**6. Phân biệt hàm delay, vTaskDelay, vTaskDelayUntil và chạy thử nghiệm 2 task**

**delay:** là loại delay được sử dụng phổ biến, không thuộc hệ điều hành nào cả. Tùy vào thư viện thì nó sẽ có API khác nhau ( HAL\_Delay() trong STM32 thư viện HAL, hay delay() trong Arduino). Hàm delay trong Arduino có tác dụng block task và trả CPU cho task khác.

**Cú pháp: delay(time\_ms);**

**vTaskDelay:** Hàm delay thuộc hệ điều hành RTOS, khi thực hiện hàm này sẽ đưa task vào trạng thái block để có thể thực hiện task các.

**Cú pháp: vTaskDelay(ticks);**

**vTaskDelay(pdMS\_TO\_TICKS(time\_ms));**

**vTaskDelayUntil:** Giống như vTaskDelay. Nhưng hoạt động dựa trên timeline Tick của hệ điều hành. (mỗi Tick = 10ms).

**Cú pháp:**

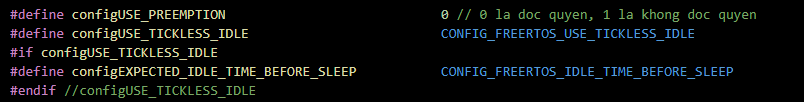
**TickType\_t xLastWakeTime;**

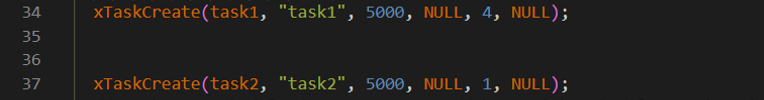
**const TickType\_t xPeriod = time;**

**xLastWakeTime = xTaskGetTickCount();**

**vTaskDelayUntil( &xLastWakeTime, xPeriod);**

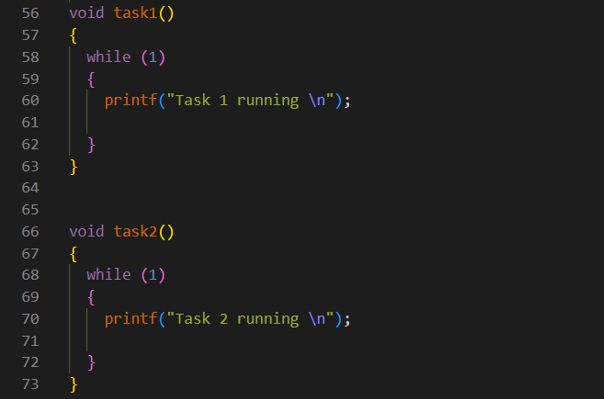
**Setup:** Sử dụng chế độ độc quyền với độ ưu tiên của task 1 lớn hơn task 2 (4 > 1)



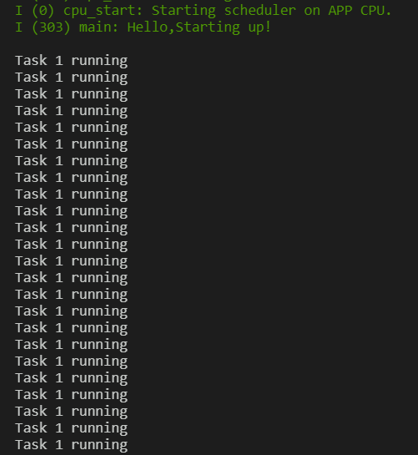


**Ví dụ 1: tạo 2 task sử dụng hàm delay và vTaskDelay**

TH1: Không có hàm delay



Do task 1 có độ ưu tiên cao hơn nên nó được thực thi trước



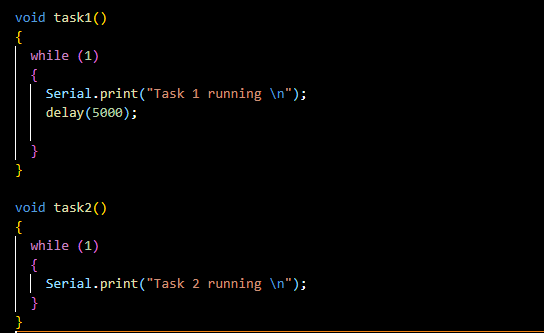
Vì task 1 không có hàm nào block nó nên nó sẽ chạy vĩnh viễn

Sơ đồ hoạt động:

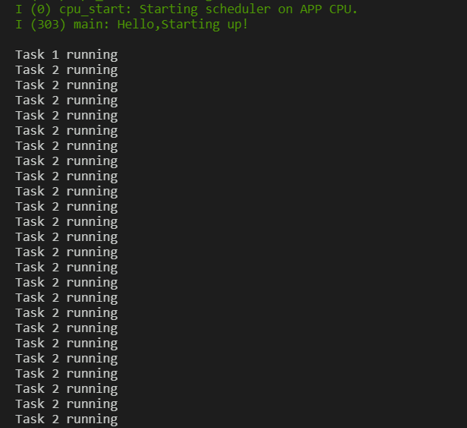
Task 1

Task 2

TH2: Sử dụng hàm delay của thư viện Arduino



Do task 1 có độ ưu tiên cao hơn nên nó được thực thi trước



Ở trường hợp này ta có sử dụng hàm delay ở task 1 nên sau lệnh print ở task 1 nó trả quyền CPU về cho task 2 vì delay ở thư viện arduino có chức năng block task và task 2 sẽ chạy vĩnh viễn.

