Máy trạng thái trong chương trình nhúng hướng sự kiện

Draft

1. Giới thiệu

Hệ thống nhúng trên vi điều khiển hoạt động dựa trên các sự kiện sinh ra từ các ngắt và các sự kiện phát sinh nội tại bên trong chương trình nhúng chạy trên nó. Nhiệm vụ của chương trình nhúng là nhận biết các sự kiện và ra lệnh phản ứng lại với các sự kiện tương ứng nhằm chuyển hệ thống từ trạng thái hiện tại sang một trạng thái mới. Một chương trình nhúng sử dụng vòng lặp tuần tự vô hạn liên tục kiểm tra các cờ, các thanh ghi hoặc các biến trong bộ nhớ để nhận biết các sự kiện xảy ra và thực hiện các đoạn mã đáp ứng lại các sự kiện đó được gọi là một chương trình nhúng tuần tự. Khi chương trình nhúng "lắng nghe" các sự kiện và lựa chọn các hàm xử lý sự kiện tương ứng để thực thi rồi trở về trạng thái nghỉ để tiếp tục chờ các sự kiện tiếp theo thì chúng ta gọi đó là một chương trình nhúng hướng sự kiện. So với mô hình tuần tự, mô hình hướng sự kiện phản ánh trung thực hơn cơ chế hoạt động của một hệ thống nhúng.

Nếu coi hệ thống nhúng là một tập hữu hạn các trạng thái và hoạt động của hệ thống nhúng là việc chuyển từ trạng thái này sang trạng thái khác khi có sự kiện phát sinh thì ta có một máy trạng thái (state machine). Máy trạng thái được sử dụng nhiều trong các chương trình máy tính vì tính đơn giản và hiệu quả của nó trong việc thể hiện luồng hoạt động của chương trình. Sự kết hợp giữa máy trạng thái và mô hình hướng sự kiện có thể coi là sự kết hợp hiệu quả để tạo ra các chương trình nhúng có độ phức tạp lớn hoặc có tần số đáp ứng nhanh, tính realtime cao.

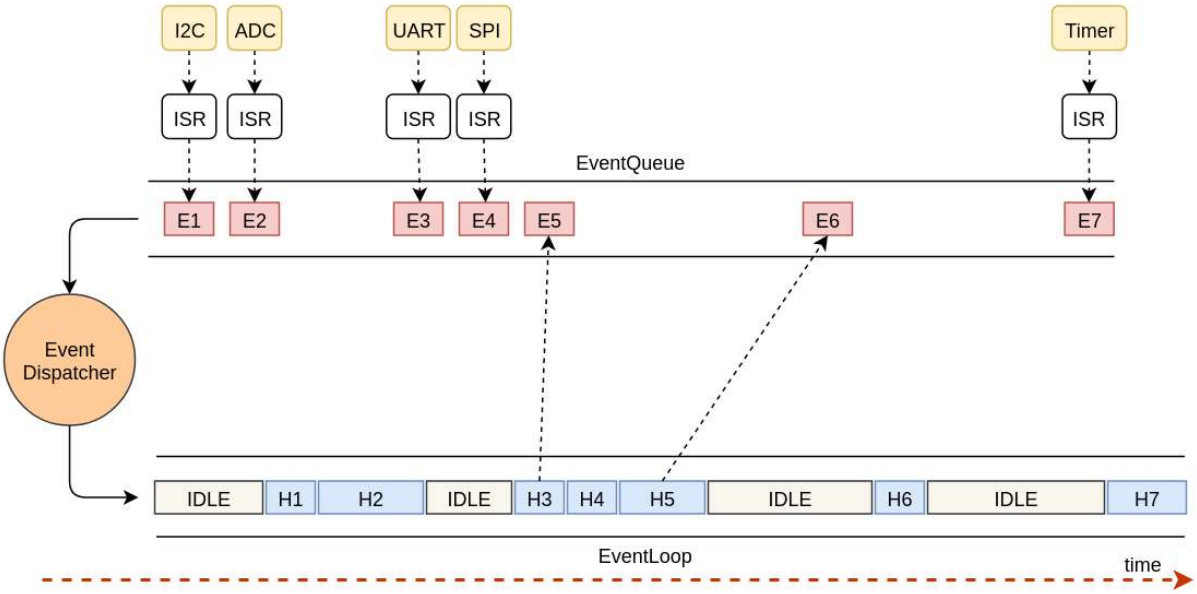
Bài báo này sẽ đề cập đến mô hình hướng sự kiện sử dụng trong các chương trình nhúng (Phần 2) và sự kết hợp của máy trạng thái với mô hình hướng sự kiện (Phần 3). Các lập luận để thiết kế một thư viện lõi cho máy trạng thái trên nền tảng hướng sự kiện sẽ được đề cập trong Phần 4, Phần 5 sẽ là các đánh giá và kết luận.

