**ĐẠI HỌC QUỐC GIA**

**ĐẠI HỌC BÁCH KHOA TP HỒ CHÍ MINH**

🙞∙∙∙☼∙∙∙🙜



**– Embedded System–**

**LAB 3**

**Nhóm 2 – L02**

|  |  |
| --- | --- |
| **Họ và tên** | **MSSV** |
| 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**

[ĐÓNG GÓP CỦA CÁC THÀNH VIÊN 2](#_Toc120304186)

[BÁO CÁO 3](#_Toc120304187)

[**1.** **Prioritized Pre-emptive Scheduling with Time Slicing** 3](#_Toc120304188)

[1.1. Điều chỉnh trong menuconfig 3](#_Toc120304189)

[1.2. Điều chỉnh trong file FreeRTOSConfig.h 3](#_Toc120304190)

[1.3. Hàm in thông tin của Task ứng với ID 3](#_Toc120304191)

[1.4. Hàm dành cho task liên tục (có Priority bằng nhau) 3](#_Toc120304192)

[1.5. Hàm danh cho task có Priority cao hơn 3](#_Toc120304193)

[1.6. Hàm main 4](#_Toc120304194)

[1.7. Kết quả hiện thực và biểu đồ thời gian 5](#_Toc120304195)

[**2.** **Prioritized Pre-emptive Scheduling (without Time Slicing)** 5](#_Toc120304196)

[2.1. Điều chỉnh trong menuconfig 5](#_Toc120304197)

[2.2. Điều chỉnh trong file FreeRTOSConfig.h 5](#_Toc120304198)

[2.3. Phần hiện thực Code 5](#_Toc120304199)

[2.4. Kết quả hiện thực và biểu đồ thời gian 5](#_Toc120304200)

[**3.** **Co-operative Scheduling** 6](#_Toc120304201)

[3.1. Điều chỉnh trong menuconfig 6](#_Toc120304202)

[3.2. Điều chỉnh trong file FreeRTOSConfig.h 6](#_Toc120304203)

[3.3. Hàm in thông tin của Task ứng với ID 6](#_Toc120304204)

[3.4. Hàm task 7](#_Toc120304205)

[3.5. Hàm main 8](#_Toc120304206)

[3.6. Kết quả hiện thực và biểu đồ thời gian 8](#_Toc120304207)

[LINK GITHUB 10](#_Toc120304208)

**ĐÓNG GÓP CỦA CÁC THÀNH VIÊN**

|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **MSSV** | **Đóng góp** |
| 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. Điều chỉnh trong menuconfig

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

|  |
| --- |
| #define configUSE\_TIME\_SLICING 1  #define configUSE\_PREEMPTION 1 |

* 1. 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. 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);  } |

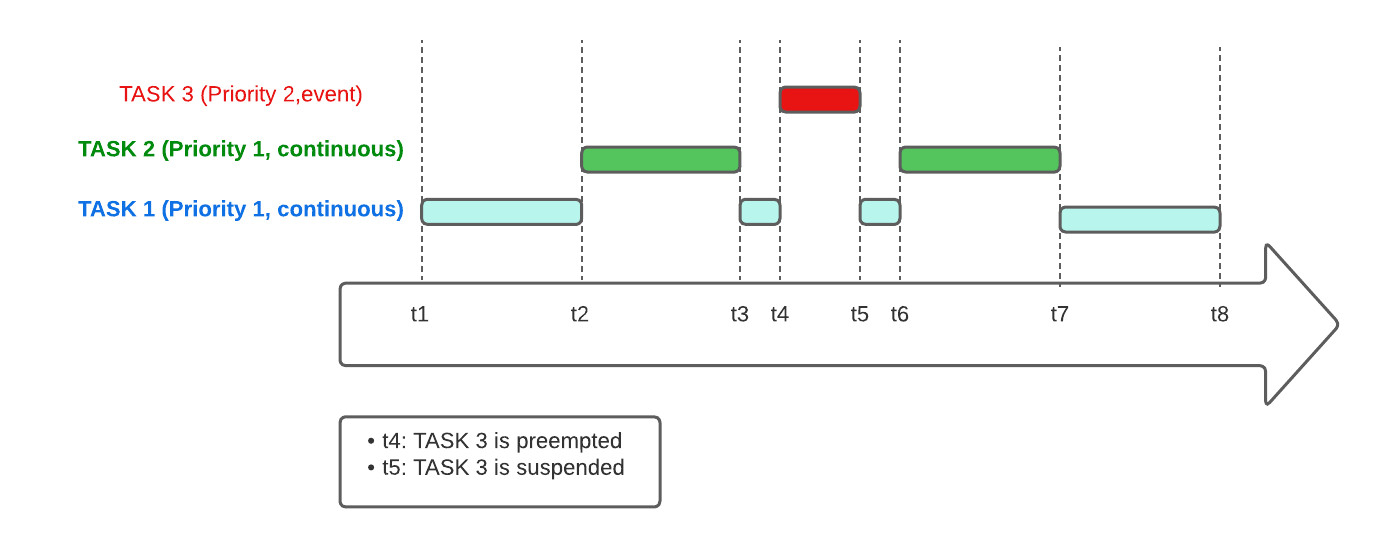
* 1. Hàm danh cho task có Priority cao hơn
* Trong vòng while(1), ta gọi hàm print\_task\_information để in thống tin của task với ID = 3.
* Sau đó, ta gọi hàm vTaskDelay(30) cho task này vào trạng Block mỗi 30 tick.

