

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ

-----*



BÀI TẬP LỚN
LẬP TRÌNH HỆ THỐNG NHÚNG
ĐỀ TÀI: HỆ THỐNG GIÁM SÁT MÁY

Sinh Viên Thực Hiện:	Nguyễn Gia Huy	2013309
Nhóm 30	Nguyễn Viễn Thông	2114918
Lớp L01	Trần Ngô Hoàng Anh	2112817

Giảng Viên Hướng Dẫn: Nguyễn Phan Hải Phú

HỌC KÌ: 241

Thành phố Hồ Chí Minh , ngày 1 tháng 12 năm 2024

TÓM TẮT ĐỀ TÀI

Đề tài tập trung vào việc nghiên cứu và ứng dụng **RTOS**, đặc biệt là **FreeRTOS**, để xây dựng hệ thống nhúng thời gian thực. Bên cạnh đó, đề tài tích hợp các công cụ như Apps script và Google sheets để quản lý dữ liệu. Phần cứng được sử dụng chính bao gồm vi xử lý **ESP32** và mô-đun hiển thị **OLED**, cảm biến nhiệt độ-độ ẩm **AHT30**, **IR vật cản**, **led**, **buzzer** ,**rung** .

Hệ thống được thiết kế nhằm tối ưu hóa hiệu suất, đảm bảo tính linh hoạt và khả năng đáp ứng thời gian thực cao, phù hợp với các ứng dụng công nghiệp và điều khiển thông minh.

MỤC LỤC

TÓM TẮT ĐỀ TÀI	ii
MỤC LỤC	iii
DANH MỤC HÌNH ẢNH	iv
CHƯƠNG I: GIỚI THIỆU ĐỀ TÀI	1
1.1. Bối cảnh và lý do chọn đề tài	1
1.2. Tầm quan trọng của hệ thống thời gian thực	1
1.3. Mục tiêu và phạm vi nghiên cứu của đề tài	2
1.3.1. Mục tiêu nghiên cứu	2
1.3.2. Phạm vi nghiên cứu	2
CHƯƠNG II: CƠ SỞ LÝ THUYẾT	3
2.1. RTOS là gì?	3
2.1.1. Khái niệm RTOS	3
2.1.2. Cách hoạt động của RTOS	4
2.1.3. Các khái niệm trong hệ điều hành thời gian thực RTOS	4
2.2. FreeRTOS	12
2.3. Phương thức truyền tải dữ liệu HTTP	13
2.3.1. Mô hình giao thức	14
2.3.2. Format gói tin	15
CHƯƠNG III: THIẾT KẾ PHẦN CỨNG	18
3.1. Phân tích thiết kế	18
3.2. Sơ đồ khối chi tiết	22
CHƯƠNG IV: THIẾT KẾ PHẦN MỀM	23
4.1. Máy trạng thái	23
4.1.1. Khái niệm máy trạng thái	23
4.1.2. Phân tích máy trạng thái đề tài	24
4.2. Lưu đồ giải thuật	25
CHƯƠNG V: KẾT QUẢ THỰC HIỆN	32
CHƯƠNG VI: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	34
CHƯƠNG VII: TÀI LIỆU THAM KHẢO	35

DANH MỤC HÌNH ẢNH

Figure 1 Kết quả Serial Monitor VS Code	32
Figure 2 Kết quả xuất ra OLED.....	33
Figure 3 Kết quả post data Google Sheets	33

CHƯƠNG I: GIỚI THIỆU ĐỀ TÀI

1.1. Bối cảnh và lý do chọn đề tài

Trong bối cảnh hiện nay, ngành công nghiệp sản xuất đang không ngừng tiến tới tự động hóa và số hóa để nâng cao hiệu suất và tối ưu hóa quy trình vận hành. Các máy móc công nghiệp, vốn là xương sống của dây chuyền sản xuất, yêu cầu được giám sát chặt chẽ để phát hiện sớm các sự cố, giảm thiểu thời gian chết và đảm bảo chất lượng sản phẩm.

Tuy nhiên, việc giám sát thủ công không chỉ mất thời gian mà còn dễ xảy ra sai sót. Vì vậy, nhu cầu xây dựng một **hệ thống giám sát thời gian thực** cho máy móc công nghiệp trở nên cấp thiết. Hệ thống này không chỉ giúp theo dõi trạng thái hoạt động của máy móc mà còn hỗ trợ cảnh báo và phân tích hiệu suất để đưa ra các quyết định kịp thời, nâng cao độ tin cậy và hiệu quả trong sản xuất.

1.2. Tầm quan trọng của hệ thống thời gian thực trong ứng dụng nhúng hiện nay

Hệ thống thời gian thực là một trong những thành phần quan trọng trong các ứng dụng nhúng, đặc biệt là trong lĩnh vực giám sát và điều khiển. Với khả năng xử lý nhanh và đáp ứng tức thời, hệ thống thời gian thực đảm bảo:

- **Phát hiện kịp thời sự cố:** Ngăn chặn hư hỏng lớn bằng cách phát hiện sớm các bất thường trong vận hành máy móc.
- **Giảm thiểu thời gian chết:** Tăng năng suất sản xuất thông qua việc cung cấp thông tin tức thì cho đội ngũ bảo trì.
- **Tối ưu hóa quy trình sản xuất:** Cung cấp dữ liệu thời gian thực giúp các nhà quản lý phân tích và tối ưu hóa vận hành.
- **Nâng cao an toàn:** Giám sát liên tục giúp giảm nguy cơ tai nạn do máy móc hỏng hóc hoặc vận hành sai cách.

Nhờ sự phát triển của các bộ vi điều khiển mạnh mẽ và công nghệ IoT, việc tích hợp hệ thống thời gian thực trong các ứng dụng giám sát công nghiệp trở nên khả thi hơn bao giờ hết, góp phần thúc đẩy cuộc cách mạng công nghiệp 4.0.

1.3. Mục tiêu và phạm vi nghiên cứu của đề tài

1.3.1. Mục tiêu nghiên cứu

- Xây dựng một hệ thống giám sát thời gian thực cho máy móc công nghiệp, hỗ trợ việc theo dõi và phân tích trạng thái hoạt động của thiết bị.
- Thiết kế hệ thống dựa trên vi điều khiển STM32, tích hợp cảm biến và giao diện truyền thông để thu thập và xử lý dữ liệu thời gian thực.
- Phát triển giao diện hiển thị dữ liệu giúp người vận hành dễ dàng giám sát và nhận cảnh báo khi xảy ra sự cố.

1.3.2. Phạm vi nghiên cứu

- Tập trung vào giám sát trạng thái vận hành của một số loại máy công nghiệp phổ biến.
- Sử dụng cảm biến đo lường các thông số như nhiệt độ, độ rung, tốc độ quay, và dòng điện để đánh giá tình trạng thiết bị.
- Tích hợp các phương thức truyền thông như UART, SPI, hoặc giao thức IoT để truyền dữ liệu từ máy móc đến máy chủ hoặc thiết bị đầu cuối.
- Đề tài sẽ không đi sâu vào điều khiển máy móc mà chỉ tập trung vào giám sát và cảnh báo.

Bằng việc hoàn thành nghiên cứu này, hệ thống hứa hẹn sẽ giúp nâng cao hiệu quả giám sát máy công nghiệp, giảm thiểu rủi ro và tăng cường khả năng vận hành ổn định trong môi trường công nghiệp.

CHƯƠNG II: CƠ SỞ LÝ THUYẾT

2.1. RTOS là gì?

2.1.1. Khái niệm RTOS

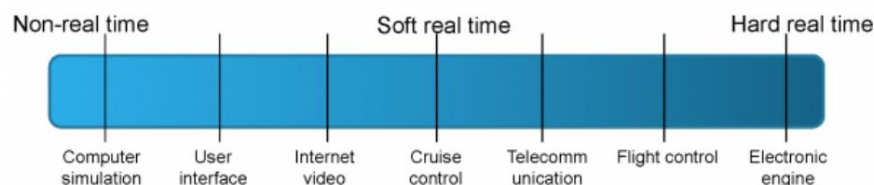
RTOS là viết tắt của cụm từ **Real-time operating system** hay hệ điều hành thời gian thực thường được nhúng trong các dòng vi điều khiển dùng để điều khiển thiết bị một cách nhanh chóng và đa nhiệm (multi tasking). Để hiểu rõ ràng nó là gì trước hết hãy làm rõ khái niệm về hệ điều hành đã.

Hệ điều hành (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.

- **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.