2. Mô hình hướng sự kiện

Ngày nay, hàng triệu máy tính để bàn và hàng tỉ thiết bị nhúng đang hoạt động dựa trên các sự kiện phát sinh từ bên ngoài hoặc bên trong hệ thống như một nút được nhấn, nhận được một gói tin, tín hiệu báo từ đồng hồ hệ thống,... Khi xác định được sự kiện, chúng phản ứng lại bằng cách lựa chọn một đoạn mã thích hợp để thực hiện. Đoạn mã đó có thể là các thao tác thay đổi trạng thái phần cứng hoặc sinh ra các "sự kiện mềm" để kích hoạt các đoạn mã xử lý khác của chương trình. Khi đoạn mã thực hiện xong, chương trình trở về trạng thái nghỉ ngơi tiếp tục chờ các sự kiện tiếp theo. Một hệ thống như vậy thường được gọi là hệ thống hướng sự kiện (event-driven system) hay một hệ thống phản ứng (reactive system).

Luồng hoạt động nói trên có nghĩa là khi một sự kiện xảy ra, CPU chỉ thực hiện một đoạn mã rất nhỏ so với toàn bộ mã của chương trình. Do đó thách thức chính của một hệ thống hướng sự kiện là việc nhanh chóng lựa chọn và thực thi đoạn mã tương ứng với sự kiện xảy ra. Đây không phải là vấn đề dễ dàng vì các sự kiện diễn ra một cách ngẫu nhiên về mặt thời gian và thứ tự xuất hiện. Hơn nữa, việc CPU bị đánh thức một cách thụ động không giống như các chương trình xử lý dữ liệu thông thường trên máy tính mà các bước được thực thi một cách tuần tự và xác định trước cho đến khi toàn bộ dữ liệu được xử lý xong. Các chương trình xử lý dữ liệu này (ví dụ một chương trình dịch) kiểm soát hoàn toàn quá trình thực thi từ đầu cho đến cuối. Ngược lại, các chương trình hướng sự kiện lại chỉ kiểm soát hệ thống một cách rời rạc khi có sự kiện xảy ra. Như vậy trong một hệ thống hướng sự kiện gần như không có khái niệm bắt đầu hay kết thúc rõ ràng.

