

**ĐẠI HỌC QUỐC GIA  
ĐẠI HỌC BÁCH KHOA TP HỒ CHÍ MINH**



**– Embedded System–**

**LAB 3**

**Nhóm 2 – L02**

<b>Họ và tên</b>	<b>MSSV</b>
Trần Nguyễn Minh Duy	1910095
Đặng Trung Kiên	1911437
Nguyễn Hải Long	1911517
Nguyễn Nhật Trường	1912344

*Thành phố Hồ Chí Minh – 2022*

## MỤC LỤC

<b>ĐÓNG GÓP CỦA CÁC THÀNH VIÊN .....</b>	<b>2</b>
<b>BÁO CÁO.....</b>	<b>3</b>
<b>1. Prioritized Pre-emptive Scheduling with Time Slicing.....</b>	<b>3</b>
1.1. Điều chỉnh trong menuconfig.....	3
1.2. Điều chỉnh trong file FreeRTOSConfig.h .....	3
1.3. Hàm in thông tin của Task ứng với ID .....	3
1.4. Hàm dành cho task liên tục (có Priority bằng nhau) .....	3
1.5. Hàm dành cho task có Priority cao hơn.....	3
1.6. Hàm main .....	4
1.7. Kết quả hiện thực và biểu đồ thời gian.....	5
<b>2. Prioritized Pre-emptive Scheduling (without Time Slicing) .....</b>	<b>5</b>
2.1. Điều chỉnh trong menuconfig .....	5
2.2. Điều chỉnh trong file FreeRTOSConfig.h .....	5
2.3. Phân hiện thực Code .....	5
2.4. Kết quả hiện thực và biểu đồ thời gian.....	5
<b>3. Co-operative Scheduling .....</b>	<b>6</b>
3.1. Điều chỉnh trong menuconfig.....	6
3.2. Điều chỉnh trong file FreeRTOSConfig.h .....	6
3.3. Hàm in thông tin của Task ứng với ID .....	6
3.4. Hàm task .....	7
3.5. Hàm main .....	8
3.6. Kết quả hiện thực và biểu đồ thời gian.....	8
<b>LINK GITHUB.....</b>	<b>10</b>

### **ĐÓNG GÓP CỦA CÁC THÀNH VIÊN**

<b>Họ và tên</b>	<b>MSSV</b>	<b>Đóng góp</b>
Trần Nguyễn Minh Duy	1910095	25%
Đặng Trung Kiên	1911437	25%
Nguyễn Hải Long	1911517	25%
Nguyễn Nhật Trường	1912344	25%

## BÁO CÁO

### 1. Prioritized Pre-emptive Scheduling with Time Slicing

#### 1.1. Điều chỉnh trong menuconfig

- Trong menuconfig, ta cần điều chỉnh để mạch ESP32 chỉ chạy trên 1 core.

#### 1.2. Điều chỉnh trong file FreeRTOSConfig.h

- Ta đặt configUSE\_TIME\_SLICING = 1 để bật chế độ Time Slicing.
- Ta đặt configUSE\_PREEMPTION = 1 để bật chế độ Prioritized Pre-emptive.

```
#define configUSE_TIME_SLICING      1
#define configUSE_PREEMPTION        1
```

#### 1.3. Hàm in thông tin của Task ứng với ID

- Ta in ra thời điểm tương ứng với Tick hiện tại (xTaskGetTickCount), và ID.

```
void print_task_information(uint8_t id)
{
    printf("Tick %04lu ID %lu\n", xTaskGetTickCount(), id);
}
```

#### 1.4. Hàm dành cho task liên tục (có Priority bằng nhau)

- Trong vòng while(1), ta gọi hàm print\_task\_information(id) (với id tương ứng).

```
void continuous_processing_task(void *id)
{
    while (1)
    {
        print_task_information((uint32_t) id);
    }

    vTaskDelete(NULL);
}
```

#### 1.5. Hàm dành cho task có Priority cao hơn

- Trong vòng while(1), ta gọi hàm print\_task\_information để in thông tin của task với ID = 3.
- Sau đó, ta gọi hàm vTaskDelay(30) cho task này vào trạng Block mỗi 30 tick.

```
void higher_priority_task(void *id)
{
    while (1)
    {
        printf("Event task %lu was preempted\n", (uint32_t) id);
        print_task_information(3);
        printf("Event task %lu was blocked\n", (uint32_t) id);

        vTaskDelay(30);
    }

    vTaskDelete(NULL);
}
```