|  |
| --- |
| 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. 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. 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):

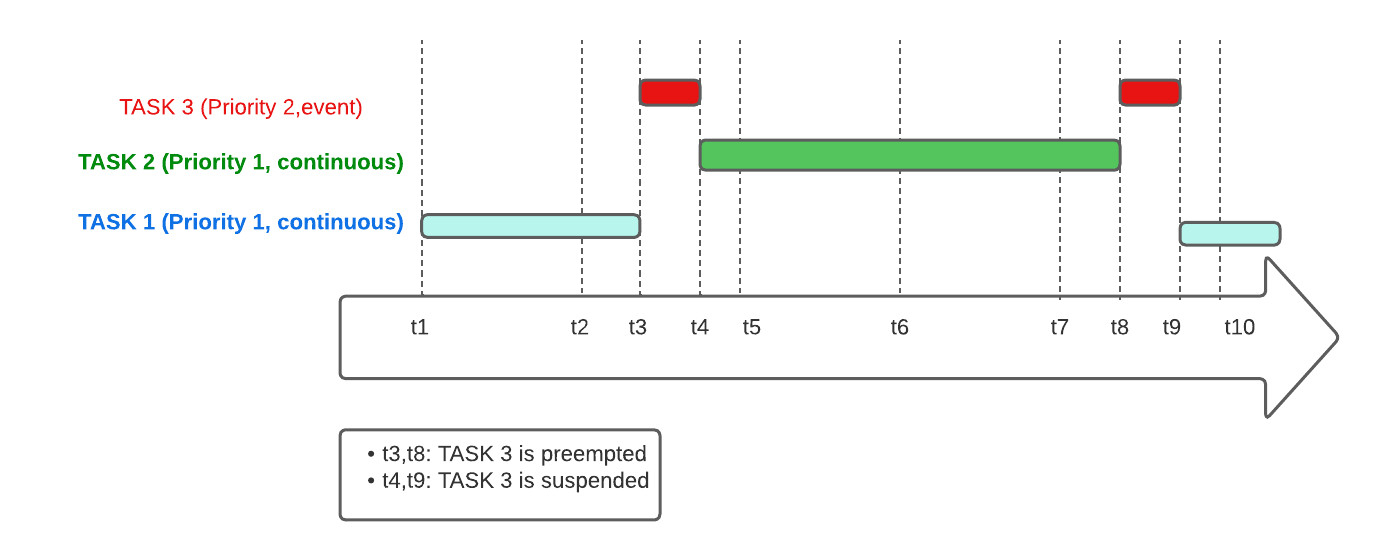


1. **Prioritized Pre-emptive Scheduling (without Time Slicing)**
   1. Điều chỉnh trong menuconfig

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

|  |
| --- |
| #define configUSE\_TIME\_SLICING 0  #define configUSE\_PREEMPTION 1 |