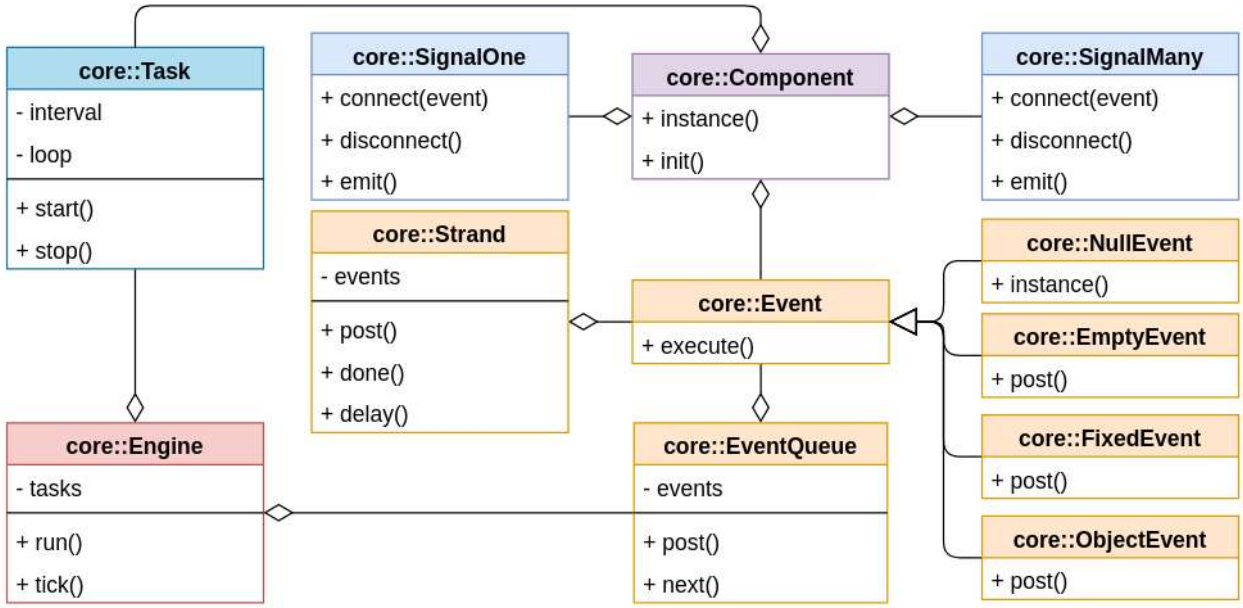
Chúng ta thấy rằng việc làm chủ mô hình hướng sự kiện là đặc biệt quan trọng với các lập trình viên khi ta biết rằng bản chất của một hệ thống nhúng là một hệ thống hướng sự kiện. Hơn nữa, các hệ thống hướng sự kiện có tầm quan trọng cơ bản trong toàn bộ ngành công nghiệp phần mềm nói chung, bởi vì phản ứng với các sự kiện là điều mà hầu hết các máy tính thường làm trong suốt quá trình hoạt động của chúng. Tuy nhiên, dường như sự hiểu biết về mô hình hướng sự kiện không phổ biến đối với các lập trình viên.



Hình 1: Mô hình hướng sự kiện

Một nền tảng hỗ trợ lập trình hướng sự kiện thường bao gồm một hoặc nhiều hàng đợi sự kiện (event-queue) một bộ phân phối sự kiện (event-dispatcher), các hàm xử lý sự kiện (event-handler) và một vòng lặp sự kiện (event-loop) như trong *Hình 1*. Với hệ thống nhúng, các sự kiện thường xuất phát từ các ngắt thông qua các hàm ISR (Interrupt Service Routine) để đánh thức CPU. Các sự kiện có thể xuất phát từ một hàm xử lý sự kiện nào đó như ví dụ trong *Hình 1*, hàm H3 sinh ra sự kiện E5 và H5 sinh ra sự kiện E6.

Trước khi đi vào thảo luận về máy trạng thái trong mô hình hướng sự kiện cho các hệ thống nhúng, chúng tôi sẽ giới thiệu sơ bộ về thư viện lõi hướng sự kiện (Event-Driven Core) mà chúng tôi xây dựng và sử dụng trong các dự án phát triển sản phẩm cũng như các dự án nghiên cứu công nghệ. Các thành phần của Event-Driven Core được mô tả trong *Hình 2* bao gồm: Engine, Component, EventQueue, Event, Task, Signal, Strand.