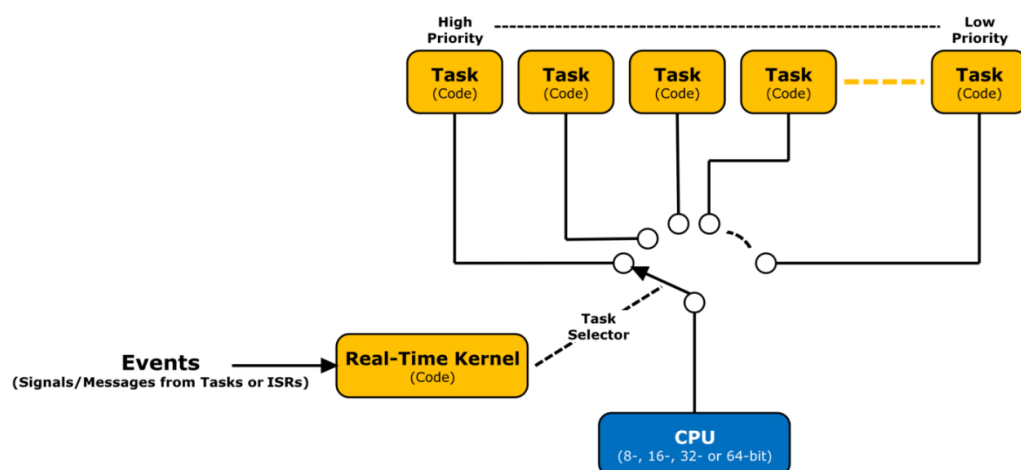
Thực tế hệ điều hành thời gian thực còn chia thành 2 loại:

- **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



2.1.2. Cách hoạt động của RTOS

RTOS là một phân 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, 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.



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.

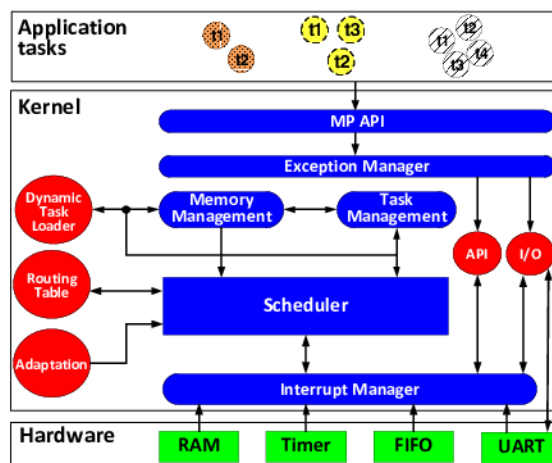
Sự chuyển dịch giữa các Task rất linh động, độ trễ thấp tạo độ tin cậy cao cho chương trình.

2.1.3. 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 (Even) như ngắt, 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 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.

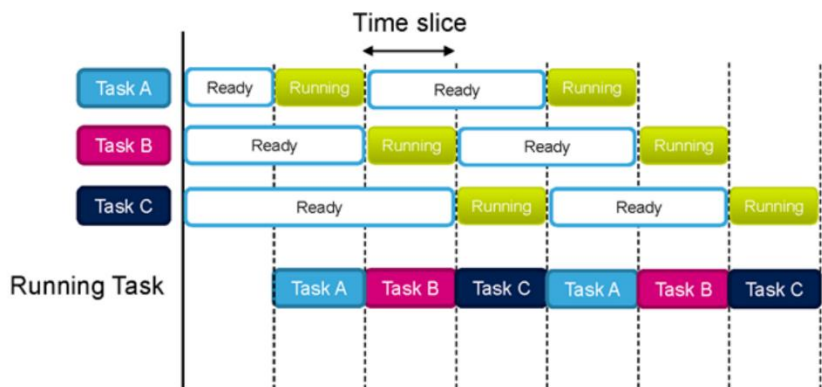
Scheduler – Lập lịch

Scheduler là một cơ chế trong kernel của RTOS, quản lý thời gian CPU và quyết định task nào sẽ chạy tại bất kỳ thời điểm nào.

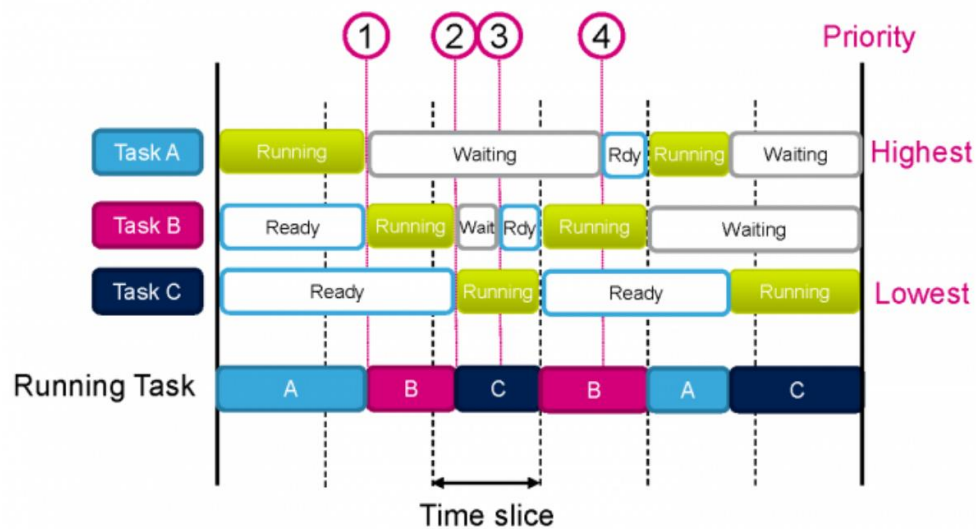
Có một số luật cho scheduling như:

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

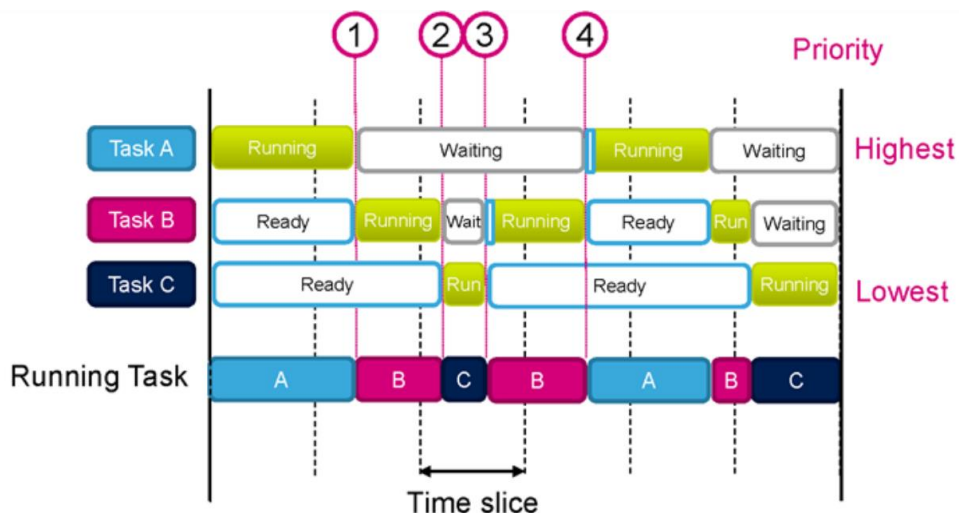
Round-robin: mỗi task được thực hiện trong thời gian định trước (time slice) và không có ưu tiên.



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

Để chương trình có thể hoạt động trơn chu thì 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 hàng 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

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).

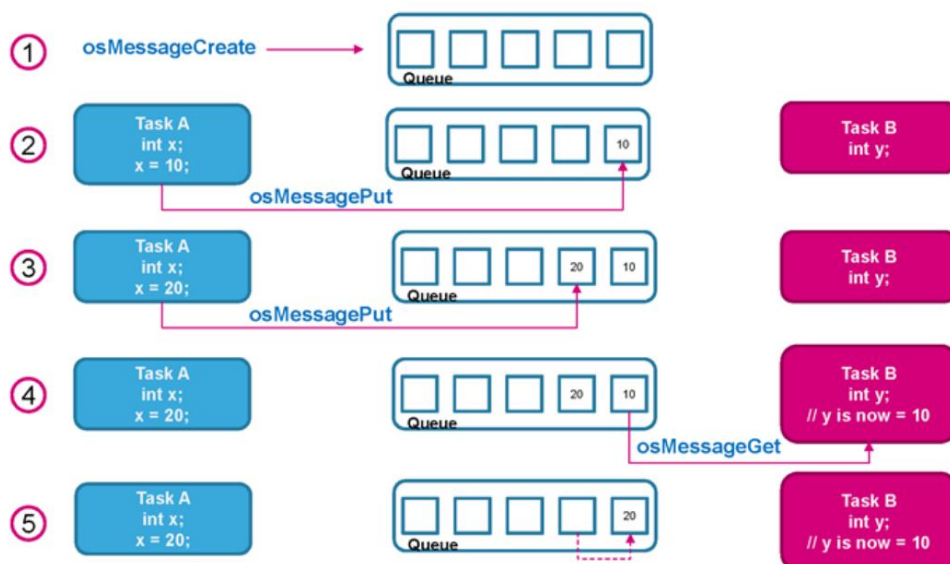
Một ứng dụng tiêu biểu là buffer cho Serial I/O, buffer cho lệnh được gửi tới task.

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 bị block thì task với mức ưu tiên cao nhất sẽ được unblock trước.

Task có thể đọc từ hàng 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.



