự. Trong trường hợp đó, bạn có thể nhận được một hoặc nhiều câu trả lời được đăng để nghiên cứu và xem xét.

Khi liên hệ với một diễn đàn, hãy đặt câu hỏi thông minh. Các bài đăng trên diễn đàn như "I2C của tôi không hoạt động, bạn có thể giúp không?" thể hiện rất ít sáng kiến. Đây là một nỗ lực khác kiểu "ném vấn đề qua tường và hy vọng điều tốt nhất". Bạn có đưa xe của mình đến một gara và chỉ nói rằng xe của bạn bị hỏng không? Các bài đăng trên diễn đàn không nên cần phải chơi trò chơi hai mươi 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 hiện tại ?
- Có thể đưa ra các chi tiết về bo mạch ESP32 của bạn
- Nền tảng phát triển là gì?
- Bạn đang sử dụng thư viện nào, nếu có?
- Bất kỳ điều kỳ lạ nào bạn có thể quan sát được?

Tôi sẽ tránh việc đăng code 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 này làm cho nó tự giải thích. Tôi tin

rằng sẽ hiệu quả hơn khi giải thích bản chất của vấn đề trước. Bạn có thể luôn đăng code như một phần theo sau.

Khi đăng code, có thể không cần thiết phải đăng tất cả (đặc biệt là khi code dài). Đôi khi chỉ cần đăng những gì 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.

Các diễn đàn thường có cách để đăng "code" trong tin nhắn (như [code] ... [/code]). Hãy chắc chắn sử dụng điều đó bất cứ khi nào có thể. Nếu không, giữa phông chữ tỷ lệ và việc thiếu tôn trọng đối với việc thụt lề, code trở thành một mớ hỗn độn khủng khiếp để đọc. Tôi ghét việc đọc code được thụt lề một cách tồi tệ.

Có một tác dụng phụ có lợi khi mô tả chính xác vấn đề, dù là trong một bài đăng hay qua email - khi bạn hoàn thành việc mô tả vấn đề, bạn có thể đã nhận ra câu trả lời. Ngoài ra, khi làm việc với một đồng nghiệp hoặc bạn học, chỉ cần giải thích vấn đề cho họ cũng có thể mang lại kết quả tương tự.

Chia Nhỏ và Hoàn Thiện

Sinh viên mới có thể gặp khó khăn với một ứng dụng bị treo. Làm thế nào để bạn cô lập phần code gây lỗi? Lập trình viên dày dạn kinh nghiệm biết kỹ thuật chia nhỏ và hoàn thiện.

Khái niệm này đơn giản như trò chơi đoán số. Nếu bạn phải đoán một số mà tôi đang nghĩ đến giữa 1 và 10, và bạn đoán 6 và tôi trả lời rằng số đó thấp hơn, thì bạn sẽ chia nó ra một lần nữa với một lần đoán có thể là 3. Cuối cùng, bạn sẽ có thể đoán số bằng cách giảm dần các khoảng với mỗi lần thử. Khi một chương trình bị treo, bạn chia nó thành hai nửa cho đến khi cô lập được vùng mã gây ra vấn đề.

Các phương pháp khác nhau có thể được sử dụng - một lệnh in ra Serial Monitor hoặc việc kích hoạt một LED. LED rất hữu ích cho việc theo dõi ISR nơi bạn không thể in thông điệp. Nếu bạn có đủ GPIO, bạn thậm chí có thể sử dụng một LED hai màu để báo hiệu những điều khác nhau. Ý tưởng là để chỉ ra rằng mã đã được thực thi tại các điểm quan tâm. Nếu bạn cần nhiều hơn từ LED của mình, 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 mã một cách cẩn thận hơn để tìm nguyên nhân.