Hình 2: Mô hình Event-Driven Core

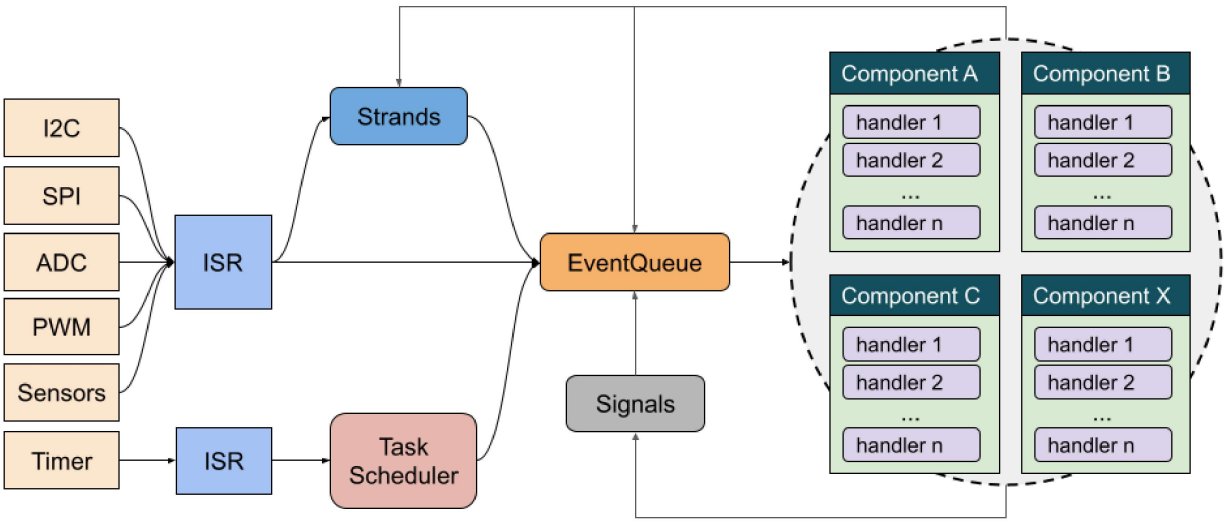
Engine và Component là khái niệm cơ bản trong Event-Driven Core có mục đích quản lý và phân chia logic hoạt động của chương trình thành các module. Hai thành phần này cung cấp công cụ cho người thiết kế mô hình hóa hệ thống một cách dễ dàng hơn và cho người quản lý dự án có thể kiểm soát tốt hơn quá trình phát triển sản phẩm. Engine cũng là nơi quản lý tất cả các Component, EventQueue và thực thi event-loop.

Các khái niệm hướng sự kiện được giới thiệu ở đây bao gồm:

- ***Event*** là lớp đại diện cho một sự kiện. Một sự kiện bao gồm loại sự kiện và các tham số (context) của sự kiện. Event-Driven Core cung cấp lớp EmptyEvent để dùng cho các sự kiện không có tham số, và 2 lớp cơ sở FixedEvent (cho các sự kiện có dữ liệu với kích thước cố định) và ObjectEvent (cho các sự kiện là một đối tượng tổng quát, có thể có kích thước thay đổi). NullEvent mà các bạn nhìn thấy trong diagram chính là Null Object của Event
- ***Task*** bản chất là một Timer mềm được Event-Driven Core cung cấp cho các tác vụ lặp lại theo thời gian có tần suất thấp (nhỏ hơn 1kHz). Event-Drivent Core sử dụng một bộ lập lịch trong Engine để kích hoạt các Task thông qua cơ chế hoạt động của Event. Bộ lập lịch này được kích hoạt mỗi mili giây một lần và sẽ xác định Task nào cần thực thi. Một Task có thể được chạy một lần (single-shot), nhiều lần (multi-shot) hoặc chạy vô hạn lần tùy theo nhu cầu của lập trình viên.
- **Signal** là khái niệm được Event-Driven Core đưa ra để giúp các Component có thể giao

tiếp với nhau mà không bị phụ thuộc lẫn nhau. Chúng tôi gọi mối liên kết này là "liên kết lỏng". Signal có hai loại cho hai mục đích sử dụng khác nhau: SignalOne là cho liên kết một-một (point-to-point) và SignalMany cho liên kết một-nhiều (publisher/subscriber).

- ***Strand*** là một dạng event-queue đặc biệt. Nó được dùng để kiểm soát luồng hoạt động của các sự kiện theo một thứ tự nhất định, ví dụ việc hai ngoại vi dùng chung bus SPI hoặc I2C chẳng hạn, các sự kiện xảy ra phải đúng theo thứ tự để tránh xung đột đường bus dẫn đến treo bus. Strand được thiết kế nhỏ gọn để nhúng trực tiếp vào các Component và tùy biến kích thước theo yêu cầu của từng vấn đề cụ thể.