* 1. Phần hiện thực Code
* Tương tự như code của bài 1 (phần 2.2, 2.3, 2.4, 2.5, 2.6).
  1. 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):



1. **Co-operative Scheduling**
   1. Điều chỉnh trong menuconfig

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

|  |
| --- |
| #define configUSE\_TIME\_SLICING 0  #define configUSE\_PREEMPTION 0 |

* 1. Hàm in thông tin của Task ứng với ID
* Ta in
  + ID của task tương ứng,
  + biến đếm của vòng for cho task có Priority bé nhất (priority = 1, id = 1).
  + biến đếm của vòng for cho task có Priority lớn hơn (priority = 2, id = 2).
  + 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]);  } |

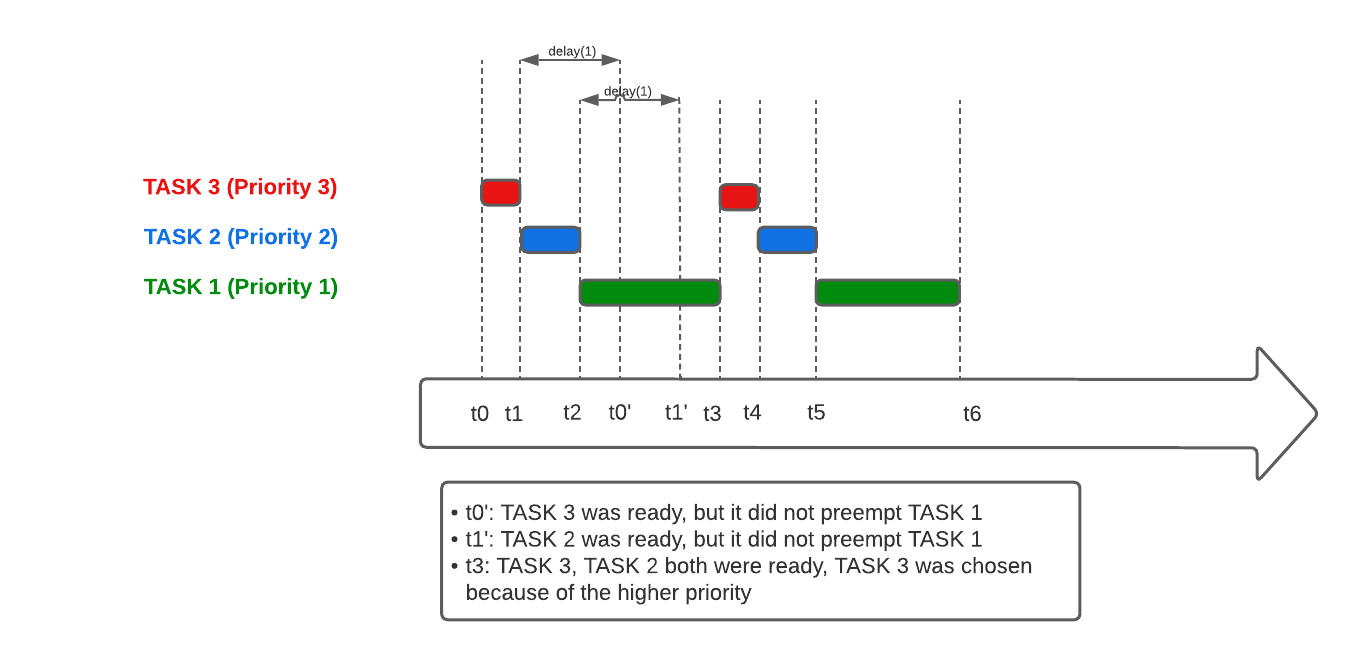
* 1. Hàm task
* Đầu tiên, ta cho hàm chạy vòng lặp for, với số lần lặp tỉ lệ nghịch với priority của task. Cụ thể là:
  + 30000 lần cho task có ID = 1, priority = 1.
  + 20000 lần cho task có ID = 2, priority = 2.
  + 10000 lần cho task có ID = 3, priority = 3.
* Sau đó, ta in thông tin của task, và 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-emt (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);  } |

* 1. 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);  } |

* 1. 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):



# LINK GITHUB

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