Mã nguồn ví dụ

```
git clone https://github.com/esp8266vn/esp-rtos-basic-task.git
```

Makefile

Cấu trúc của Makefile trong ví dụ này cũng tương tự Makefile cho dự án phức tạp sử dụng NONOS-SDK. Tuy nhiên, để sử dụng cho RTOS SDK thì một số biến trong Makefile cần thay đổi như sau:

```
# Đường dẫn tới RTOS-SDK
SDK_BASE    ?= C:/Espressif/ESP8266_RTOS_SDK
...
# Thư viện sử dụng
SDK_LIBS = gcc hal phy pp net80211 wpa crypto main freertos lwip minic smartconfig
...
# Thư mục đưa vào include
SDK_INC = extra_include include include/espressif include/json include/udhcp include/lwip inclu
...
# Cờ khi biên dịch C
CFLAGS = -g -save-temps -std=gnu90 -Os -Wpointer-arith -Wundef -Werror \
         -Wl,-EL -fno-inline-functions -nostdlib -mlongcalls -mtext-section-literals \
         -mno-serialize-volatile -D__ets__ -DICACHE_FLASH -DBUID_TIME=\"$(DATETIME)\"
...
# Comment-out dòng này:
# ifeq ("$(USE_OPENSDK)","yes")
# CFLAGS        += -DUSE_OPENSDK
# else
# CFLAGS        += -D_STDINT_H
# endif
```

Cấu trúc chương trình `user_main.c`

Chương trình có nhiệm vụ tạo ra 2 task riêng biệt, một dùng để nháy LED có chu kỳ 200 `ticks`, task còn lại để in thông tin ra UART0 với chu kỳ 1000 `ticks`.

Muốn vậy, trước hết phải tạo ra 2 hàm con, tuân theo tiền khai báo kiểu `TaskFunction_t` có dạng `void vTaskCode( void * pvParameters )` cho task LED và UART:

```
void task_led(void *pvParameters)
{
    for(;;){
        vTaskDelay(100);
        GPIO_OUTPUT_SET(LED_GPIO, led_state);
        led_state ^=1;
    }
}

void task_printf(void *pvParameters)
{
    for(;;){
        printf("task_printf\n");
        vTaskDelay(500);
    }
}
```

Lưu ý

- Bên trong hàm con phải thực hiện vòng lặp vô tận (*infinite loop*), không được *return*.
- Ngoài ra, task có thể tự *hủy* (delete) chính nó khi cần (bằng hàm `vTaskDelete( TaskHandle_t xTask )` - xem ví dụ bên dưới)

Tick

- `Tick` là hành động khi timer ngắt định kỳ dùng trong nhân FreeRTOS để MCU thực hiện chuyển *ngữ cảnh* (*context*) khi chuyển qua lại giữa các task với nhau, khái niệm *thực hiện tác vụ song song, đồng thời, cùng lúc* chỉ mang tính tương đối, vì RTOS thực hiện điều này 1 cách tuần tự.
- Giá trị của `tick` không phải lúc nào cũng là 1ms, tùy thuộc vào cấu hình khi *port* FreeRTOS lên từng MCU khác nhau (trong trường hợp, `configTICK_RATE_HZ` của RTOS-SDK có giá trị là 100). Có thể kiểm tra chính xác chu kỳ ms của `tick` bằng macro `portTICK_RATE_MS` –> Để delay chính xác `t(ms)` thì tham số truyền cho `vTaskDelay` là `t/portTICK_RATE_MS`

Trong hàm `user_init()` của ESP8266, sau khi khởi tạo các giá trị cần thiết cho UART và chân GPIO Ouput để nháy LED, sử dụng hàm `xTaskCreate` trong FreeRTOS để tạo task thực thi 2 hàm con này, cú pháp `xTaskCreate`:

```
BaseType_t xTaskCreate(    TaskFunction_t pvTaskCode,
                           const char * const pcName,
                           unsigned short usStackDepth,
                           void *pvParameters,
                           UBaseType_t uxPriority,
                           TaskHandle_t *pxCreatedTask
                       );
```

Trong đó:

- `pvTaskCode` : trỏ đến hàm con cần thực hiện khi tạo task
- `pcName` : chuỗi mô tả tên của task này
- `usStackDepth` : độ lớn của con trỏ ngăn xếp, chọn sao cho lớn hơn độ lớn của con trỏ ngăn xếp khi thực hiện hàm con, ví dụ như khi hàm con gọi càng nhiều hàm khác bên trong lồng nhau, khi đó độ lớn này càng tăng.
- `pvParameters` : trỏ đến tham số cần truyền vào hàm con khi task khởi tạo.
- `uxPriority` : mức độ ưu tiên của task.
- `pxCreatedTask` : trỏ đến biến kiểu `TaskHandle_t` , biến sẽ được gán sau khi gọi `xTaskCreate` thành công, xem như *ID* để phân biệt các task với nhau, và được sử dụng cho nhiều mục đích, ví dụ như xóa task (dùng hàm `vTaskDelete( TaskHandle_t xTask )`) Sử dụng `xTaskCreate` để tạo task LED và UART như sau:

```
xTaskCreate(task_led, "task_led", 256, NULL, 2, NULL);
xTaskCreate(task_printf, "task_printf", 256, NULL, 2, NULL);
```