Hình 3: Luồng tương tác của các thành phần trong Event-Driven Core

3. Máy trạng thái

Như đã phân tích ở trên, thách thức chính của một hệ thống nhúng hướng sự kiện là việc tổ chức chương trình thành các đoạn mã nhỏ xử lý các sự kiện và lựa chọn nhanh chóng và chính xác đoạn mã tương ứng với sự kiện xảy ra để thực hiện. Trong hầu hết các trường hợp, việc đáp ứng sự kiện không chỉ phụ thuộc vào đặc tính của sự kiện đó mà còn phụ thuộc vào lịch sử các sự kiện xảy ra trước đó trong hệ thống. Nhìn từ quan điểm của người lập trình, sự phụ thuộc này sẽ dẫn đến các cấu trúc rẽ nhánh ***if-else*** hoặc ***switch-case*** lồng nhau. Hầu hết các chương trình nhúng đều bắt đầu khá đơn giản và có cấu trúc tốt, nhưng khi các tính năng mới được ghép vào thì số lượng cờ và biến được sinh ra để lưu giữ trạng thái của hệ thống ngày càng gia tăng. Điều này dẫn đến nhiều lệnh ***if*** và ***else*** cần được thêm vào để kiểm tra các biểu thức logic ngày càng phức tạp được xây dựng từ nhiều biến và cờ khác nhau. Sự lồng nhau chồng chéo của các lệnh rẽ nhánh trong chương trình thường được gọi là spaghetti code. Sự phức tạp cứ tăng dần lên cho đến khi không ai có thể thực sự biết rõ phần nào của đoạn mã được thực thi để đáp ứng với một sự kiện nhất định nào đó.

Cách tiếp cận truyền thống đã bỏ qua yếu tố quan trọng là ngữ cảnh (context) của hệ thống dẫn đến cấu trúc chương trình bị phá vỡ bởi số lượng lớn logic điều kiện phức tạp. Nếu lập trình

viên có thể cắt giảm dù chỉ một phần nhỏ trong số các nhánh có điều kiện này thì chương trình sẽ trở lên dễ hiểu, dễ kiểm chứng và dễ bảo trì hơn rất nhiều. Máy trạng thái là công cụ tốt để giải quyết chính xác vấn đề nêu trên. Nó được coi là một cố máy “diệt spaghetti” mạnh mẽ giúp giảm đáng kể số lượng luồng thực thi thông qua các đoạn mã lệnh, đơn giản hóa các điều kiện tại các điểm phân nhánh và đơn giản hóa việc chuyển đổi giữa các chế độ thực thi khác nhau.

Máy trạng thái làm cho việc xử lý sự kiện phụ thuộc một cách rõ ràng vào cả bản chất của sự kiện và vào ngữ cảnh (trạng thái) của hệ thống. Một trạng thái có thể nắm bắt rất hiệu quả các khía cạnh liên quan của lịch sử hệ thống. Ví dụ: khi bạn gõ một phím trên bàn phím, mã ký tự được tạo sẽ là ký tự viết hoa hoặc viết thường, tùy thuộc vào việc Shift có hoạt động hay không. Do đó, bàn phím ở trạng thái "hoa", hoặc trạng thái "thường". Hoạt động của bàn phím chỉ phụ thuộc vào một số khía cạnh nhất định trong lịch sử của nó, cụ thể là liệu Shift có bị tắt hay không, nhưng không phụ thuộc vào số lượng và ký tự cụ thể đã được gõ trước đó. Do đó một trạng thái có thể loại bỏ tất cả các chuỗi sự kiện không liên quan và chỉ tập trung nắm bắt những chuỗi sự kiện có ý nghĩa.

Tại một thời điểm bất kỳ, hệ thống có một trạng thái nhất định nào đó, khi một sự kiện xảy ra hệ thống có thế sẽ được chuyển sang một trạng thái mới.

4. Đề xuất

TBD

5. Kết luận

TBD

Tài liệu tham khảo

TBD