#### 1.6. Hàm main

- Sau đó, ta dùng hàm xTaskCreate() để tạo 2 task liên tục (có priority bằng nhau, và bằng 1), với ID lần lượt bằng 1 và 2.
- Tiếp theo, ta tạo 1 task có priority cao hơn = 2 (ID = 3).

```
void app_main()
{
    vTaskPrioritySet(NULL, 10);

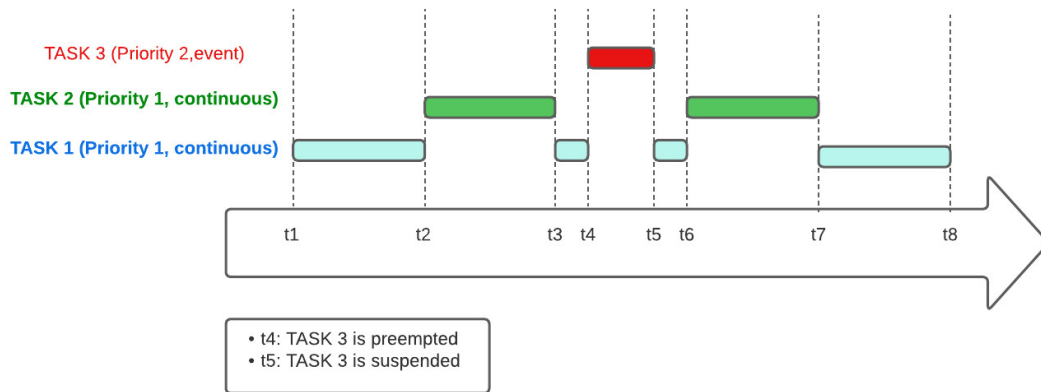
    init_gpio_and_interrupt();

    xTaskCreate(continuous_processing_task, "Continous task #1", 2048,
        (void*) 1u, 1, &contTask01_handle);
    xTaskCreate(continuous_processing_task, "Continous task #2", 2048,
        (void*) 2u, 1, &contTask02_handle);
    xTaskCreate(higher_priority_task, "Event task #3", 2048, (void*) 3, 2, &eventTask03_handle);

    vTaskDelete(NULL);
}
```

### 1.7. Kết quả hiện thực và biểu đồ thời gian

- Kết quả hiện thực: trong file lab3/ex1/output.txt.
- Biểu đồ thời gian của kết quả (theo file output):



## 2. Prioritized Pre-emptive Scheduling (without Time Slicing)

### 2.1. Điều chỉnh trong menuconfig

- Trong menuconfig, ta cần điều chỉnh để mạch ESP32 chỉ chạy trên 1 core.

### 2.2. Điều chỉnh trong file FreeRTOSConfig.h

- Ta đặt `configUSE_TIME_SLICING = 0` để tắt chế độ Time Slicing
- Ta đặt `configUSE_PREEMPTION = 1` để bật chế độ Prioritized Pre-emptive

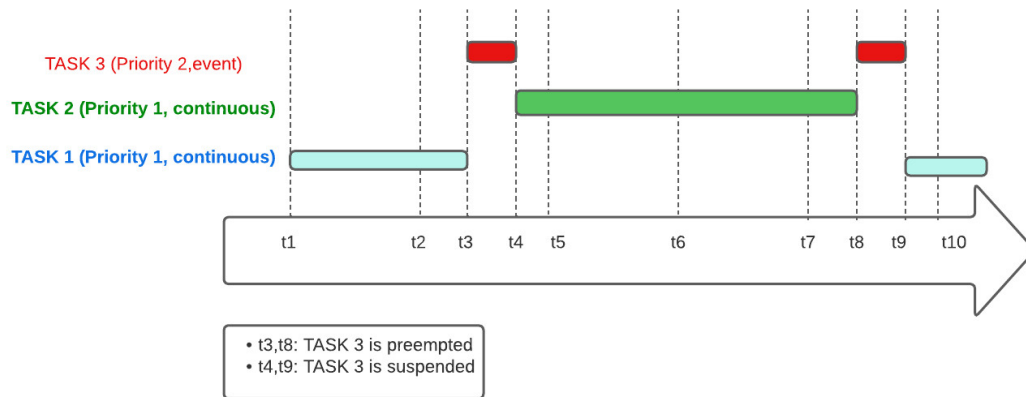
<code>#define configUSE_TIME_SLICING</code>	<code>0</code>
<code>#define configUSE_PREEMPTION</code>	<code>1</code>

### 2.3. Phần hiện thực Code

- Tương tự như code của bài 1 (phần 2.2, 2.3, 2.4, 2.5, 2.6).

### 2.4. Kết quả hiện thực và biểu đồ thời gian

- Kết quả hiện thực: trong file lab3/ex2/output.txt.
- Biểu đồ thời gian của kết quả (theo file output):



### 3. Co-operative Scheduling

#### 3.1. Điều chỉnh trong menuconfig

- Trong menuconfig, ta cần điều chỉnh để mạch ESP32 chỉ chạy trên 1 core.