Mail queue

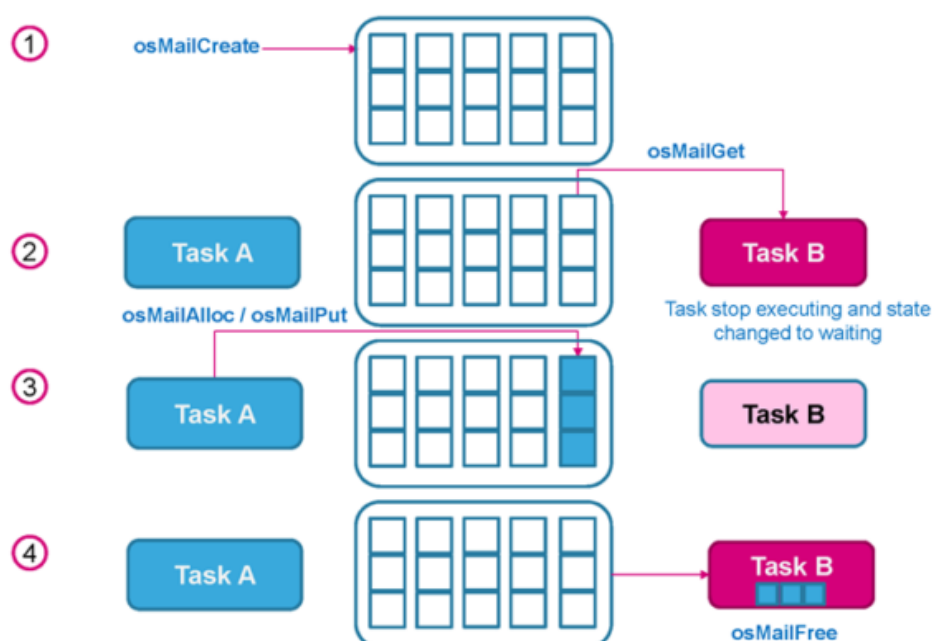
Mail queue truyền dữ liệu sẽ được truyền dưới dạng khối(memory block) thay vì dạng đơn. Mỗi memory block thì cần phải cấp phát trước khi đưa dữ liệu vào và giải phóng sau khi đưa dữ liệu ra.

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



Semaphore

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

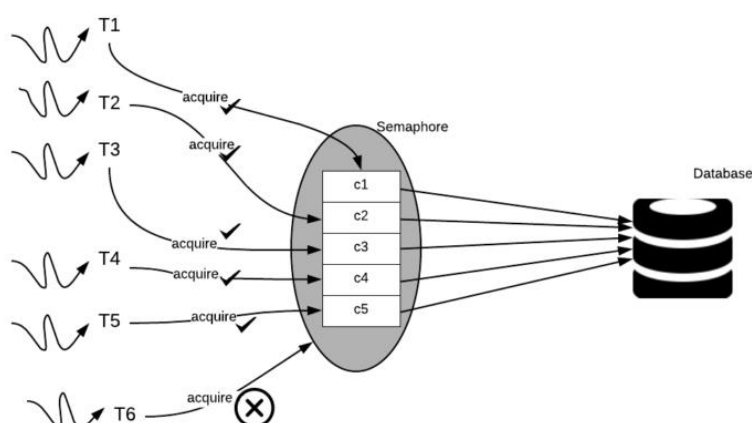
Semaphore là một mã thông báo giống như một mã thông báo cho phép một tác vụ thực hiện việc thực thi nếu tác vụ nhận được semaphore. Nếu không, tác vụ vẫn ở trạng thái khối và không thể thực thi trừ khi nó có được semaphore nhị phân

- 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



Counting 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

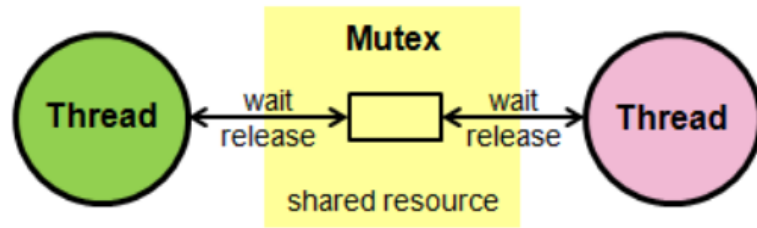
- Count value sẽ chỉ ra số resource sẵn có
- Để điều khiển và kiểm soát được resource của task dựa trên count value của semaphore (giá trị giảm), nếu count value giảm xuống bằng 0 nghĩa là không có resource nào free.
- Khi một task finish với resource thì nó sẽ give semaphore trở lại để tăng count value của semaphore.
- Trong trường hợp resource management thì count value sẽ bằng với giá trị max của count value khi semaphore được tạo.

Mutex

Mutex 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.

2.2. FreeRTOS

FreeRTOS là một hệ điều hành thời gian thực (Real-Time Operating System - RTOS) mã nguồn mở, nhỏ gọn và tối ưu, được thiết kế đặc biệt cho các hệ thống nhúng. FreeRTOS phổ biến nhờ tính linh hoạt, khả năng hỗ trợ nhiều kiến trúc vi điều khiển, và cộng đồng phát triển rộng lớn.

Đặc Điểm Nổi Bật Của FreeRTOS

- Mã nguồn FreeRTOS có kích thước nhỏ (~10 KB đến 15 KB), phù hợp cho các vi điều khiển có tài nguyên hạn chế (RAM/Flash thấp).
- FreeRTOS tương thích với hơn 40 vi điều khiển, như STM32, ESP32, PIC, AVR, ARM Cortex-M, RISC-V.
- FreeRTOS hỗ trợ việc thêm tính năng hoặc module tùy thuộc vào yêu cầu của dự án.
- Có tài liệu chi tiết, ví dụ mẫu, và cộng đồng người dùng rộng rãi.

-
-
- FreeRTOS cung cấp cơ chế quản lý đa nhiệm dựa trên preemptive hoặc cooperative scheduling.
 - Đáp ứng các yêu cầu thời gian thực nhờ cơ chế ưu tiên task và lập lịch chính xác.

Thư viện FreeRTOS.h

Thư viện FreeRTOS.h là thư viện chính của FreeRTOS (hệ điều hành thời gian thực phổ biến cho các vi điều khiển và các hệ thống nhúng). Nó chứa các định nghĩa và hàm cần thiết để khởi tạo và cấu hình RTOS trong chương trình của bạn.

Một số chức năng cơ bản trong FreeRTOS.h:

- Khởi tạo RTOS: Bạn cần bao gồm thư viện này trong mã nguồn để khởi tạo FreeRTOS và sử dụng các API của nó.
- Quản lý bộ nhớ: FreeRTOS cung cấp một số cơ chế để quản lý bộ nhớ động, ví dụ như `pvPortMalloc()` và `vPortFree()`, thay thế các hàm `malloc()` và `free()` tiêu chuẩn.
- Thiết lập các ngữ cảnh hệ thống: FreeRTOS cho phép thiết lập các nhiệm vụ (task), các hàng đợi (queue), các semaphore, và các mutex để đồng bộ hóa và giao tiếp giữa các tác vụ.

Thư viện task.h

Thư viện task.h là một phần trong FreeRTOS, cung cấp các API để tạo và quản lý các nhiệm vụ (tasks). Các nhiệm vụ này thực thi song song trong hệ thống và có thể chia sẻ tài nguyên, đồng bộ hóa với nhau, hoặc giao tiếp qua các cơ chế của FreeRTOS.

Một số chức năng cơ bản trong task.h:

- `xTaskCreate()`: Tạo một nhiệm vụ mới trong FreeRTOS.
- `vTaskDelete()`: Xóa một nhiệm vụ hiện tại hoặc đã được tạo.
- `vTaskDelay()`: Dừng một nhiệm vụ trong một khoảng thời gian xác định, cho phép các nhiệm vụ khác có cơ hội thực thi.
- `vTaskStartScheduler()`: Khởi động hệ thống scheduler của FreeRTOS để bắt đầu thực thi các nhiệm vụ.

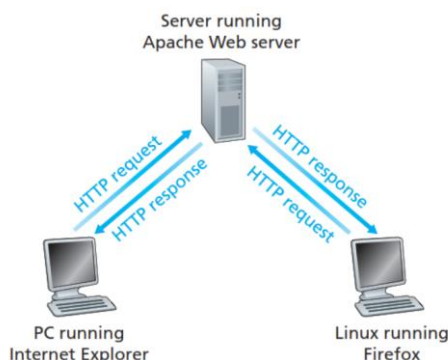
2.3. Phương thức truyền tải dữ liệu HTTP

HTTP (Hyper Text Transfer Protocol) là một giao thức nằm ở tầng ứng dụng (Application layer) của tập giao thức TCP/IP, sử dụng để truyền nhận dữ liệu giữa các hệ thống phân tán

thông qua internet, cụ thể giao thức hoạt động theo mô hình Client-Server bằng cách thực hiện các quá trình request-response giữa các hệ thống máy tính khác nhau. Giao thức HTTP quy định cấu trúc của các gói tin và cách thức truyền nhận dữ liệu giữa client và server thông qua môi trường internet. Với khả năng truyền dẫn siêu văn bản (text, hình ảnh, âm thanh, video,...), HTTP hiện là nền tảng truyền dẫn dữ liệu của ứng dụng duyệt web hiện nay và được ứng dụng rất nhiều trong các hệ thống Internet of Things.