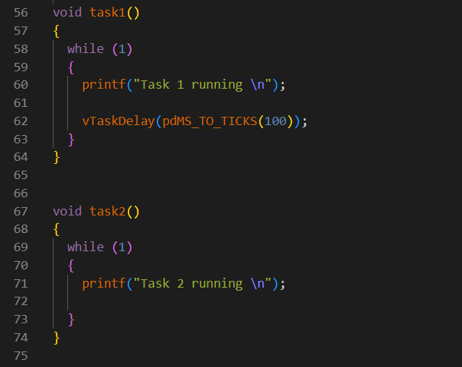
Sơ đồ hoạt động:

Task 1

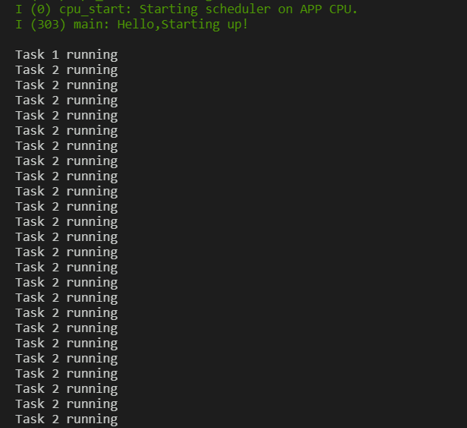
Task 2

5 s

TH3: Sử dụng hàm vTaskDelay



Do task 1 có độ ưu tiên cao hơn nên nó được thực thi trước



Ở trường hợp này ta có sử dụng hàm vTaskDelay ở task 1 thì khi task 1 print xong gặp hàm vTaskDelay nó sẽ bị block ngay lập tức và trả quyền CPU về cho task 2 chạy vĩnh viễn.

Sơ đồ hoạt động:

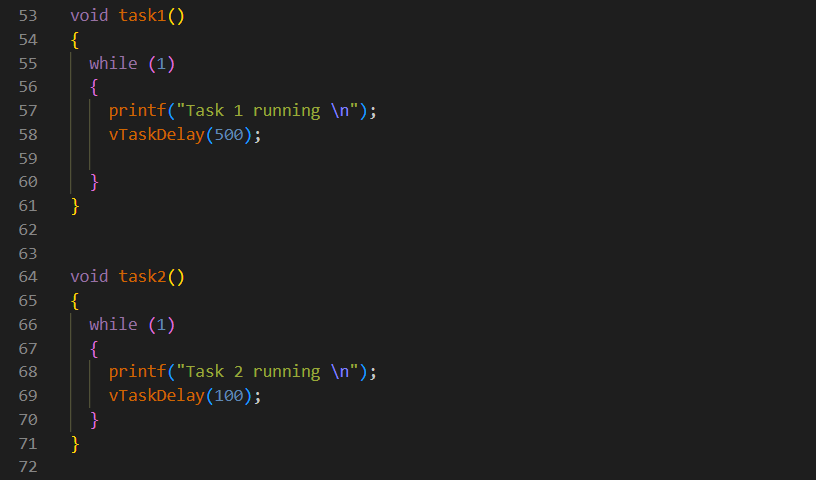
Task 1

Task 2

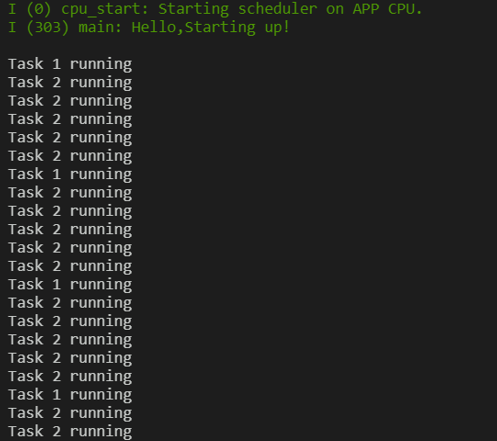
0.1s

**Ví dụ 2: tạo 2 task sử dụng hàm vTaskDelay và vTaskDelayUntil**

TH1: Sử dụng hàm vTaskDelay



Do task 1 có độ ưu tiên cao hơn nên nó được thực thi trước



Khi cả hai task 1 task 2 thực hiện xong lệnh print cả hai gặp hàm vTaskDekay sẽ bị block, task 1 bị block 5s, task 2 bị block 1s nên task 2 sẽ print 5 lần, task 1 print 1 lần sau 5s

Sơ đồ hoạt động:

Task 1

Task 2

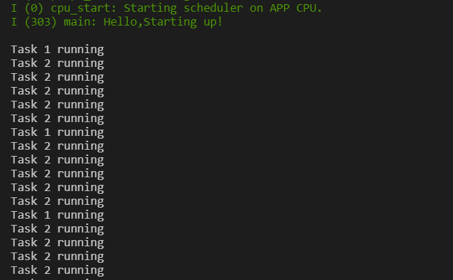
0s 5s

TH2: Sử dụng hàm vTaskDelayUntil



Do task 1 có độ ưu tiên cao hơn nên nó được thực thi trước

Ở hàm vTaskDelayUntil thì nó sẽ lặp lại theo khoảng thời gian dựa trên biến xFrequency



Khi cả hai task 1 task 2 thực hiện xong lệnh print cả hai gặp hàm vTaskDekay sẽ bị block, task 1 bị block 5s do biến xFrequency = 5000 ms, task 2 bị block 1s do biến xFrequency = 1000 ms, nên task 2 sẽ print 5 lần, task 1 print 1 lần sau 5s

Sơ đồ hoạt động:

Task 1

Task 2

0s 5s