



LẬP TRÌNH STM32, LẬP TRÌNH ARDUINO, LẬP TRÌNH ESP32, LẬP TRÌNH ESP8266, LẬP TRÌNH STM8

Tổng quan về hệ điều hành thời gian thực RTOS

POSTED ON 15/07/2021 BY KHUÊ NGUYỄN



Nếu đã là dân công nghệ, chắc hẳn bạn chẳng lạ lẫm gì với hệ điều hành hay OS, thế nhưng bạn đã nghe RTOS bao giờ chưa? RTOS là cái gì và cách nó hoạt động như thế nào? Tất cả sẽ có trong bài viết ngày hôm nay.



Mục Lục

- 1. Rtos là gì?
- 2. Khi nào bạn cần sử dụng RTOS ?
- 3. Tại sao lại phải dùng RTOS ?
- 4. Cách hoạt động của Rtos
- 5. Các khái niệm trong hệ điều hành thời gian thực RTOS
 - 5.1. Kernel – Nhân
 - 5.2. Task – Tác vụ
 - 5.3. Task States – Trạng thái Task
 - 5.4. Scheduler – Lập lịch
 - 5.5. Kết nối Inter-task & Chia sẻ tài nguyên
 - 5.6. Signal event
 - 5.7. Message queue – Hàng đợi tin nhắn
 - 5.8. Mail queue
 - 5.9. Semaphore
 - 5.10. Mutex
- 6. Kết
 - 6.1. Related posts:

Rtos là gì?

RTOS là viết tắt của cụm từ **Real-time operating system** hay hệ điều hành thời gian thực thường được nhúng trong các dòng vi điều khiển dùng để điều khiển thiết bị một cách nhanh chóng và đa nhiệm (multi tasking). Để hiểu rõ ràng nó là gì trước hết hãy làm rõ khái niệm về hệ điều hành đã.

Hệ điều hành (tiếng Anh: **Operating System** – viết tắt: **OS**) là một phần mềm dùng để điều hành, quản lý toàn bộ tất cả thành phần (bao gồm cả phần cứng và phần mềm) của thiết bị điện tử.

Nói đơn giản, hệ điều hành giống như hội đồng quản trị vậy. Họ có quyền quyết định ai làm gì và thời gian như thế nào. Các nhân viên cũng như các ứng dụng, nhận lệnh của cấp trên và thực thi các công việc theo đúng chức năng của mình.

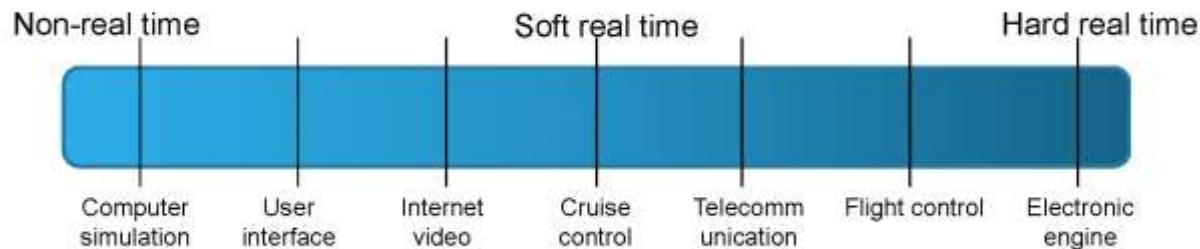
Vậy hệ điều hành thời gian thực với hệ điều hành bình thường khác gì nhau?

- **Hệ điều hành thông thường (non-realtime)**: như Window, linux, android, ios... chính là thứ mà chúng ta sử dụng hằng ngày. Khi mở một phần mềm trên đó, có thể bạn phải chờ nó tải rất lâu, việc chờ đợi này cũng không ảnh hưởng gì cả. Bởi vì đa số phần mềm đó tương tác với con người chứ ít tương tác với các phần mềm hoặc thiết bị khác
- **Hệ điều hành thời gian thực (realtime)**: sinh ra cho các tác vụ cần sự phản hồi nhanh của hệ thống, thường được nhúng trong các loại vi điều khiển và không có giao diện (GUI) tương tác với người dùng. Chúng cần phản hồi nhanh bởi vì đa số các tác vụ tương tác với thiết bị, máy móc khác chứ không phải con người. Các tài nguyên bên trong rất hữu hạn nên chỉ một sự chậm trễ cũng có thể làm hệ thống làm việc hoàn toàn sai lệch.

Bạn cứ thử tưởng tượng một hệ điều hành đang chạy các tác vụ điều khiển tên lửa mà độ trễ chỉ 2s. Với tốc độ của tên lửa cũng có thể bắn lệch từ Hà Nội thành TP Hồ Chí Minh rồi.

Thực tế hệ điều hành thời gian thực còn chia thành 2 loại:

- **Soft-realtime:** Sử dụng cho các ứng dụng cruise control (điều khiển hành trình) trong ô tô và các ứng dụng viễn thông
- **Hard-realtime:** Sử dụng trong các ứng dụng điều khiển máy bay, động cơ điện



Khi nào bạn cần sử dụng RTOS ?

Các ứng dụng không cần dùng RTOS

- Ứng dụng đơn (ứng dụng chỉ có 1 chức năng)
- Ứng dụng có vòng lặp đơn giản
- Ứng dụng <32kB

Các ứng dụng cần RTOS

- Ứng dụng nhiều trạng thái máy (States Machine)
- Ứng dụng lớn
- Ứng dụng liên quan tới các tác vụ xử lý nhanh, xử lý ảnh, âm thanh.

Nếu ứng dụng của bạn mà kích thước chương trình lớn dần và độ phức tạp tăng lên thì RTOS sẽ rất hữu dụng trong trường hợp này, lúc đó RTOS sẽ chia các ứng dụng phức tạp thành các phần nhỏ hơn và dễ quản lý hơn.

RTOS được sử dụng rất nhiều khi lập trình **ESP32**, **ESP8266**, **STM32** và các dòng chip khác.

Tại sao lại phải dùng RTOS ?

Chia sẻ tài nguyên một cách đơn giản: cung cấp cơ chế để phân chia các yêu cầu về bộ nhớ và ngoại vi của MCU

Dễ debug và phát triển: Mọi người trong nhóm có thể làm việc một cách độc lập, các lập trình viên thì có thể tránh được các tương tác với ngắn, timer, với phần cứng (cái này mình không khuyến khích lầm vì hiểu được phần cứng vẫn sẽ tốt hơn nhiều)

Tăng tính linh động và dễ dàng bảo trì: thông qua API của RTOS,...

Cách hoạt động của Rtos

RTOS là một phần đoạn hoặc một phần của chương trình, trong đó nó giải quyết việc điều phối các task, lập lịch và phân mức ưu tiên cho task, nắm bắt các thông điệp gửi đi từ task.