2.3.1. Mô hình giao thức

Mỗi khi người dùng sử dụng trình duyệt và truy cập vào một website, một phiên làm việc HTTP (gọi là Session) sẽ được diễn ra với client là máy tính của người dùng và server là máy chủ của website. Mặc định HTTP sẽ được thực hiện thông qua port 80, đây là port chuẩn của giao thức được định nghĩa bởi tổ chức **IANA** quy định.



Ví dụ khi người dùng truy cập vào địa chỉ:

https://script.google.com/macros/s/AKfycbxiriZ4rLr0z96XZQyXxjqFrK_0tJl7QIBWGqUjXJ_t4y97mma6yGuXseywKxGkfAeL/exec

Quá trình của một phiên làm việc HTTP diễn ra như sau:

1. HTTP client thiết lập một kết nối TCP đến server. Nếu thiết lập thành công, client và server sẽ truyền nhận dữ liệu với nhau thông qua kết nối này, kết nối được thiết lập còn gọi là socket interface bao gồm các thông tin: địa chỉ IP, loại giao thức giao vận (chính là TCP), và port (mặc định là 80).

2. Sau khi kết nối thành công, client gửi một HTTP request đến server thông qua socket interface vừa được thiết lập. Trong gói tin request sẽ chứa đường dẫn yêu cầu (path name) là

/macros/s/AKfycbxiriZ4rLr0z96XZQyXxjqFrK_0tJl7QIBWGqUjXJ_t4y97mma6yGuXseywkxGkfAeL/exec

3. Server sẽ nhận và xử lý request từ client thông qua socket, sau đó đóng gói dữ liệu tương ứng và gửi một HTTP response về cho client. Dữ liệu trả về sẽ là một file HTML chứa các loại dữ liệu khác nhau như văn bản, hình ảnh,...
4. Server đóng kết nối TCP.
5. Client nhận được dữ liệu phản hồi từ server và đóng kết nối TCP.

2.3.2. Format gói tin

Trong suốt phiên làm việc HTTP như ở trên, có hai loại gói tin chính được trao đổi là gói tin yêu cầu gửi từ client đến server gọi là “Request message” và gói tin phản hồi gửi từ server đến client gọi là “Response message”. Hai loại gói tin này được giao thức HTTP định nghĩa rất cụ thể:

HTTP request message

Gồm 3 phần chính là: Request line, Header và Body. Dữ liệu được chia thành các dòng và định dạng kết thúc dòng là <CR><LF> (tương ứng với các ký tự 0x0A, 0x0D trong bảng mã ASCII). Riêng dòng cuối cùng của gói tin sẽ được báo hiệu bằng 2 lần định dạng kết thúc: <CR><LF><CR><LF>.



Request line là dòng đầu tiên của gói HTTP Request, bao gồm 3 trường: phương thức (method), đường dẫn (path – có nhiều bài viết gọi là URL hoặc URI cho trường này) và phiên bản giao thức (HTTP version).

- **Phương thức (method)** có thể là: GET, POST, HEAD, PUT và DELETE. Hai phương thức phổ biến nhất là GET và POST, trong ví dụ trên là phương thức GET thường dùng để yêu cầu tài nguyên cung cấp trong trường URL.
- **Đường dẫn (path)** dùng để định danh nguồn tài nguyên mà client yêu cầu, bắt buộc phải có ít nhất là dấu “/”.
- **Phiên bản giao thức (HTTP version)**: là phiên bản HTTP client đang sử dụng (thường là HTTP/1.0 hoặc HTTP/1.1)

Tiếp theo là các dòng Header, các dòng này là không bắt buộc, viết ở định dạng “Name:Value” cho phép client gửi thêm các thông tin bổ sung về thông điệp HTTP request và thông tin về chính client. Một số header thông dụng như:

- **Accept**: loại nội dung có thể nhận được từ thông điệp response. Ví dụ: text/plain, text/html
- **Accept-Encoding**: các kiểu nén được chấp nhận. Ví dụ: gzip, deflate, xz, exi...
- **Connection**: tùy chọn điều khiển cho kết nối hiện thời. Ví dụ: Keep-Alive, Close...
- **Cookie**: thông tin HTTP Cookie từ server

Cuối cùng là phần Body, là dữ liệu gửi từ client đến server trong gói tin HTTP request. Đa số các gói tin gửi theo phương thức GET sẽ có Body trống, các phương thức như POST hay PUT thường dùng để gửi dữ liệu nên sẽ có bao gồm dữ liệu trong trường Body.

Sử dụng phương thức GET gửi dữ liệu đến server sử dụng chuỗi truy vấn (query string): khi gửi một gói tin HTTP request theo phương thức GET, trong trường hợp client muốn gửi dữ liệu lên server để thực hiện lưu trữ hoặc thực hiện một yêu cầu cụ thể nào đó, thông thường client sẽ không đóng gói dữ liệu trong Body mà sẽ được gửi thẳng ở dòng Header line phía sau trường path theo định dạng các tham số dạng “**key=value**” như sau:

GET /path?key1=value1&key2=value2&...&keyn=valuen HTTP/1.1

Dữ liệu gửi lên như vậy gọi là một chuỗi truy vấn (query string), được bắt đầu bằng dấu “?” và kế tiếp là các cặp “**key=value**”. Trong đó giá trị của các key thường sẽ được cung cấp bởi API từ phía server và client sẽ cung cấp dữ liệu của key thông qua value tương ứng. Ví dụ API cập nhật dữ liệu lên ThingSpeak có dạng:

GET api.thingspeak.com/update?api_key=xxxxxxxxxx&field1=value HTTP/1.1

HTTP response message

Định dạng gói tin HTTP response gồm 3 phần chính là: Status line, Header, Body.



Dòng Status line gồm 3 trường là phiên bản giao thức (HTTP version), mã trạng thái (Status code) và mô tả trạng thái (Status text):

- **Phiên bản giao thức (HTTP version):** phiên bản của giao thức HTTP mà server hỗ trợ, thường là HTTP/1.0 hoặc HTTP/1.1
- **Mã trạng thái (Status code):** mô tả trạng thái kết nối dưới dạng số, mỗi trạng thái sẽ được biểu thị bởi một số nguyên. Ví dụ: 200, 404, 302,...
- **Mô tả trạng thái (Status text):** mô tả trạng thái kết nối dưới dạng văn bản một cách ngắn gọn, giúp người dùng dễ hiểu hơn so với mã trạng thái. Ví dụ: 200 OK, 404 Not Found, 403 Forbidden,...

Các dòng Header line của gói tin response có chức năng tương tự như gói tin request, giúp server có thể truyền thêm các thông tin bổ sung đến client dưới dạng các cặp “Name:Value”.

Phần Body là nơi đóng gói dữ liệu để trả về cho client, thông thường trong duyệt web thì dữ liệu trả về sẽ ở dưới dạng một trang HTML để trình duyệt có thể thông dịch được và hiển thị ra cho người dùng.

CHƯƠNG III: THIẾT KẾ PHẦN CỨNG

3.1. Phân tích thiết kế

- Vi xử lý: ESP32



Thông số:

- Loại: Wifi + Bluetooth Module.
- Mô hình: ESP32 38 chân.
- Điện áp nguồn (USB): 5V DC.
- Đầu vào/Đầu ra điện áp: 3.3V DC.
- Công suất tiêu thụ: 5 μ A trong hệ thống treo chế độ.
- Hiệu suất: Lên đến 600 DMIPS.
- Tần số: lên đến 240MHz.
- Wifi: 802.11 B/g/n/E/I (802.11N @ 2.4 GHz lên đến 150 Mbit/S).
- Bluetooth: 4.2 BR/EDR BLE 2 chế độ điều khiển.
- Bộ nhớ: 448 Kbyte ROM, 520 Kbyte SRAM, 6 Kbyte SRAM trên RTC và QSPI Hỗ trợ đèn flash / SRAM chip.
- Chip USB-Serial: CP2102.
- Ăng ten: PCB.
- GPIO kỹ thuật số: 24 chân (một số chân chỉ làm đầu vào).
- Kỹ thuật số Analog: 12bit SAR loại ADC, hỗ trợ các phép đo trên lên đến 18 kênh, một số chân hỗ trợ một bộ khuếch đại với lập trình tăng.
- Bảo mật: IEEE 802.11, bao gồm cả WFA, WPA/WPA2 và WAPI.

- Phần cứng tăng tốc mật mã học: AES, SHA-2, RSA, hình elip mật mã Đường Cong (ECC), số ngẫu nhiên Máy phát điện (RNG).

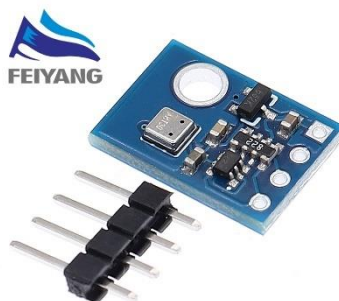
- *Mô-đun hiển thị OLED 128x64.*



Thông số:

- Hoàn toàn tương thích với Arduino, 51 Series, MSP430 Series, STM32 / 2, CSR IC, v.v
- Tiêu thụ điện năng cực thấp - toàn màn hình sáng 0,08W.
- Độ sáng và độ tương phản siêu cao có thể điều chỉnh được.
- Loại giao diện là IIC.
- Điện áp “ 3V ~ 5V DC.
- Nhiệt độ làm việc ' -30 ° ~ 70 °
- Độ phân giải cao ' 128 * 64.
- Trình điều khiển IC ' SSD1306.