Lập trình để có câu trả lời

Tôi đã thấy các lập trình viên trong công việc tranh cãi trong nửa giờ về những gì xảy ra khi điều này hay điều kia xảy ra. Ngay cả sau đó, cuộc tranh cãi thường vẫn không được giải quyết. Toàn bộ vấn đề thường có thể được giải quyết bằng cách viết một chương trình đơn giản trong một phút để kiểm tra giả thuyết. Tất nhiên, hãy sử dụng một chút lý trí với những gì bạn đã quan sát :

- Hành vi quan sát có được hỗ trợ API không?
- Hay hành vi này là do việc 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 và các chú thích sẽ chỉ ra ý định của các giao diện được tài liệu kém. Kết luận: đừng ngại viết mã tạm thời.

Tận dụng lệnh Command

Khi kiểm tra mã nguồn mở, bạn có thể tìm kiếm nó trực tuyến hoặc xem những gì bạn đã cài đặt trên hệ thống của mình. Việc xem mã đã cài đặt là quan trọng khi bạn nghĩ rằng bạn 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 ẩn sau và vẫn ẩn đối với sinh viên. Nếu bạn đang sử dụng hệ thống POSIX (Linux, FreeBSD hoặc MacOS, v.v.), thì lệnh tìm là cực kỳ hữu ích. Người dùng Windows có thể cài đặt WSL (Windows Subsystem for Linux) để thực hiện điều tương tự hoặc sử dụng phiên bản lệnh cho Windows.

Hãy dành thời gian để làm quen với lệnh Command. Nó mạnh mẽ và trông có vẻ đáng sợ đối với người mới, nhưng không có bí ẩn lớn nào ở đó. Chỉ có rất nhiều tính linh hoạt có thể được tích hợp theo từng đợt nhỏ. Lệnh command hỗ trợ một loạt các tùy chọn khiến nó có vẻ phức tạp. Hãy xem xét những điều quan trọng và hữu ích nhất trong số này. Định dạng lệnh cơ bản theo dạng tổng quát sau:

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

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

```
$ find basicshell stubs -print
```

Hoặc chỉ cần :

```
$ find basicshell stubs
basicshell
basicshell/basicshell.ino
stubs
stubs/stubs.ino
```

Với dạng lệnh trên, tất cả các tên đường dẫn từ các thư mục đã cho sẽ được liệt kê. Điều này bao gồm cả thư mục và tệp. Hãy hạn chế đầu ra chỉ cho các tệp với tùy chọn loại với đối số "f" (chỉ định tệp):

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

Bây giờ đầu ra chỉ hiển thị các tên đường dẫn tệp. Đầu ra này vẫn không hữu ích lắm. Điều chúng ta cần làm là yêu cầu lệnh tìm thực hiện một cái gì đó với những tên tệp này. Lệnh grep là một lựa chọn tuyệt vời:

```
$ find basicshell stubs -type f -exec grep 'setup' {} \;
void setup() {
  delay(2000); // Cho phép thiết lập serial
  printf("Hello from setup()\n");
void setup() {
  delay(2000); // Cho phép thiết lập serial
  printf("Hello from setup()\n");
```

Chúng ta đã gần đến nơi, nhưng trước tiên, hãy giải thích một vài điều. Chúng ta đã thêm tùy chọn tìm exec theo sau là tên của lệnh (grep) và một số cú pháp đặc biệt. Đối số 'setup' là biểu thức chính quy grep mà chúng ta đang tìm kiếm (hoặc một chuỗi đơn giản). Nó thường cần được đặt trong dấu nháy đơn để ngăn shell can thiệp vào nó. Đối số "{}" chỉ ra nơi trên dòng lệnh để truyền tên đường dẫn (cho grep). Cuối cùng, token "\;" đánh dấu sự kết thúc của lệnh. Điều này là cần thiết vì bạn có thể thêm nhiều tùy chọn tìm khác sau lệnh đã cung cấp.