RTOS khá phức tạp, nói một cách dễ hiểu hơn là nó thực hiện việc xử lý các trạng thái máy (State Machine). Các bạn có thể tìm hiểu tại bài viết States Machine và lập trình nhúng

Để giải quyết một bài toán nhiều trạng thái máy, thông thường chúng ta sử dụng code sau:

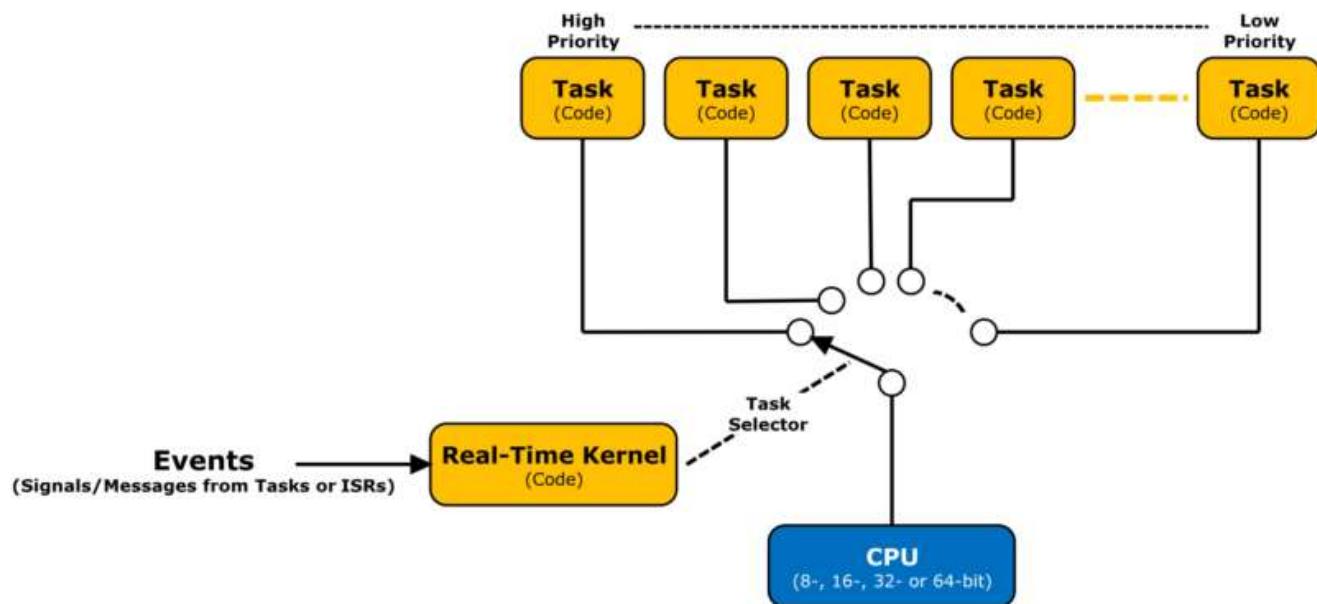
```
while(1)
{
    switch(state)
    {
        case 1: //Code for Task 1;
                  state= 2;
        case 2: //Code for Task 2;
                  state= 3;
        case 3: //Code for Task 3;
                  state= 4;
        case 4: //Code for Task 4;
                  state=1;
    }
}
```

Bạn có thể thấy, chương trình sẽ thực thi từ states 1 tới states 4 sau đó quay vòng lại. Bất kì khi nào states thay đổi, chương trình sẽ nhảy qua phục vụ task đó.

Ví dụ: nếu trong task 1 có lệnh states = 4, thì ngay sau khi Task 1 được thực thi xong chương trình sẽ nhảy qua Task 4 mà bỏ qua Task 2 và 3.

Nhược điểm của phương pháp này đó là tài nguyên sử dụng chung, tốc độ chuyển chậm khi thay đổi states bởi nó phải hoàn thành mỗi Task trước khi chuyển sang Task khác, khó kiểm soát khi nhiều tác vụ (Task)

Vậy nên RTOS ra đời giải quyết các nhược điểm trên.



Cách hoạt động của RTOS

Nhân Kernel sẽ điều phối sự hoạt động của các tác vụ (Task), mỗi task sẽ có một mức ưu tiên (prioritize) và thực thi theo chu kỳ cố định. Nếu có sự tác động như ngắt, tín hiệu hoặc tin nhắn giữa các Task, Kernel sẽ điều phối chuyển tới Task tương ứng với Code đó.

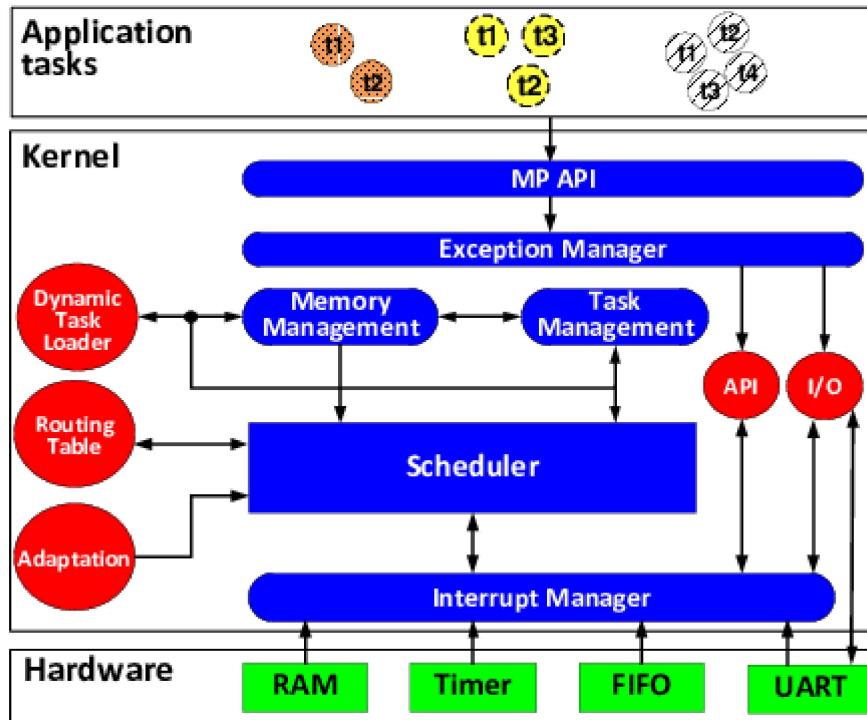
Sự chuyển dịch giữa các Task rất linh động, độ trễ thấp mang lại độ tin cậy cao cho chương trình.

Các khái niệm trong hệ điều hành thời gian thực RTOS

Kernel – Nhân

Kernel hay còn gọi là Nhân có nhiệm vụ quản lý và điều phối các Task. Mọi sự kiện (Event) như ngắn, Timer, data truyền tới... đều qua Kernel xử lý để quyết định xem nên làm gì tiếp theo.

Thời gian xử lý của Kernel thường rất nhanh nên độ trễ rất thấp.