- *Mô Đun Cảm Biến Nhiệt Độ Và Độ Ẩm aht30*



Thông số:

- Kích thước mô-đun: 16 * 11 mm
- Loại giao diện: I2C
- Điện áp làm việc: 1,8-6,0 V
- Kích thước giao diện: 4 * 2,54mm cao độ
- Độ ảm chính xác: $\pm 2\%$ điển hình
- Độ phân giải độ ảm: 0,024%
- Độ chính xác nhiệt độ: điển hình $\pm 0,3^\circ\text{C}$
- Độ phân giải nhiệt độ: Tiêu chuẩn 0,01 $^\circ\text{C}$
- Nhiệt độ làm việc: $-40^\circ\text{C} - 85^\circ\text{C}$

- Cảm biến hồng ngoại IR vượt vật cản.



Thông số:

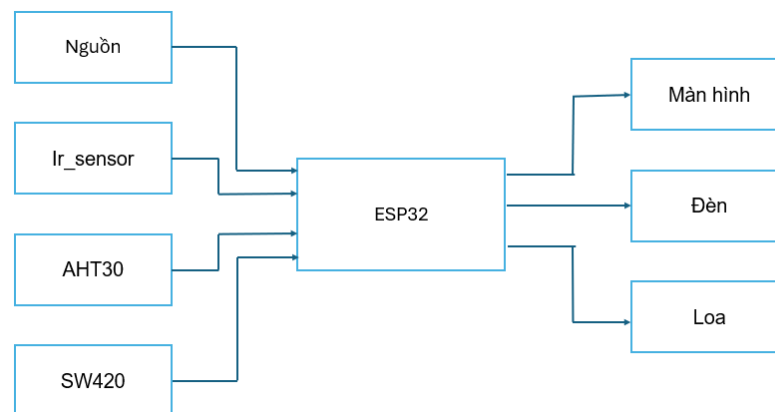
- Điện áp sử dụng : 3.3~5V DC.
- Nhận biết vật cản bằng ánh sáng hồng ngoại.
- Đầu ra: Kỹ thuật số TTL
- Tích hợp biến trở chỉnh khoảng cách nhận biết vật cản.
- Kích thước: 32 x 14mm

- Cảm biến rung SW420



- Điện áp: 3.3-5V
- Dòng tiêu thụ: 15mA
- Biến trở điều chỉnh ngưỡng so sánh
- Kích thước : 32x14MM
- Chân sử dụng: VCC, GND, DO
- Vcc: 3.3-5V
- GND: 0V
- DO: Tín hiệu ra 0 và 1
- DO đưa vào IO của Vi Điều Khiển hoặc Điều Khiển Relay qua Transistor.

3.2. Sơ đồ khối chi tiết



- Vi điều khiển chính là ESP32 điều khiển các ngoại vi, nhận tín hiệu từ các cảm biến.
- Khối nguồn dùng để cấp nguồn cho toàn bộ hệ thống 5V.
- IR sensor để nhận biết vật cản.
- AHT30 đọc giá trị nhiệt độ và độ ẩm.
- SW420 nhận diện rung.
- Màn hình hiển thị những thông số đo được và các thông tin cần thiết.
- Đèn cảnh báo khi có nhiệt độ $> 60^{\circ}\text{C}$.

CHƯƠNG IV: THIẾT KẾ PHẦN MỀM

4.1. Máy trạng thái.

4.1.1. Khái niệm máy trạng thái.

Máy trạng thái (State Machine) là một mô hình toán học được sử dụng để biểu diễn hành vi của một hệ thống. Nó mô tả hệ thống dưới dạng một tập hợp các trạng thái và các chuyển đổi (transitions) giữa các trạng thái đó dựa trên các sự kiện hoặc điều kiện cụ thể.

Thành phần của một máy trạng thái:

Tập hợp các trạng thái: Các trạng thái là những tình huống mà hệ thống có thể tồn tại.

Ví dụ: *Đang chờ, Đang hoạt động, Dừng hoạt động.*

Trạng thái ban đầu: Trạng thái mà hệ thống bắt đầu khi khởi động.

Tập hợp các sự kiện hoặc đầu vào: Đây là các điều kiện hoặc tín hiệu kích hoạt các chuyển đổi giữa các trạng thái.

Chuyển đổi (Transitions): Xác định sự thay đổi từ trạng thái này sang trạng thái khác khi có một sự kiện xảy ra. Có thể kèm theo hành động (Actions) khi thực hiện chuyển đổi.

Hành động (Actions): Là các hoạt động mà hệ thống thực hiện khi chuyển đổi trạng thái hoặc khi ở trong một trạng thái nhất định.

Phân loại máy trạng thái:

Máy trạng thái hữu hạn (Finite State Machine - FSM):

- Có số lượng trạng thái hữu hạn.
- Thường dùng trong hệ thống nhúng, xử lý tín hiệu, và thiết kế phần mềm.

Máy trạng thái Mealy và Moore:

- Máy trạng thái **Mealy**: Đầu ra phụ thuộc vào cả trạng thái hiện tại và đầu vào.
- Máy trạng thái **Moore**: Đầu ra chỉ phụ thuộc vào trạng thái hiện tại.

4.1.2. Phân thích máy trạng thái đề tài.

- Task 1: Hiển thị dữ liệu lên màn hình OLED (Mức ưu tiên 1)

Nhiệm vụ này đảm bảo việc hiển thị thông tin quan trọng (nhiệt độ, độ ẩm, trạng thái cảm biến rung, tín hiệu cảnh báo) lên màn hình OLED để người dùng có thể dễ dàng theo dõi tình trạng hệ thống.

- Task 2: Đọc dữ liệu từ cảm biến hồng ngoại (IR Sensor) (Mức ưu tiên 2)

Nhiệm vụ này thu thập tín hiệu từ cảm biến hồng ngoại để phát hiện chuyển động hoặc sự thay đổi ánh sáng trong phạm vi giám sát.

- Task 3: Đọc dữ liệu từ cảm biến nhiệt độ và độ ẩm AHT30 (Mức ưu tiên 2)

Cảm biến AHT30 được sử dụng để đo nhiệt độ và độ ẩm môi trường. Dữ liệu này sẽ được sử dụng cho các nhiệm vụ cảnh báo.

- Task 4: Đọc dữ liệu từ cảm biến rung (Mức ưu tiên 2)

Cảm biến rung giám sát chấn động hoặc rung lắc trong hệ thống, cung cấp thông tin cảnh báo khi có sự cố.

- Task 5: Kích hoạt loa cảnh báo khi phát hiện rung (Mức ưu tiên 3)

Loa được kích hoạt khi cảm biến rung phát hiện chấn động, nhằm cảnh báo người sử dụng về tình trạng bất thường.

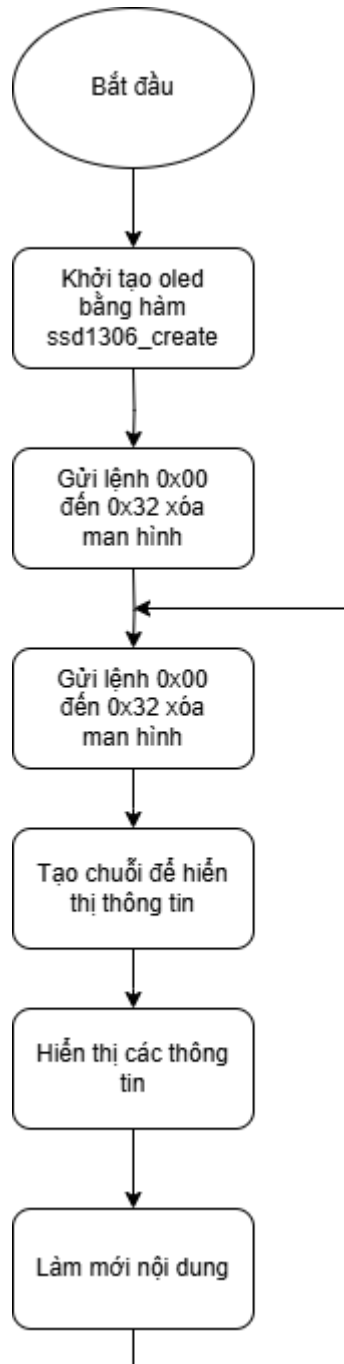
- Task 6: Bật đèn đỏ cảnh báo khi nhiệt độ $> 60^{\circ}\text{C}$ (Mức ưu tiên 5)

