Vấn đề đồng bộ / trao đổi dữ liệu giữa các task

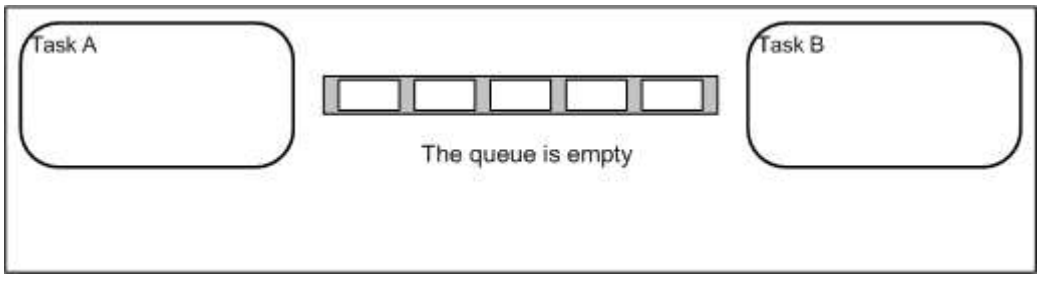
Mỗi task thực hiện tác vụ riêng của mình, chúng có không gian bộ nhớ riêng và hoàn toàn độc lập với nhau.

- Vậy làm thế nào để phối hợp chúng trong một chương trình lớn hơn, nếu xét mỗi task như là một module?
- Nếu task này muốn giao tiếp/trao đổi thông tin với task kia thì làm thế nào?
- Nếu 2 task cùng mức ưu tiên, cùng chạy song song và cùng muốn truy cập đến một tài nguyên chung thì làm sao để tránh xung đột?

Tất cả chỉ gói gọn trong hàng đợi (queue), semaphore và mutex

Hàng đợi (queue)

Hoạt động theo cơ chế FIFO (First In - First Out) dùng trong FreeRTOS để trao đổi thông điệp (message) giữa các task với nhau.



Ví dụ

Sử dụng lại ví dụ `esp-rtos-basic-task`, nhưng 2 task sửa lại theo mục đích sau:

- `task_led`: nháy led, đếm biến `counter`, đếm xong thì gửi giá trị biến đếm này qua queue cho `task_printf` in ra giá trị.
- `task_printf`: chỉ chờ có thông điệp mới trên queue thì lấy thông điệp này ra khỏi queue và in giá trị này!

Mã nguồn chương trình:

```
git clone https://github.com/esp8266vn/esp-rtos-basic-queue-sem-mutex.git
cd esp-rtos-basic-queue-sem-mutex
```

Để sử dụng được queue phải `#include "freertos/queue.h"`

Tạo queue

Cú pháp:

```
 xQueueHandle xQueueCreate( UBaseType_t uxQueueLength,
                            UBaseType_t uxItemSize );
```

- `uxQueueLength`: kích thước/số phần tử tối đa trên queue
- `uxItemSize`: kích thước của mỗi phần tử, tùy thuộc vào *kiểu* dữ liệu của phần tử trên queue.

Trường hợp này, biến `counter` là `uint32_t`. Giả sử cần tối đa 10 phần tử trên queue. Khai báo và tạo như sau:

```
xQueueHandle xCounterQueue;
// ...
xCounterQueue = xQueueCreate( 10, sizeof(uint32_t));
```

Gửi thông điệp lên queue

Theo cú pháp:

```
BaseType_t xQueueSend(
                          QueueHandle_t xQueue,
                          const void * pvItemToQueue,
                          TickType_t xTicksToWait
                      );
```

- `xQueue`: queue handle (`xCounterQueue`)
- `pvItemToQueue`: trỏ đến phần tử/thông điệp cần gửi lên queue (`&counter`)
- `xTicksToWait`: thời gian chờ tối đa (*timeout* - tính bằng `tick`) trong trường hợp queue bị đầy không gửi thêm vào được.

`task_led` sẽ làm nhiệm vụ này:

```
void task_led(void *pvParameters)
{
    uint32_t counter = 0;
    for(;;){
        vTaskDelay(100);
        GPIO_OUTPUT_SET(LED_GPIO, led_state);
        led_state ^=1;
        counter++;
        if( xQueueSend( xCounterQueue,( void * ) &counter, 10 ) != pdPASS )
        {
            // Failed to send msg to queue, TODO handle
        }
    }
}
```

Nhận thông điệp từ queue

Theo cú pháp:

```
BaseType_t xQueueReceive(
                              QueueHandle_t xQueue,
                              void *pvBuffer,
                              TickType_t xTicksToWait
                            );
```

- `xQueue`: queue handle (`xCounterQueue`)
- `pvBuffer`: trỏ đến nơi sẽ chứa thông điệp lấy từ queue xuống.
- `xTicksToWait`: timeout trong trường hợp không có phần từ nào trên queue (đang trống)

`task_printf` sẽ thực hiện nhiệm vụ này:

```
void task_printf(void *pvParameters)
{
    uint32_t receiveCounter;

    printf("task_printf started\n");

    for(;;){
        if( xQueueReceive( xCounterQueue, &( receiveCounter ), 100 ) )
        {
           printf("task_printf, received counter = %u\n", receiveCounter);
        }
    }
}
```

Chạy chương trình

```
make clean
make
make flash
```

Semaphore

Mutex