Task – Tác vụ

Bạn cứ tưởng tượng một chương trình là một công ty, với ông to nhất là Giám đốc – Kernel. Ông này chỉ điều hành và chả biết nghiệp vụ gì cả, để thực hiện các nghiệp vụ khác nhau, công ty đó cần các nhân viên. Và các nhân viên đó gọi là các Task.

Nói đơn giản, Task là một đoạn chương trình thực thi một hoặc nhiều vấn đề gì đó, được Kernel quản lý.

Kernel sẽ quản lý việc chuyển đổi giữa các task, nó sẽ lưu lại ngữ cảnh của task sắp bị hủy và khôi phục lại ngữ cảnh của task tiếp theo bằng cách:

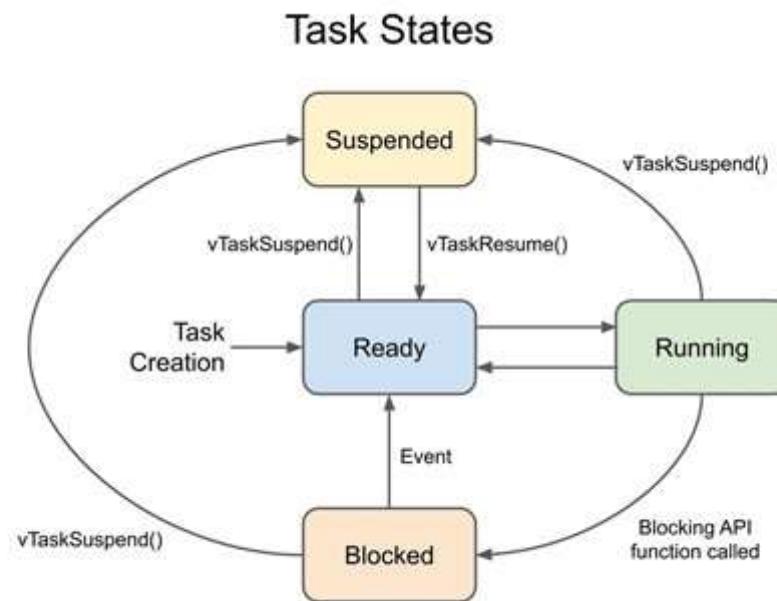
- Kiểm tra thời gian thực thi đã được định nghĩa trước (time slice được tạo ra bởi ngắt systick)
- Khi có các sự kiện unblocking một task có quyền cao hơn xảy ra (signal, queue, semaphore,...)
- Khi task gọi hàm Yield() để ép Kernel chuyển sang các task khác mà không phải chờ cho hết time slice
- Khi khởi động thì kernel sẽ tạo ra một task mặc định gọi là Idle Task.

Task States – Trạng thái Task

Một ông nhân viên thì cũng có lúc này lúc kia, lúc rảnh rỗi, lúc đang làm việc, hay cũng có lúc nghỉ vì... vợ đẻ. Những lúc như vậy Kernel phải điều phối làm sao cho chương trình chạy hợp lý.

Không thể bắt ông đang trông vợ đẻ đi làm được, đúng không?

Một task trong RTOS thường có các trạng thái như sau



Trạng thái Task trong Rtos

- **RUNNING:** đang thực thi
- **READY:** sẵn sàng để thực hiện
- **WAITING:** chờ sự kiện
- **INACTIVE:** không được kích hoạt

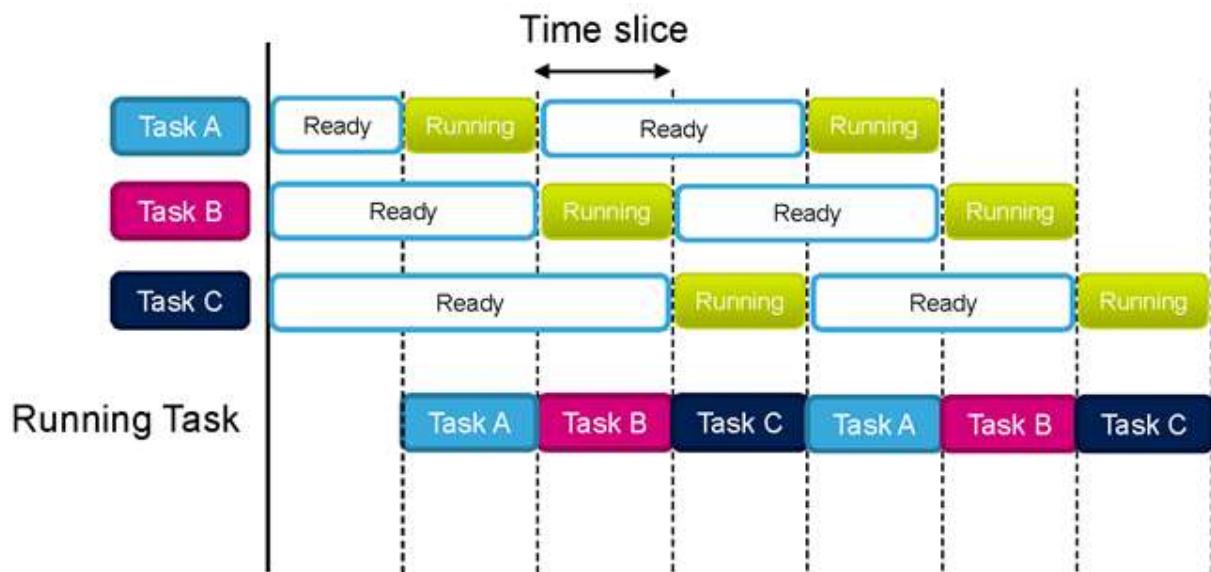
Scheduler – Lập lịch

Nói đơn giản thì Scheduler là một bé thư ký hoặc trợ lý của ông giám đốc Kernel. Bé này sẽ có nhiệm vụ nhắc nhở ông xếp mình cần làm gì tại các thời điểm khác nhau.