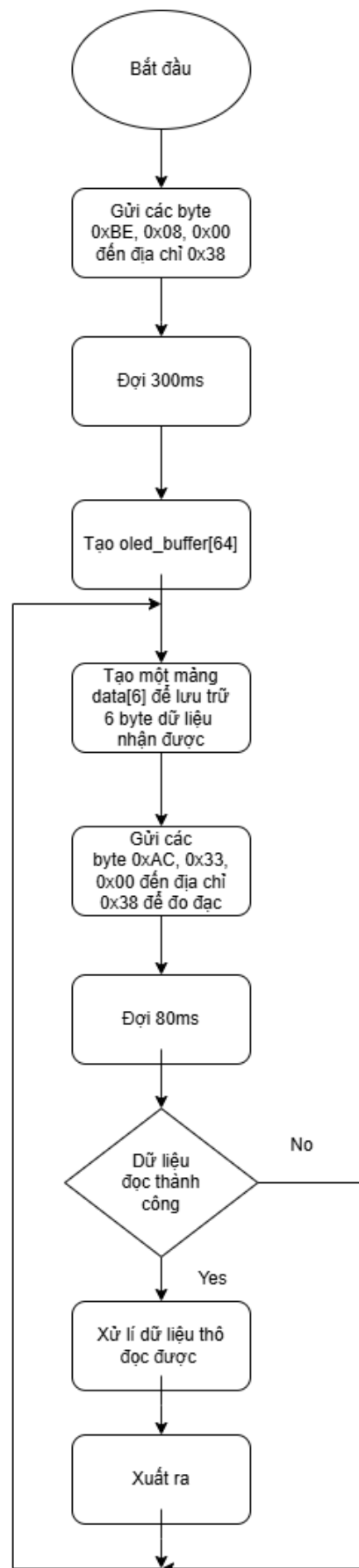
Đèn đỏ được sử dụng để cảnh báo khi nhiệt độ môi trường vượt ngưỡng 60°C , nhằm đảm bảo an toàn cho hệ thống và người sử dụng.

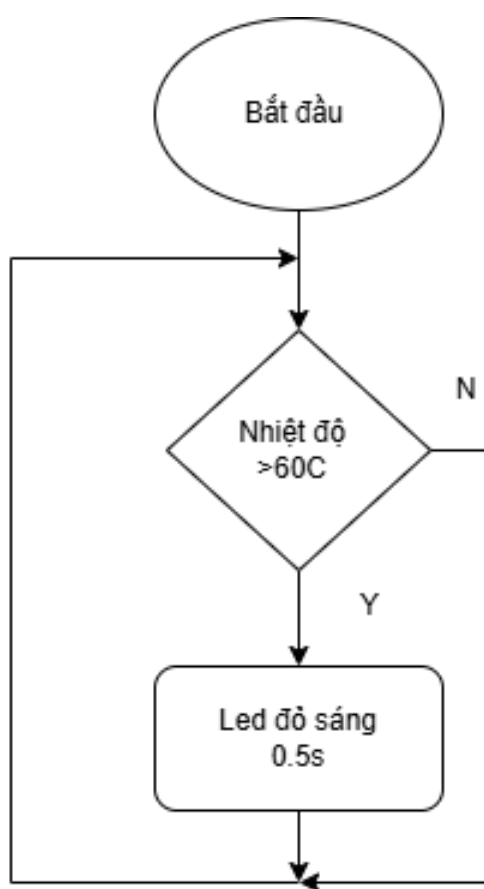
- Task 7: Post data gửi lên gg sheet sau mấy task đọc cảm biến (Mức ưu tiên 3)