#### 3.2. Điều chỉnh trong file FreeRTOSConfig.h

- Ta đặt `configUSE_TIME_SLICING = 0` để tắt chế độ Time Slicing
- Ta đặt `configUSE_PREEMPTION = 0` để tắt chế độ Prioritized Pre-emptive

```

#define configUSE_TIME_SLICING    0
#define configUSE_PREEMPTION      0
  
```

#### 3.3. Hàm in thông tin của Task ứng với ID

- Ta in
  - o ID của task tương ứng,
  - o biến đếm của vòng for cho task có Priority bé nhất (priority = 1, id = 1).
  - o biến đếm của vòng for cho task có Priority lớn hơn (priority = 2, id = 2).
  - o biến đếm của vòng for cho task có Priority lớn nhất (priority = 3, id = 3).

```

void print_task_information(uint8_t id)
  
```

```

{
    printf("id %u - %5d %5d %5d\n", id, time_count[0], time_count[1], time_count[2]);
}

```

### 3.4. Hàm task

- Đầu tiên, ta cho hàm chạy vòng lặp for, với số lần lặp tỉ lệ nghịch với priority của task. Cụ thể là:
  - o 30000 lần cho task có ID = 1, priority = 1.
  - o 20000 lần cho task có ID = 2, priority = 2.
  - o 10000 lần cho task có ID = 3, priority = 3.
- Sau đó, ta in thông tin của task, và số lần lặp của vòng for để kiểm tra xem có task nào bị pre-empt (thay thế) khi đang chạy hay không. Nếu 1 task nào đó bị pre-empt (thay thế), thì số lần lặp tương ứng được in ra sẽ không bằng 30000 tương ứng với task ID = 1, 20000 tương ứng với task ID = 2, 10000 tương ứng với task ID = 3.
- Sau khi đã in ra, ta dùng lệnh taskYIELD() để cho phép task khác chạy.
- Cuối cùng, ta cho task delay trong 1 tick (1 khoảng rất nhỏ, bé hơn thời gian task chạy vòng for).

```

void task(void *id)
{
    uint8_t task_id = (uint8_t) id;

    while (1)
    {
        time_count[task_id-1] = 0;

        for (int i = 0; i < (40000 - 10000*task_id); i++)
        {
            time_count[task_id-1]++;
        }

        print_task_information(task_id);

        time_count[task_id - 1] = 0;
    }
}

```



```
taskYIELD();

vTaskDelay(1);
}

vTaskDelete(NULL);
}
```

### 3.5. Hàm main

- Đầu tiên, ta tiến hành khởi tạo GPIO và interrupt.
- Tiếp theo, ta tạo 3 task, với priority tăng dần, và id = priority (từ 1 tới 3).

```
void app_main()
{
    vTaskPrioritySet(NULL, 10);

    xTaskCreate(task, "id1", 2048, (void*) 1u, 1, NULL);
    xTaskCreate(task, "id2", 2048, (void*) 2u, 2, NULL);
    xTaskCreate(task, "id3", 2048, (void*) 3, 3, NULL);

    vTaskDelete(NULL);
}
```

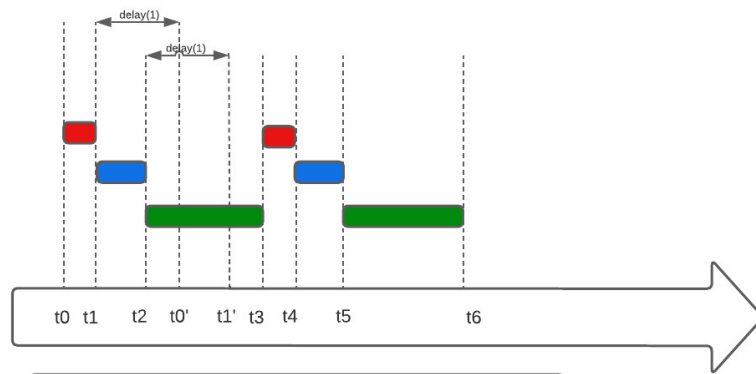
### 3.6. Kết quả hiện thực và biểu đồ thời gian

- Kết quả hiện thực: trong file lab3/ex3/output.txt.
- Biểu đồ thời gian của kết quả (theo file output):

TASK 3 (Priority 3)

TASK 2 (Priority 2)

TASK 1 (Priority 1)



- t0': TASK 3 was ready, but it did not preempt TASK 1
- t1': TASK 2 was ready, but it did not preempt TASK 1
- t3: TASK 3, TASK 2 both were ready, TASK 3 was chosen because of the higher priority

## **LINK GITHUB**

[duytran1511/Embedded\\_System\\_LAB \(github.com\)](https://github.com/duytran1511/Embedded_System_LAB)