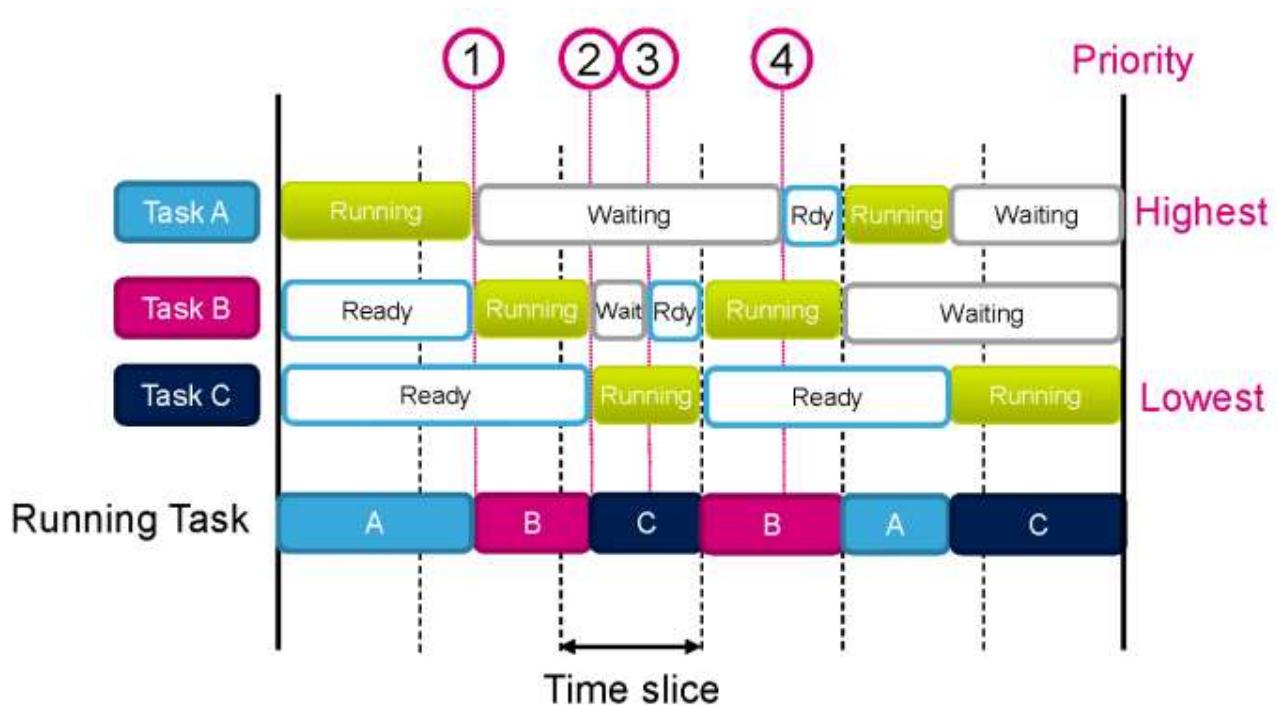
Đây là 1 thành phần của kernel quyết định task nào được thực thi. Có một số luật cho scheduling như:

Cooperative: giống với lập trình thông thường, mỗi task chỉ có thể thực thi khi task đang chạy dừng lại, nhược điểm của nó là task này có thể dùng hết tất cả tài nguyên của CPU

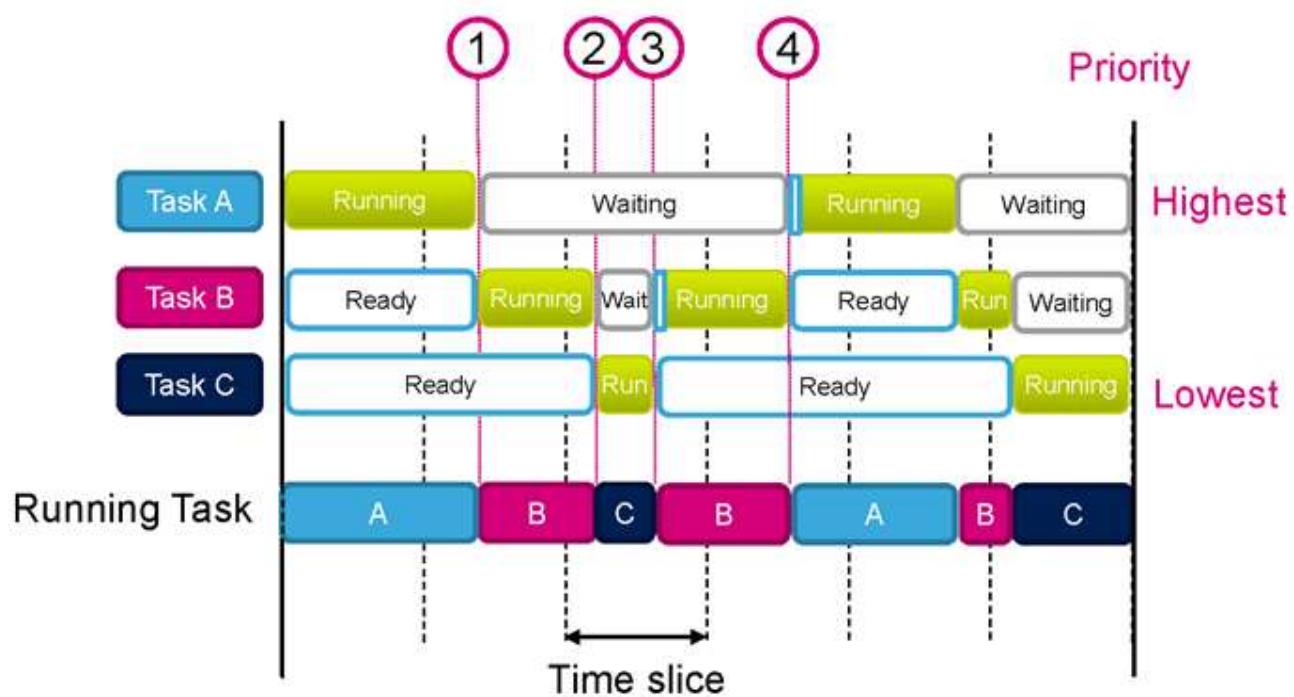
Round-robin: mỗi task được thực hiện trong thời gian định trước (time slice) và không có ưu tiên.



Priority base: Task được phân quyền cao nhất sẽ được thực hiện trước, nếu các task có cùng quyền như nhau thì sẽ giống với round-robin, các task có mức ưu tiên thấp hơn sẽ được thực hiện cho đến cuối time slice



Priority-based pre-emptive: Các task có mức ưu tiên cao nhất luôn nhường các task có mức ưu tiên thấp hơn thực thi trước.



Kết nối Inter-task & Chia sẻ tài nguyên

Để cho công ty vận hành được trơn chu, thì các nhân viên cũng phải liên lạc và trao đổi với nhau. Làm coder thì cũng phải nói chuyện với ông làm Sales để xem mình code vậy có bán được không, làm nhân viên cũng phải hiểu các nghiệp vụ của nhau, chứ không xung đột tài nguyên là vỡ đầu chảy máu.

Vậy nên task cần phải kết nối và trao đổi dữ liệu với nhau để có thể chia sẻ tài nguyên, có một số khái niệm cần lưu ý:

Với Inter-task Communication:

- Signal Events – Đồng bộ các task
- Message queue – Trao đổi tin nhắn giữa các task trong hoạt động giống như FIFO
- Mail queue – Trao đổi dữ liệu giữa các task sử dụng hằng đợi của khối bộ nhớ

Với Resource Sharing:

- Semaphores – Truy xuất tài nguyên liên tục từ các task khác nhau

- Mutex – Đồng bộ hóa truy cập tài nguyên sử dụng Mutual Exclusion

Signal event

Signal event được dùng để đồng bộ các task, ví dụ như bắt task phải thực thi tại một sự kiện nào đó được định sẵn

Ví dụ: Một cái máy giặt có 2 task là Task A điều khiển động cơ, Task B đọc mức nước từ cảm biến nước đầu vào

- Task A cần phải chờ nước đầy trước khi khởi động động cơ. Việc này có thể thực hiện được bằng cách sử dụng signal event
- Task A phải chờ signal event từ Task B trước khi khởi động động cơ
- Khi phát hiện nước đã đạt tới mức yêu cầu thì Task B sẽ gửi tín hiệu tới Task A

Với trường hợp này thì task sẽ đợi tín hiệu trước khi thực thi, nó sẽ nằm trong trạng thái là WAITING cho đến khi signal được set. Ngoài ra ta có thể set 1 hoặc nhiều signal trong bất kỳ các task nào khác.