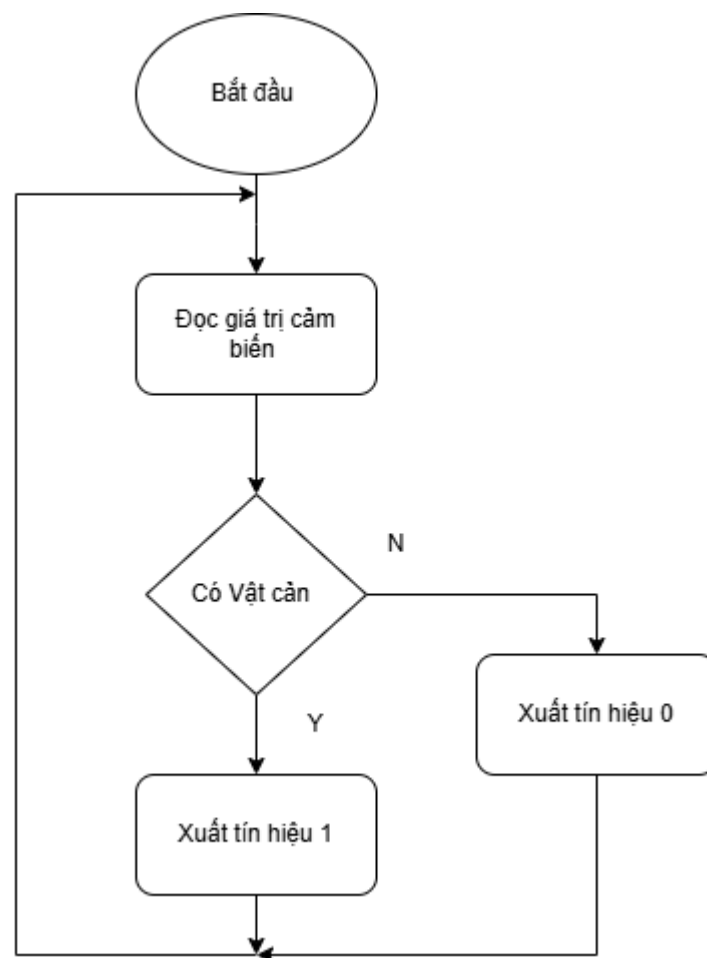
4.2. Lưu đồ giải thuật

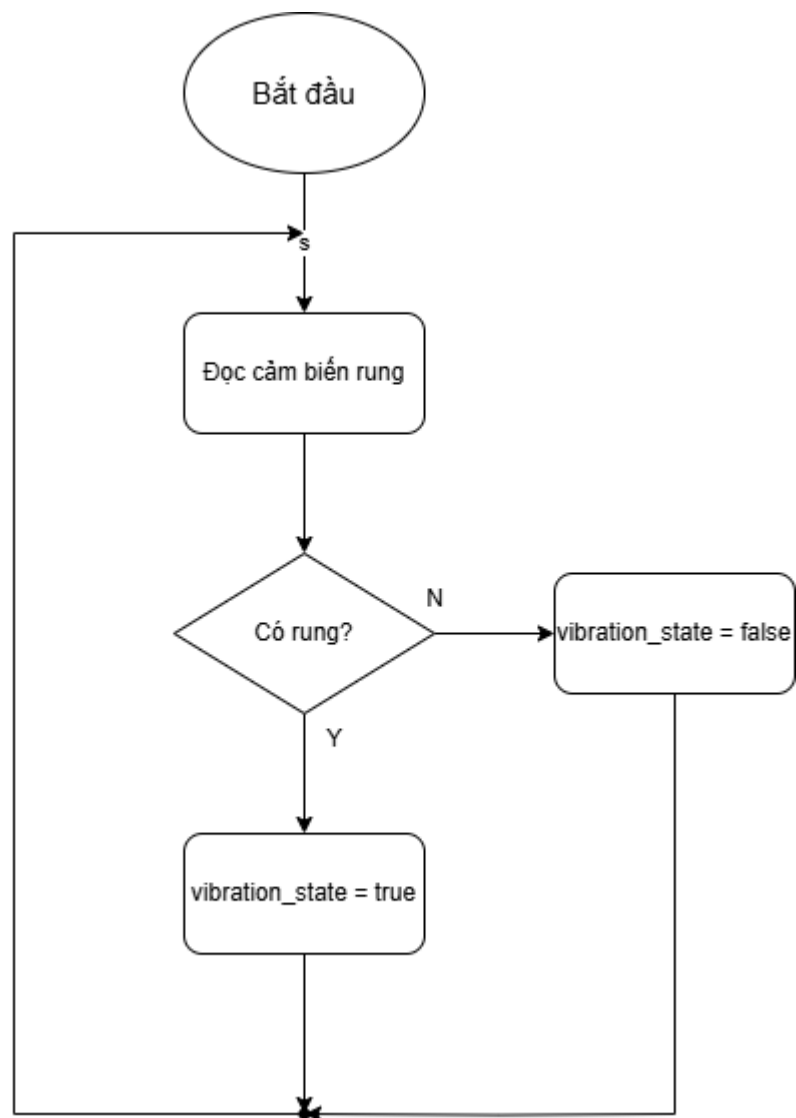
Task oled

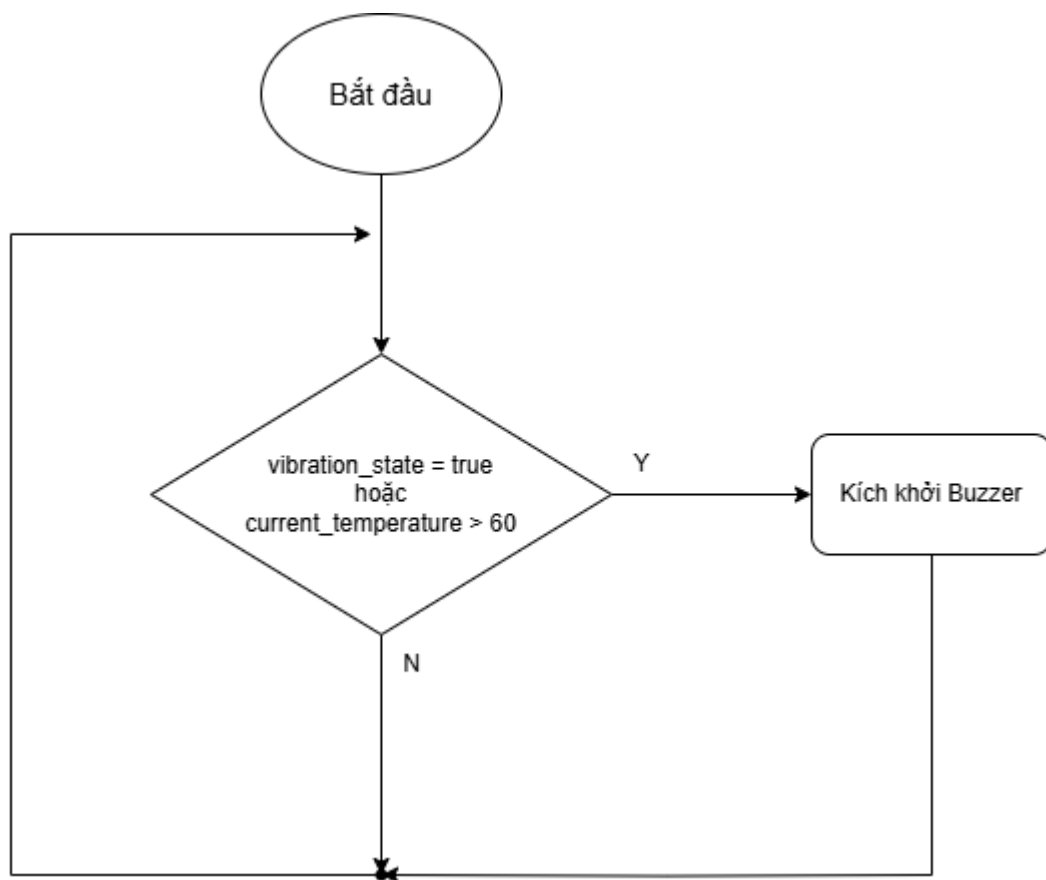




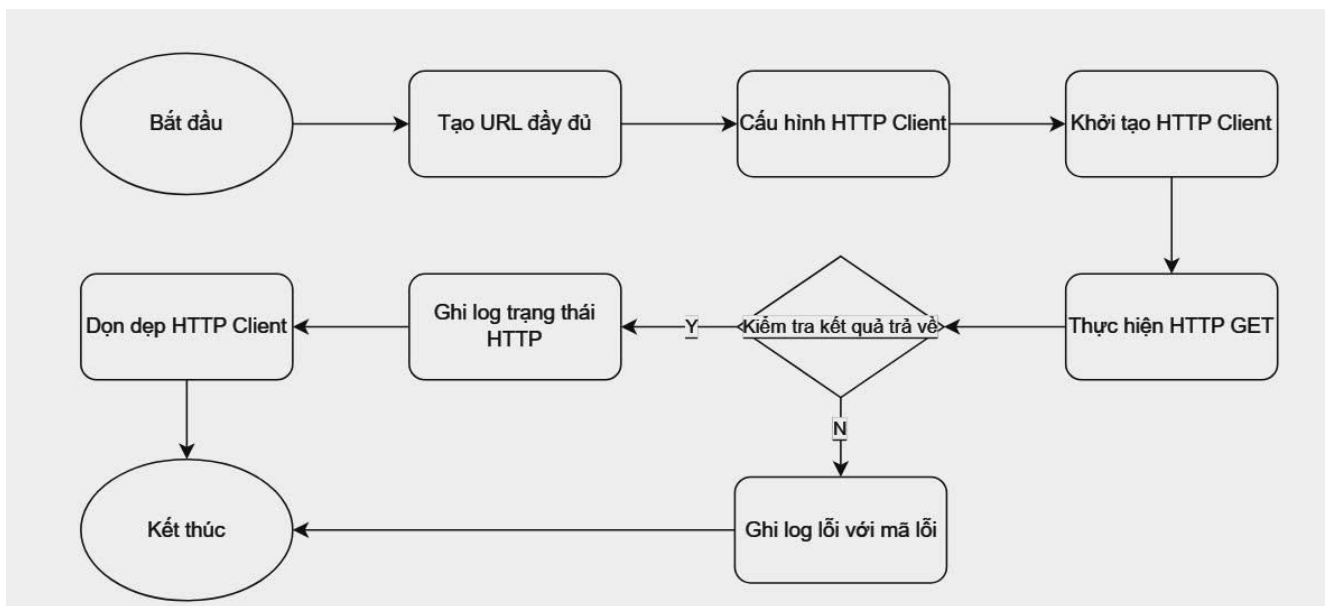






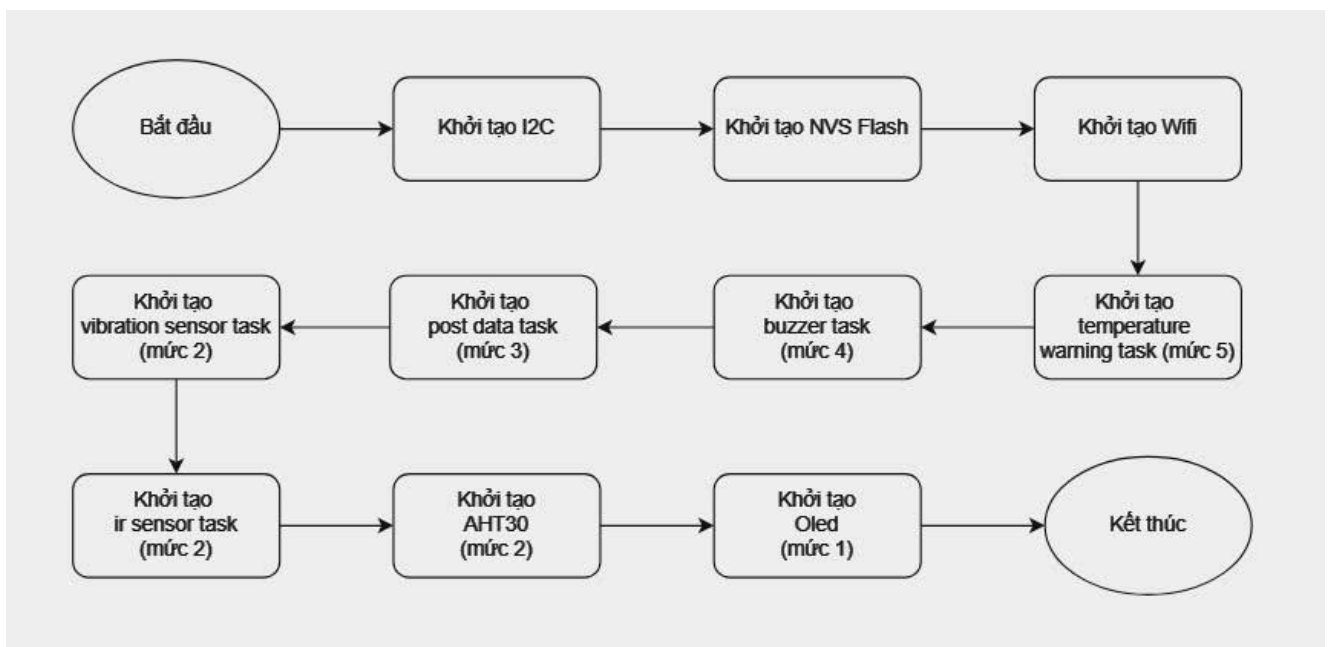


Task post data



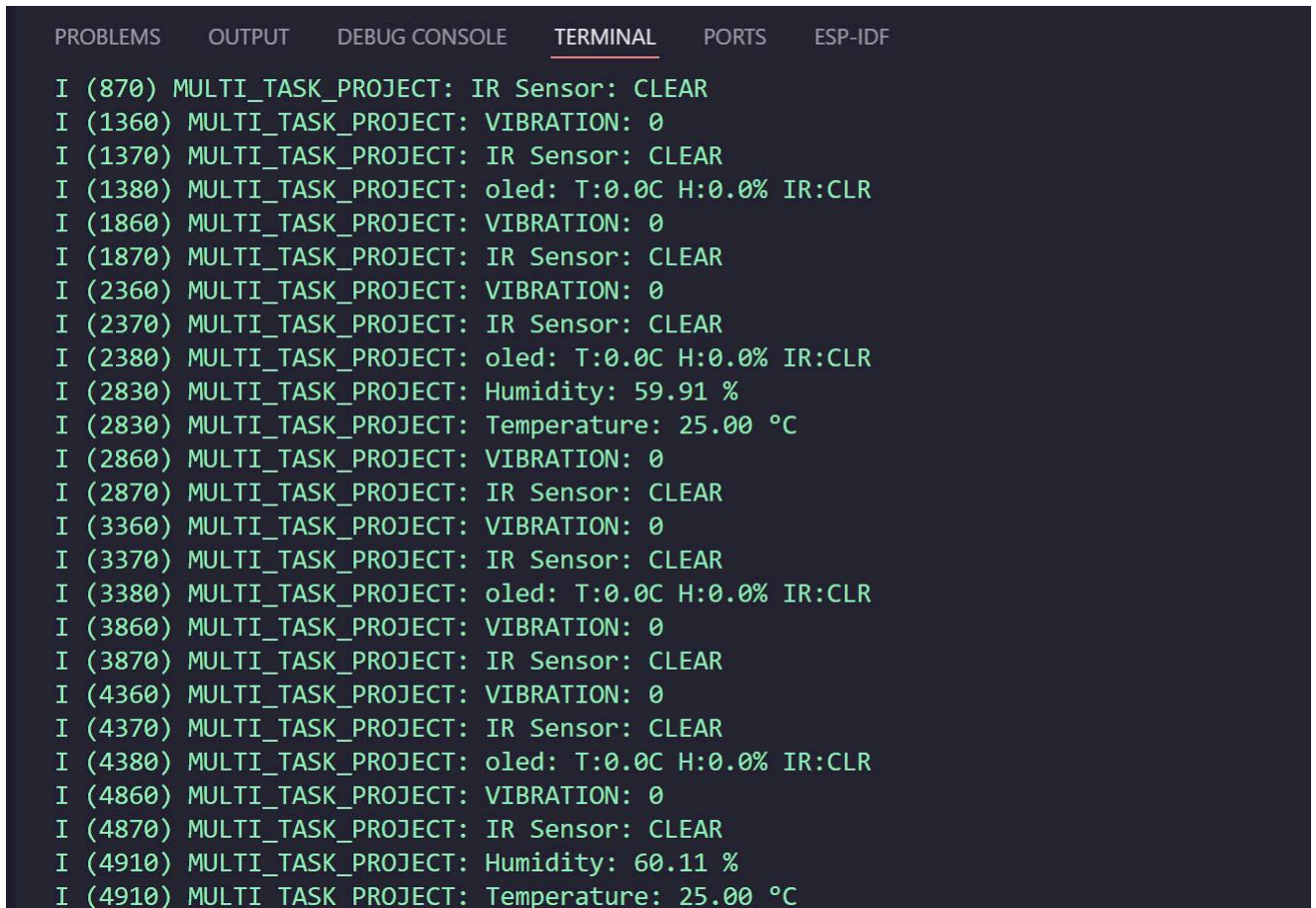
Sơ đồ giải thuật chương trình chính :

Các task có mức ưu tiên cao hơn khi xảy ra sẽ được ưu tiên thực hiện trước



CHƯƠNG V: KẾT QUẢ THỰC HIỆN

Kết quả xuất ra monitor của vs code



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  ESP-IDF

I (870) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (1360) MULTI_TASK_PROJECT: VIBRATION: 0
I (1370) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (1380) MULTI_TASK_PROJECT: oled: T:0.0C H:0.0% IR:CLR
I (1860) MULTI_TASK_PROJECT: VIBRATION: 0
I (1870) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (2360) MULTI_TASK_PROJECT: VIBRATION: 0
I (2370) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (2380) MULTI_TASK_PROJECT: oled: T:0.0C H:0.0% IR:CLR
I (2830) MULTI_TASK_PROJECT: Humidity: 59.91 %
I (2830) MULTI_TASK_PROJECT: Temperature: 25.00 °C
I (2860) MULTI_TASK_PROJECT: VIBRATION: 0
I (2870) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (3360) MULTI_TASK_PROJECT: VIBRATION: 0
I (3370) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (3380) MULTI_TASK_PROJECT: oled: T:0.0C H:0.0% IR:CLR
I (3860) MULTI_TASK_PROJECT: VIBRATION: 0
I (3870) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (4360) MULTI_TASK_PROJECT: VIBRATION: 0
I (4370) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (4380) MULTI_TASK_PROJECT: oled: T:0.0C H:0.0% IR:CLR
I (4860) MULTI_TASK_PROJECT: VIBRATION: 0
I (4870) MULTI_TASK_PROJECT: IR Sensor: CLEAR
I (4910) MULTI_TASK_PROJECT: Humidity: 60.11 %
I (4910) MULTI TASK PROJECT: Temperature: 25.00 °C
```

Figure 1 Kết quả Serial Monitor VS Code

Kết quả xuất ra oled



Figure 2 Kết quả xuất ra OLED

Kết quả post data

	A	B	C	D	E	F
573	1/12/2024	11:27:20	30.00	40.00	1	1
574	1/12/2024	11:27:16	30.00	40.00	1	1
575	1/12/2024	11:27:10	30.00	40.00	1	1
576	1/12/2024	11:27:03	30.00	40.00	1	1
577	1/12/2024	11:26:48	30.00	40.00	1	1
578	1/12/2024	11:26:40	30.00	40.00	1	1
579	1/12/2024	11:26:31	30.00	40.00	1	1
580	1/12/2024	11:26:26	30.00	40.00	1	1
581	1/12/2024	11:26:21	30.00	40.00	1	1
582	1/12/2024	11:26:14	30.00	40.00	1	1
583	1/12/2024	11:26:07	30.00	40.00	1	1
584	1/12/2024	11:26:01	30.00	40.00	1	1
585	1/12/2024	11:25:54	30.00	40.00	1	1
586	1/12/2024	11:25:49	30.00	40.00	1	1
587	1/12/2024	11:25:44	30.00	40.00	1	1
588	1/12/2024	11:25:39	30.00	40.00	1	1