Để trở nên hữu ích hơn, chúng ta cần thấy tên tệp nơi grep tìm thấy một kết quả. Trong một số trường hợp, bạn cũng có thể muốn biết số dòng nơi kết quả được tìm thấy. Cả hai điều này đều được grep đáp ứng bằng cách sử dụng tùy chọn H (hiển thị đường dẫn) 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); // Cho phép thiết lập
serial basicshell/basicshell.ino:5: printf("Hello from setup()\n");
stubs/stubs.ino:11:void setup() {
stubs/stubs.ino:12: delay(2000); // Cho phép thiết lập serial
stubs/stubs.ino:13: printf("Hello from setup()\n");
```

Điều này giờ đây cung cấp tất cả các chi tiết mà bạn có thể muốn.

Đôi khi, chúng ta chỉ muốn tìm hiểu nơi tệp tiêu đề được cài đặt. Trong trường hợp này, chúng ta không muốn grep tệp, mà chỉ muốn xác định nơi tệp đó. Ví dụ, tệp tiêu đề cho thư viện Arduino nRF24L01 được cài đặt ở đâu? Tệp tiêu đề có tên là RF24.h. Lumen có thể sử dụng lệnh find sau trên iMac:

```
$ 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 ngã (~) đại diện cho thư mục chính (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 thông báo lỗi về các thư mục mà không có quyền truy cập (điều này xảy ra rất nhiều trên Mac). Điều này đặc biệt hữu ích nếu bạn kết thúc việc tìm kiếm qua mọi thứ (bắt đầu từ thư mục gốc "/"). Hãy dành nhiều thời gian khi tìm kiếm từ gốc (chắc chắn là một khoảnh khắc để pha cà phê). Đầu ra của lệnh find trong ví dụ trước được chuyển vào grep để nó chỉ báo cáo những đường dẫn có chuỗi RF24.h trong đó. Tìm kiếm này cũng có thể được thực hiện bằng cách sử dụng tùy chọn tên 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
```

Dù bằng cách nào, rõ ràng rằng trên iMac của Lumen, thư mục ~/Documents/Arduino/libraries/RF24 chứa tệp tiêu đề RF24.h (và các tệp liên quan).

Tùy chọn tên cũng chấp nhận các tìm kiếm globbing tệp. Để tìm tất cả các tệp tiêu đề, bạn có thể thử:

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

Hãy tận dụng nó.

Có thể Biến Đổi Vô Hạn

Một số lập trình viên chỉ phát triển phần mềm của họ cho đến khi nó "có vẻ hoạt động". Khi họ thấy kết quả mà họ mong đợi, họ cho rằng công việc của họ đã hoàn thành. Ngay lập tức họ rời tay khỏi nó, họ đưa nó vào sản xuất chỉ để thấy nó quay lại để sửa lỗi. Đôi khi là nhiều lần như vậy.

Đối với công việc sở thích, bạn có thể phản đối rằng điều này là chấp nhận được. Tuy nhiên, những người sở thích đó sẽ chia sẻ mã của họ. Bạn có muốn gây ra mã xấu hoặc đáng xấu hổ cho người khác không? Khác với một mạch điện tử hàn, phần mềm có thể biến đổi vô hạn, vì vậy đừng ngại cải thiện nó. Phần mềm thường cần được chau chuốt.

Hãy tự hỏi bản thân:

- Có cách nào tốt hơn để ứng dụng này có thể được viết không?
  - Thay thế các thủ tục macro bằng các hàm inline?
  - Sử dụng các mẫu C++ cho mã tổng quát?
  - Liệu việc sử dụng Thư viện mẫu chuẩn (STL) có cải thiện ứng dụng không?
  - Có cách nào hiệu quả hơn để thực hiện một số thao tác không?
- Mã có dễ hiểu không?
- Dễ bảo trì, hoặc mở rộng không?
- Ứng dụng có dễ bị khai thác hoặc lạm dụng không?Không có lỗi?