Mỗi task có thể được gán tối đa là 32 signal event

Message queue – Hàng đợi tin nhắn

Nói nôm na là một group chat trong một công ty để trao đổi giữa các thành viên.

Message queue là cơ chế cho phép các task có thể kết nối với nhau, nó là một **FIFO** (First In First Out) buffer được định nghĩa bởi độ dài (số phần tử mà buffer có thể lưu trữ) và kích thước dữ liệu (kích thước của các thành phần trong buffer).

Một ứng dụng tiêu biểu là buffer cho Serial I/O, buffer cho lệnh được gửi tới task

Task có thể ghi vào hàng đợi (queue)

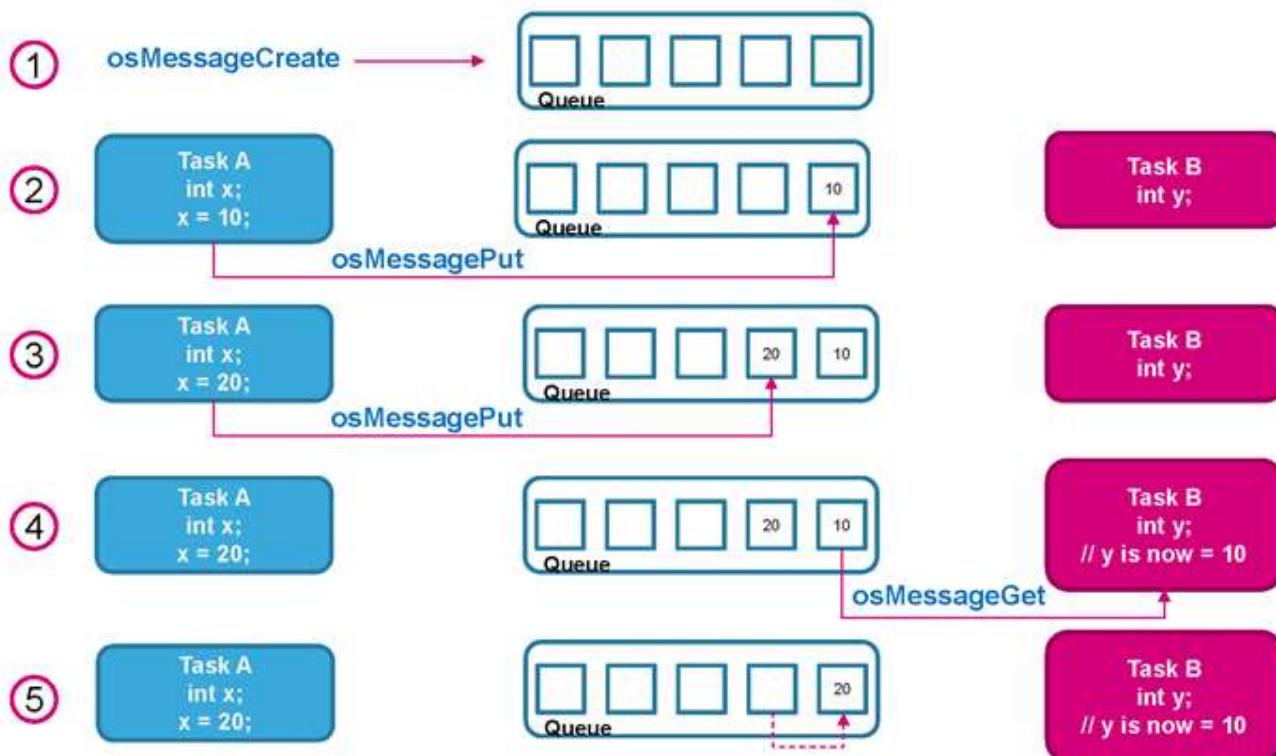
- Task sẽ bị khóa (block) khi gửi dữ liệu tới một message queue đầy đủ
- Task sẽ hết bị khóa (unblock) khi bộ nhớ trong message queue trống

- Trường hợp nhiều task mà bị block thì task với mức ưu tiên cao nhất sẽ được unblock trước

Task có thể đọc từ hằng đợi (queue)

- Task sẽ bị block nếu message queue trống
- Task sẽ được unblock nếu có dữ liệu trong message queue.
- Tương tự ghi thì task được unblock dựa trên mức độ ưu tiên

Ví dụ:



Mail queue

Nói nôm na giống như các bạn trao đổi Email giữa các nhân viên với nhau. Các email này sẽ bao gồm nhiều thứ như văn bản và các file đính kèm.

Mail queue truyền dữ liệu sẽ được truyền dưới dạng khối(memory block) thay vì dạng đơn. Mỗi memory block thì cần phải cấp phát trước khi đưa dữ liệu vào và

giải phóng sau khi đưa dữ liệu ra.