589	1/12/2024	11:25:33	30.00	40.00	1	1
590	1/12/2024	11:25:28	30.00	40.00	1	1
591	1/12/2024	11:25:22	30.00	40.00	1	1

☰
NOW ▾
Dữ Liệu ▾

Figure 3 Kết quả post data Google Sheets

CHƯƠNG VI: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Tổng kết những kết quả đạt được: Trong nghiên cứu này, hệ thống giám sát máy công nghiệp sử dụng ESP32 đã hoàn thành các nhiệm vụ cơ bản như đọc dữ liệu từ các cảm biến nhiệt độ, độ ẩm, vật cản và rung. Các cảm biến đã được tích hợp hiệu quả để thu thập dữ liệu môi trường trong thời gian thực. Hệ thống xử lý dữ liệu và cung cấp thông tin qua các thiết bị đầu ra như màn hình hiển thị, loa và đèn để cảnh báo hoặc thông báo tình trạng của máy móc.

Thông qua việc sử dụng FreeRTOS, hệ thống đã đảm bảo được việc xử lý đồng thời các tác vụ cảm biến và điều khiển các thiết bị đầu ra một cách mượt mà và chính xác.

Hướng phát triển tương lai:

Ứng dụng FreeRTOS trong các lĩnh vực khác: FreeRTOS có thể được mở rộng và áp dụng vào các hệ thống nhúng khác có yêu cầu về độ tin cậy cao, chẳng hạn như các ứng dụng trong y tế, tự động hóa công nghiệp hoặc các hệ thống giám sát từ xa. Việc sử dụng FreeRTOS giúp giảm tải cho bộ vi xử lý và cải thiện hiệu suất của hệ thống.

Nâng cấp phần cứng hoặc tích hợp thêm các tính năng:

- **Nâng cấp cảm biến:** Có thể tích hợp thêm các cảm biến khác như cảm biến khí gas, cảm biến ánh sáng, hoặc cảm biến chuyển động để mở rộng khả năng giám sát.
- **Tối ưu hóa phần mềm:** Phát triển các thuật toán xử lý dữ liệu cảm biến nâng cao để cải thiện độ chính xác và giảm độ trễ trong việc phản hồi của hệ thống.

Thông qua các nâng cấp này, hệ thống không chỉ có thể phát hiện và cảnh báo các vấn đề trong máy công nghiệp, mà còn mở ra khả năng tích hợp với các hệ thống giám sát và quản lý khác trong các ứng dụng công nghiệp hiện đại.

CHƯƠNG VII: TÀI LIỆU THAM KHẢO

- [1] H. T. -. B. Q. Bảo, Lập trình hệ thống nhúng, Nhà xuất bản đại học quốc gia Tp Hồ Chí Minh, 2019.
- [2] Digikey, "Grove - Vibration Sensor(SW-420)," [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/Grove_Vibration_Sensor_SW-420_Web.pdf.
- [3] Eleparts, " Datasheet AHT30," [Online]. Available: https://eleparts.co.kr/data/goods_attach/202306/good-pdf-12751003-1.pdf.
- [4] Espressif, "ESP32 C3," [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.
- [5] C. Diy, "HW201 Infrared (IR) Sensor Module," [Online]. Available: <https://www.circuits-diy.com/hw201-infrared-ir-sensor-module/>.
- [6] Datasheethub, "SSD1306 128×64 Mono 0.96 Inch I2C OLED Display," [Online]. Available: <https://www.datasheethub.com/wp-content/uploads/2022/08/SSD1306.pdf>.