Biên dịch và chạy chương trình

```
make clean
make
make flash
```

Nâng cao một chút

- In vài thông tin cơ bản về `portTICK_RATE_MS` và `configMAX_PRIORITIES`
- Ví dụ về `vTaskDelete()` cho `task_printf`

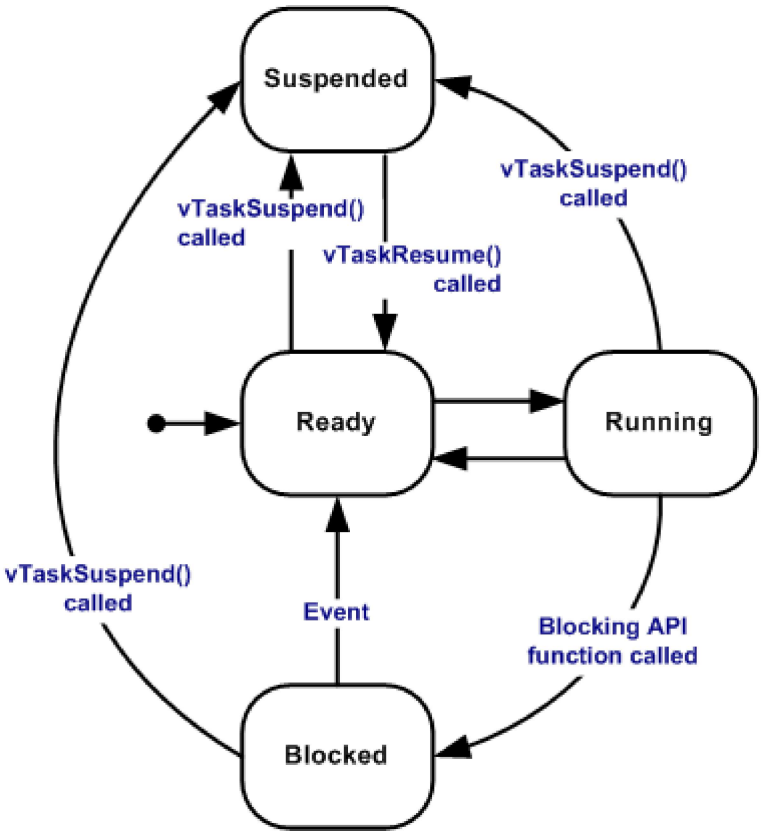
```
git checkout task_delete
make clean
make
make flash
```

Giới thiệu về tác vụ (task) trong FreeRTOS

Các trạng thái (states) của task

Có 4 trạng thái: Running, Ready, Blocked và Suspended

- Running: là trạng thái task đang được MCU thực thi thực sự, vì thế trong một thời điểm chỉ có duy nhất một task ở trạng thái running.
- Ready: là trạng thái task đã sẵn sàng để thực thi (không bị *blocked* hoặc *suspended*) nhưng đang không được MCU thực thi bởi vì MCU đang thực thi một task khác.
- Blocked: task đang bị *blocked* nếu nó đang đợi sự kiện (*event*) bên ngoài hoặc sự kiện thời gian. Ví dụ: khi task gọi hàm `vTaskDelay()` thì nó sẽ bị *blocked* cho đến khi hết thời gian delay (sự kiện thời gian). Hoặc task có thể bị *blocked* để đợi một hàng đợi (queue), semaphore, notification… nào đó. Thông thường, task bị *blocked* trong khoảng thời gian quá hạn `timeout` cho trước, vì thế task sẽ luôn được *unblocked* (nếu có sự kiện bên ngoài) hoặc `timeout` nếu hết thời gian chờ.
- Suspended: Cũng giống như trạng thái *blocked* nhưng không có thời gian `timeout`, vì vậy chỉ có thể `enter` hoặc `exit` khỏi trạng thái *suspended* bởi hàm gọi từ bên ngoài tương ứng là `vTaskSuspend()` và `xTaskResume()`



Các mức ưu tiên (priorities) của task

Mỗi task được tạo ra với mức ưu tiên được gán từ 0 đến giá trị `(configMAX_PRIORITIES - 1)`, với `configMAX_PRIORITIES` là giá trị được định nghĩa trong `FreeRTOSConfig.h` (với bản RTOS-SDK v1.4 đang sử dụng, `configMAX_PRIORITIES` là 15)

Task đang ở trạng thái *ready* có ưu tiên cao hơn sẽ được chọn để thực thi (chuyển sang *running*) trong mỗi lần `tick`

Nếu các task có cùng mức ưu tiên? Trong trường hợp RTOS sẽ chia đều ra xử lý (do giá trị `configUSE_TIME_SLICING` đã được định nghĩa là 1 trong bản RTOS-SDK này)