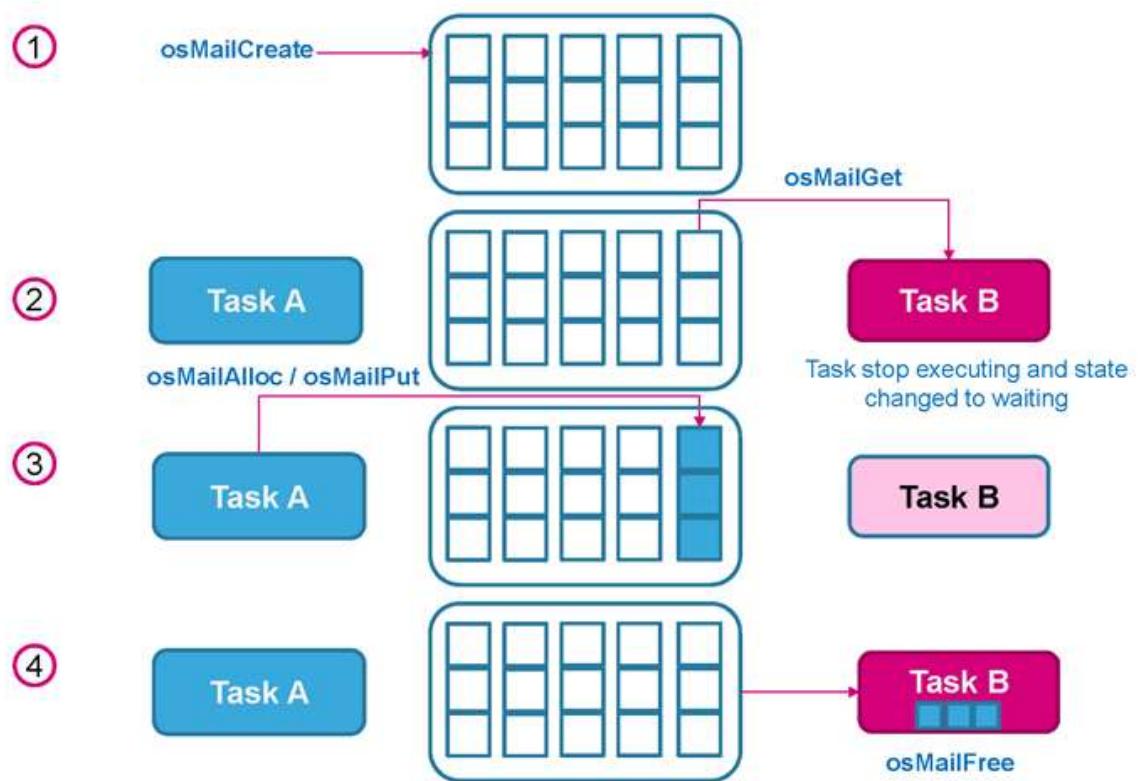
Thao tác gửi dữ liệu với mail queue

1. Cấp phát bộ nhớ từ mail queue cho dữ liệu được đặt trong mail queue
2. Lưu dữ liệu cần gửi vào bộ nhớ đã được cấp phát
3. Đưa dữ liệu vào mail queue

Thao tác nhận dữ liệu trong mail queue bởi task khác

1. Lấy dữ liệu từ mail queue, sẽ có một hàm để trả lại cấu trúc/ đối tượng
2. Lấy con trỏ chứa dữ liệu
3. Giải phóng bộ nhớ sau khi sử dụng dữ liệu

Ví dụ:



Semaphore

Nói đơn giản Semaphore là một thủ quỹ của công ty bạn vậy. Mỗi một công ty sẽ có 1 lượng tiền cố định, vậy nên nếu nhân viên ông nào cũng đòi tiền để làm cái này cái kia, mà không lấy lại về cho công ty, thì chắc chắn phá sản.

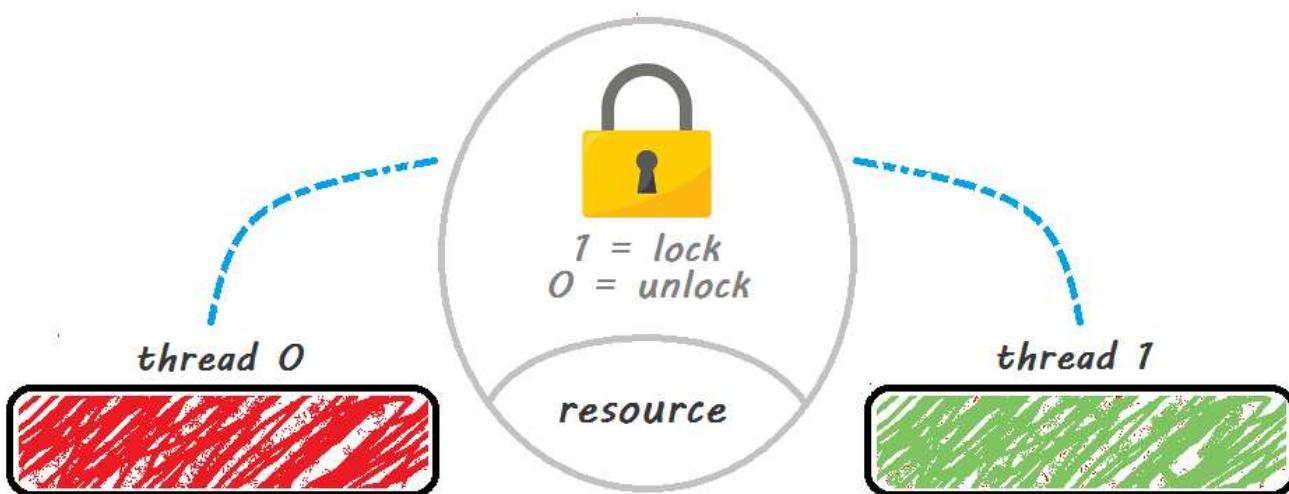
Trong vi điều khiển chúng ta có Semaphore để quản lý những tài nguyên (Resource) như Ram, ngoại vi ...

Được sử dụng để đồng bộ task với các sự kiện khác trong hệ thống. Có 2 loại

Binary semaphore

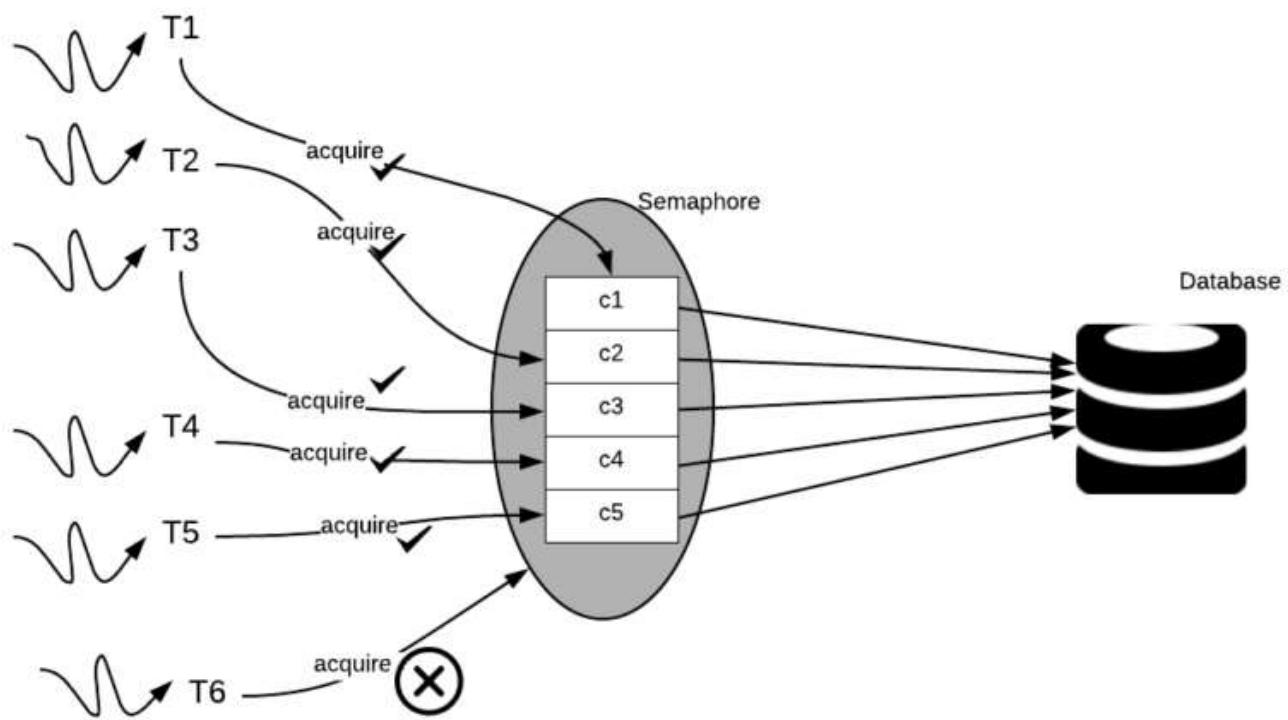
Semaphore là một mã thông báo giống như một mã thông báo cho phép một tác vụ thực hiện việc thực thi nếu tác vụ nhận được semaphore. Nếu không, tác vụ vẫn ở trạng thái khôi và không thể thực thi trừ khi nó có được semaphore nhị phân