- Mã có sử dụng tốt các quy tắc phạm vi C/C++ để hạn chế quyền truy cập vào các tài nguyên khác trong chương trình không?
- Có rò rỉ bộ nhớ không?
- Có sự hỏng hóc bộ nhớ trong một số điều kiện không?

Hãy tự hào về công việc của bạn và làm cho nó tốt nhất có thể. Nếu làm đúng, nó có thể phục vụ bạn một ngày nào đó trong một đơn xin việc. Các kỹ sư luôn tìm cách hoàn thiện tay nghề của họ.

Làm Quen với Các Bit

Tôi thường nhăn mặt khi thấy các 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 yêu cầu sự tin tưởng rằng nó được triển khai theo cách mà bạn giả định. Tôi ghét việc phải giả định. Vâng, tôi có thể tra cứu định nghĩa macro, nhưng cuộc sống thì ngắn ngủi. Việc thiết lập một bit có khó đến mức cần thiết phải có sự gián tiếp này không? Tôi khuyến khích tất cả những người mới bắt đầu làm chủ việc thao tác bit trong C/C++. Cảnh báo duy nhất của tôi là hãy chú ý đến thứ tự thực hiện. Nhưng điều này dễ dàng được khắc phục với một số dấu ngoặc quanh biểu thức.

Điều này dẫn đến việc dành thời gian để học thứ tự ưu tiên của các toán tử C và sự khác biệt giữa & và &&. Nếu bạn không chắc chắn về những điều này thì hãy đầu tư vào bản thân. Hãy học chúng một cách chắc chắn để bạn có thể áp dụng chúng trong suốt phần đời còn lại của mình. Tôi đã bắt đầu sự nghiệp của mình với một bảng nhỏ dán trên màn hình. Nó rất hữu ích.

Sự hiệu quả

Dường như hầu hết các lập trình viên mới bắt đầu đều bị ám ảnh bởi sự hiệu quả khi làm việc. Đối với họ, đó là một biểu tượng danh dự để lập trình phiên bản hiệu quả nhất của một bài tập. Đừng hiểu sai – có một chỗ cho hiệu quả, như trong một MPU nơi nó phải xử lý mã hóa và giải mã video, với barely đủ tài nguyên để thực hiện điều đó. Tuy nhiên, nhìn chung, nhu cầu không lớn như bạn nghĩ.

Một lập trình viên Linux junior từng phàn nàn với tôi về việc một truy vấn MySQL không hiệu quả mà anh ta đang làm việc trong một chương trình C++. Anh ta đã dành nhiều thời gian hơn để tìm cách giảm thiểu chi phí này hơn là thời gian mà anh ta sẽ tiết kiệm được từ việc tối ưu hóa mã. Truy vấn chỉ chạy một lần, hoặc có thể vài lần mỗi ngày mà thôi. Trong bức tranh lớn, hiệu quả của thành phần này là không quan trọng.

Khi bạn bắt đầu một thay đổi vì lý do hiệu quả, hãy tự hỏi bản thân liệu điều đó có quan trọng trong bức tranh lớn không. Người dùng cuối có nhận thấy sự khác biệt không? Liệu điều đó có làm cho mã ứng dụng khó hiểu và duy trì hơn không? Liệu mã có an toàn hơn không? Đã có một thời điểm khi thời gian máy tính là quý giá. Nếu tất cả những gì bạn đạt

được chỉ là nhiều thời gian cho tác vụ nhàn rỗi FreeRTOS, thì bạn đã đạt được điều gì?

Sự chỉn chu bên ngoài của mã nguồn

Khi tôi còn trẻ và đầy nhiệt huyết, tôi đã nhận được từ giáo sư của mình bài tập đầu tiên được chấm điểm cho học kỳ đó, với điểm không trọn vẹn. Tôi khá bị xúc phạm vì chương trình hoạt động hoàn hảo. Vậy vấn đề là gì? Vấn đề là nó không đủ đẹp.