- Có duy nhất 1 token
- Chỉ có 1 hoạt động đồng bộ



Counting semaphore

Counting semaphore đưa ra các Token cho các Task sử dụng tài nguyên, nếu tài nguyên được sử dụng hết thì các Task còn lại sẽ phải chờ đến khi Tài nguyên được giải phóng trở lại



Couting semaphore được dùng để:

Counting event

- Một event handler sẽ ‘give’ semaphore khi có event xảy ra (tăng giá trị đếm semaphore)
- Một task handler sẽ ‘take’ semaphore khi nó thực thi sự kiện (giảm giá trị đếm semaphore)
- Count value là khác nhau giữa số sự kiện xảy ra và số sự kiện được thực thi
- Trong trường hợp counting event thì semaphore được khởi tạo giá trị đếm bằng 0

Resource management

- Count value sẽ chỉ ra số resource sẵn có
- Để điều khiển và kiểm soát được resource của task dựa trên count value của semaphore (giá trị giảm), nếu count value giảm xuống bằng 0 nghĩa là không có resource nào free.

- Khi một task finish với resource thì nó sẽ give semaphore trở lại để tăng count value của semaphore.
- Trong trường hợp resouce management thì count value sẽ bằng với giá trị max của count value khi semaphore được tạo.

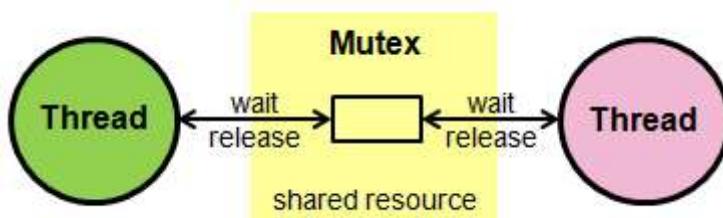
Mutex

Mutex giống như ông thủ kho của công ty bạn vậy. Vì tài nguyên của công ty hữu hạn, giả sử khi nhân viên A đang mượn sử dụng cái đồng hồ bấm giờ, nếu nhân viên B tới mượn chiếc đồng hồ đó thì Mutex sẽ bắt nhân viên B chờ cho tới khi Nhân viên A trả lại.

Sử dụng cho việc loại trừ (mutial exclusion), hoạt động như là một token để bảo vệ tài nguyên được chia sẻ. Một task nếu muốn truy cập vào tài nguyên chia sẻ

- Cần yêu cầu (đợi) mutex trước khi truy cập vào tài nguyên chia sẻ (Sharing Resource)
- Đưa ra token khi kết thúc với tài nguyên.

Tại mỗi một thời điểm thì chỉ có 1 task có được mutex. Những task khác muốn cùng mutex thì phải block cho đến khi task cũ thả mutex ra.



Về cơ bản thì Mutex giống như binary semaphore nhưng được sử dụng cho việc loại trừ chứ không phải đồng bộ. Ngoài ra thì nó bao gồm cơ chế thừa kế mức độ ưu tiên(Priority inheritance mechanism) để giảm thiểu vấn đề đảo ngược ưu tiên, cơ chế này có thể hiểu đơn giản qua ví dụ sau:

- Task A (low priority) yêu cầu mutex
- Task B (high priority) muốn yêu cầu cùng mutex trên.

- Mức độ ưu tiên của Task A sẽ được đưa tạm về Task B để cho phép Task A được thực thi
- Task A sẽ thả mutex ra, mức độ ưu tiên sẽ được khôi phục lại và cho phép Task B tiếp tục thực thi.

Kết

Và đó là tất cả những lý thuyết về hệ điều hành thời gian thực RTOS, có thể nghe nói thì rất là mông lung, huyền ảo và có phần hại não. Nhưng thực ra nó cũng không khó chút nào.

Vậy nên hãy tiếp tục đến với những bài tiếp theo để biết RTOS hoạt động trên **STM32** như thế nào nhé!

Bài viết này được tham khảo tại nguồn: [học arm](#), [thanhnt](#), [deviot](#)... và một số nguồn nước ngoài khác

Nếu thấy hay, hãy like và tham gia vào nhóm những người anh em [Nghiên Lập Trình](#) để giao lưu và học hỏi nhé!

4.8/5 - (9 bình chọn)

Related Posts:

1. [Hướng dẫn đọc Datasheet cho sinh viên điện tử và lập trình nhúng](#)
2. [Bài 7: Lập trình ESP32 Touch Pin bật tắt led với một cái chạm tay](#)
3. [Hướng dẫn cài đặt Platform IO lập trình ESP32](#)
4. [Bài 18: Lập trình SMT32 USB HID Keyboard bàn phím máy tính](#)
5. [Bài 1: Hướng dẫn cài đặt Arduino IDE và cách thêm thư viện](#)
6. [Bài 8: Lập trình STM32 đọc ADC một kênh](#)



KHUÊ NGUYỄN

Chỉ là người đam mê điện tử và lập trình. Làm được gì thì viết cho anh em xem thôi. :D

Trả lời

Email của bạn sẽ không được hiển thị công khai. Các trường bắt buộc được đánh dấu *