Tôi quên các chi tiết về vẻ đẹp bây giờ, nhưng bài học đã ở lại với tôi. Bạn có thể nói rằng bài học đã để lại dấu ấn tốt cho tôi. Khi tôi ban đầu phản đối, ông đã trả lời lớp rằng mã chỉ được viết một lần nhưng được đọc nhiều lần. Nếu mã xấu hoặc lộn xộn, nó có thể khó duy trì và ít người muốn thực hiện nhiệm vụ duy trì nó. Ông khuyến khích chúng tôi làm cho mã dễ đọc và trở thành một tác phẩm đẹp. Điều này bao gồm mã được định dạng đẹp, các chú thích được định dạng đẹp, nhưng không quá nhiều chú thích. Quá nhiều chú thích có thể làm mờ mã và bị bỏ qua trong việc bảo trì chương trình.

Fritzing vs Schematics

Tôi tin rằng làm việc từ các sơ đồ Fritzing là một thực hành không tốt. Một người muốn trở thành họa sĩ không tiếp tục với các bức theo số. Tuy nhiên, đây chính xác là những gì sơ đồ Fritzing cho thấy. Giống như một họa sĩ nghiệp dư vẽ theo số, nó có thể phù hợp với một số người chỉ muốn tái tạo bản dựng. Tuy nhiên, nó nên được tránh bởi những người đang hướng tới một sự nghiệp trong lĩnh vực này.

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

Ngay bây giờ hay Sau này

Đây là một số lời khuyên chung cho sinh viên dự kiến theo đuổi sự nghiệp lập trình, cho dù đó là cho tính toán nhúng hay không. Trong sự phát triển nghề nghiệp lành mạnh, bạn sẽ bắt đầu với các nhiệm vụ junior và tiến tới các nhiệm vụ cao hơn khi tài năng của bạn phát triển. Hãy cho phép thời gian và kinh nghiệm phát triển tài năng đó. Đừng quá tham vọng và vội vàng.

Đã từng có một quảng cáo Midas Muffler cũ ở Bắc Mỹ vào những năm 1970 với nội dung "bạn có thể trả cho tôi ngay bây giờ hoặc trả sau". Thông điệp là về việc duy trì và hài lòng quá sớm. Nhưng nếu bạn làm việc chăm chỉ ngay bây giờ, nó sẽ mang lại lợi ích cho sự nghiệp của bạn sau này. Đừng ngại dành thời gian và hy sinh trong những năm đầu.

Trở thành lập trình viên không thể thiếu của 1 tổ chức?

Lời khuyên cuối cùng của tôi liên quan đến thái độ của nhân viên. Sau khi những năm đầu sự nghiệp trôi qua, một vài lập trình viên biến thành chế độ "đảm bảo công việc". Họ sẽ xây dựng các hệ thống khó theo dõi và giữ thông tin cho riêng mình. Họ không thích chia sẻ với các nhân viên khác. Động lực cho điều này là để trở nên không thể thiếu cho 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 điều đó mãi mãi. Họ sẽ chịu thiệt nếu cần thiết để phá vỡ sự phụ thuộc đó.

Có một lý do khác để tránh điều này– bạn sẽ muốn chuyển sang những thách thức mới và để lại các chức năng công việc cũ của mình (cho một người junior). Ban quản lý sẽ không giao cho bạn những thách thức mới và thú vị nếu bạn cần hỗ trợ cho những chức năng cũ đó. Nếu những thứ cũ đó quá khó để chuyển giao cho một người junior, thì người junior đó có thể nhận được cơ hội mới đó thay vì bạn. Trong môi trường làm việc, bạn muốn sẵn sàng để đối mặt với những thách thức mới.

Lời kết cuối

Chúng ta đã đến phần cuối – kết thúc của cuốn sách này. Nhưng đây không phải là kết thúc cho bạn vì bạn sẽ áp dụng những gì bạn đã thực hành và áp dụng các khái niệm FreeRTOS trong các ứng dụng của riêng bạn. Tôi hy vọng bạn đã thích hành trình này. Cảm ơn bạn đã cho phép tôi hướng dẫn bạn.