Bình luận *

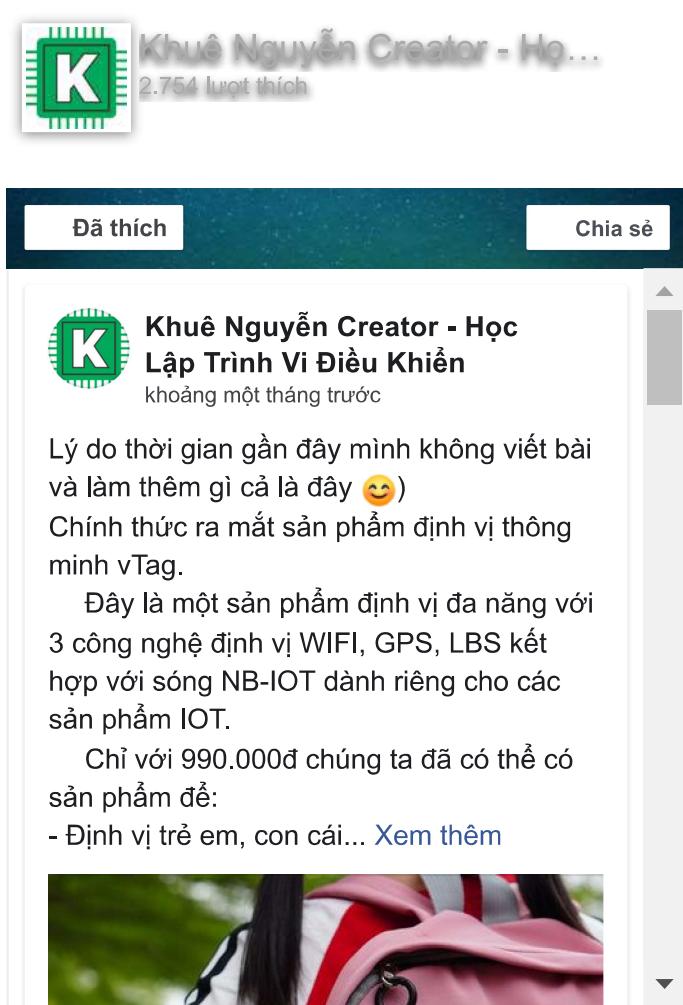
Tên *

Email *

Trang web

PHẢN HỒI

Fanpage



Khuê Nguyễn Creator - Học Lập Trình Vi Điều Khiển
khoảng một tháng trước

Lý do thời gian gần đây mình không viết bài và làm thêm gì cả là đây 😊
Chính thức ra mắt sản phẩm định vị thông minh vTag.

Đây là một sản phẩm định vị đa năng với 3 công nghệ định vị WIFI, GPS, LBS kết hợp với sóng NB-IOT dành riêng cho các sản phẩm IOT.

Chỉ với 990.000đ chúng ta đã có thể có sản phẩm để:

- Định vị trẻ em, con cái... [Xem thêm](#)



Bài viết khác



Lập trình 8051 - AT89S52

MICROCHIP AT89S51

Khuê Nguyễn Creator

PROTEUS

Bài 1: Tổng quan về 8051 và chip AT89S51 - 52

Tổng quan về 8051

8051 là một dòng chip nhập môn cho lập trình viên nhúng, chúng được sử...

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX



Khuê Nguyễn Creator



Lập trình STM32 HID Host giao tiếp với chuột và bàn phím

Lập trình STM32 USB HID Host giao tiếp với chuột và bàn phím máy tính

Trong bài này chúng ta sẽ cùng học STM32 HID Host, biến STM32 giống như...

[ĐỌC THÊM](#)



Khuê Nguyễn Creator



Lộ trình học lập trình nhúng từ A tới Z

Lộ trình học lập trình nhúng từ A tới Z

Lập trình nhúng là một ngành có cơ hội nhưng cũng đòi hỏi nhiều kiến...

3 COMMENTS

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX



Khuê Nguyễn Creator



Lập trình STM32F407 SDIO đọc dữ liệu thẻ nhớ

Lập trình STM32 SDIO đọc ghi dữ liệu vào thẻ nhớ SD card

Trong bài này chúng ta cùng học cách lập trình STM32 SDIO, một chuẩn giao...

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX



Khuê Nguyễn Creator



Lập trình STM32F407 DAC chuyển đổi số sang tương tự

Lập trình STM32 DAC tạo sóng hình Sin trên KIT STM32F407 Discovery

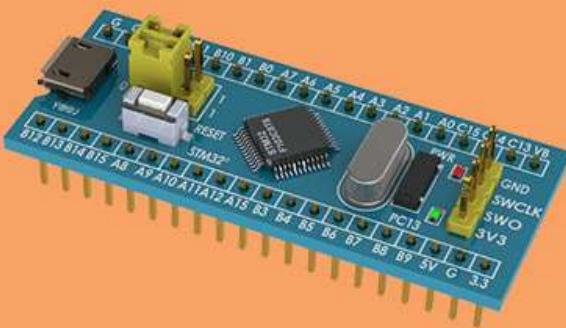
Trong bài này chúng ta sẽ cùng nhau tìm hiểu STM32 DAC với KIT STM32F407VE...

[ĐỌC THÊM](#)

Lập trình STM32 và CubeMX



Khuê Nguyễn Creator



Sử dụng hàm printf để in Log khi Debug trên STM32

Hướng dẫn sử dụng printf với STM32 Uart để in Log trên Keil C

Trong bài này chúng ta sẽ học cách retarget hàm printf của thư viện stdio...

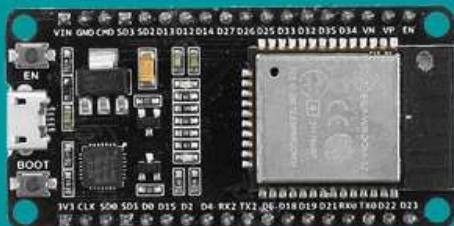
3 COMMENTS

[ĐỌC THÊM](#)

ESP32 và Platform IO



Khuê Nguyễn Creator



Bài 9 WIFI: Lập trình ESP32 OTA nạp firmware trên Internet

Lập trình ESP32 FOTA nạp firmware qua mạng Internet với OTA Drive

Trong bài này chúng ta sẽ học cách sử dụng ESP32 FOTA (Firmware Over The...

4 COMMENTS

[ĐỌC THÊM](#)

Lập trình Nuvoton



Khuê Nguyễn Creator

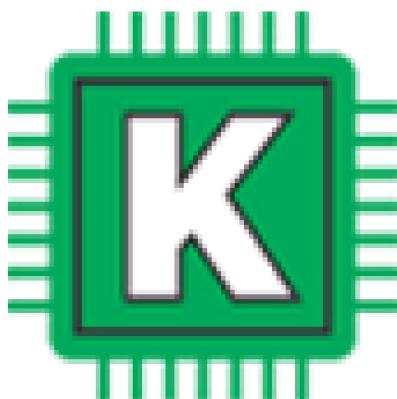


Cài đặt SDC Complier và Code:Blocks IDE

Hướng dẫn cài đặt SDCC và Code::Blocks lập trình Nuvoton

Ở bài này chúng ta sẽ cài đặt các công cụ cần thiết cho việc...

ĐỌC THÊM



KHUÊ NGUYỄN CREATOR
Chia sẻ đam mê

Blog này làm ra để lưu trữ tất cả những kiến thức, những câu chuyện của mình. Đôi khi là những ý tưởng nhất thời, đôi khi là các dự án tự mình làm. Chia sẻ cho người khác

cũng là niềm vui của mình, kiến thức mỗi người là khác nhau, không hẳn quá cao siêu nhưng sẽ có lúc hữu dụng.

DMCA PROTECTED

Liên Kết

Nhóm: Nghịen Lập Trình

Fanpage: Khuê Nguyên Creator

My Shop

Thông Tin

Tác Giả

Chính Sách Bảo Mật



Copyright 2022 © Khuê Nguyễn