ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO HỆ THỐNG NHÚNG

Bài thực hành số 3

Lương Hữu Phú Lợi - 1911545
Cao Thanh Lương - 1914076
Nguyễn Văn Việt - 1912436
Huỳnh Ngọc Bảo Trân - 1912269

Họ và tên	MSSV	Đóng góp
Lương Hữu Phú Lợi	1911545	100%
Cao Thanh Lương	1914076	100%
Nguyễn Văn Việt	1912436	100%
Huỳnh Ngọc Bảo Trân	1912269	100%

GitHub:
PhuLoi-1911545/school-LAB-embedded-ESP-IDF (github.com)

a) Prioritized Pre-emptive Scheduling with Time Slicing

Ta có đoạn chương trình như sau:

```
static void task_1(void* args)
{
    while(1)
    {
        printf("\n L01 - Nhom 3 - TASK 1 +\n");
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

static void task_2(void* args)
{
    while(1)
    {
        printf("\n L01 - Nhom 3 - TASK 2\n");
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

static void task_3(void* args)
{
    while(1)
    {
        printf("\n L01 - Nhom 3 - TASK 3\n");
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void app_main(void) {
    vTaskDelay(pdMS_TO_TICKS(100));

    xTaskCreatePinnedToCore(task_3, "task 3", 1024, NULL, 0, NULL,
0);
    xTaskCreatePinnedToCore(task_2, "task 2", 1024, NULL, 0, NULL,
0);
    vTaskDelay(pdMS_TO_TICKS(5000));
    xTaskCreatePinnedToCore(task_1, "task 1", 1024, NULL, 1, NULL,
0);
```

```
}
```

Ta vào đường dẫn:
… esp-idf\components\freertos\include\esp_additions\freertos
rồi mở file FreeRTOSConfig.h

Tìm và chỉnh sửa 2 dòng sau:

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

Sau đó nạp code xuống ESP32 và được kết quả như hình bên dưới. Ta thấy task_2 và task_3 luân phiên nhau chạy (task_2 bị delay 100ms và task_3 bị delay 1000ms). Cho đến khi task_1 có priority cao hơn (priority 1) nhảy vào pre-empt chiếm quyền chạy.

PROBLEMS 4 OUTPUT DEBUG CONSOLE JUPYTER COMMENTS TERMINAL

```
I (230) cpu_start: Compile time:     Oct 31 2022 13:23:10
I (237) cpu_start: ELF file SHA256:  49e65afcdbc88f5f...
I (242) cpu_start: ESP-IDF:          v4.4.2
I (248) heap_init: Initializing. RAM available for dynamic allocation:
I (255) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (261) heap_init: At 3FFB2C30 len 0002D3D0 (180 KiB): DRAM
I (267) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (273) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (280) heap_init: At 4008B2C4 len 00014D3C (83 KiB): IRAM
I (287) spi_flash: detected chip: generic
I (291) spi_flash: flash io: dio
I (296) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.

 L01 - Nhom 3 - TASK 3

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 2

 L01 - Nhom 3 - TASK 3

 L01 - Nhom 3 - TASK 2
```

main > C hello_world_main.c > taskTwo(void *)

```
16  {
17      while(1)
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE JUPYTER COMMENTS TE

```
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 1 +
L01 - Nhom 3 - TASK 3
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 1 +
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 1 +
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 1 +
L01 - Nhom 3 - TASK 1 +
L01 - Nhom 3 - TASK 2
L01 - Nhom 3 - TASK 1 +
L01 - Nhom 3 - TASK 2
```

b) Prioritized Pre-emptive Scheduling (without Time Slicing)

Ta có đoạn chương trình sau:

```
static void task_1(void* arg)
{
    while(1)
    {
        for (int j=0; j<5; j++)
        {
             printf("L01 - Nhom 3 - TASK 1 \n");
        }
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

static void task_2(void* arg)
{
    while(1)
    {
        printf("\n L01 - Nhom 3 - TASK 2\n");
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

static void task_3(void* arg)
{
    while(1)
    {
        printf("\n L01 - Nhom 3 - TASK 3\n");
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}

void app_main(void) {
    vTaskDelay(pdMS_TO_TICKS(100));

    xTaskCreatePinnedToCore(task_3, "task 3", 1024, NULL, 0, NULL,
0);
    xTaskCreatePinnedToCore(task_2, "task 2", 1024, NULL, 0, NULL,
0);
    vTaskDelay(pdMS_TO_TICKS(5000));
```

```
    xTaskCreatePinnedToCore(task_1, "task 1", 1024, NULL, 1, NULL,
0);

}
```

Ta tiến hành đổi giá trị của configUSE_TIME_SLICING thành 0

```
#define configUSE_PREEMPTION                    1
#define configUSE_TIME_SLICING                  0
```

Sau đó nạp code xuống ESP32 và được kết quả như hình bên dưới. Ta thấy vì không sử dụng time slicing cho nên task_3 sẽ được chạy liên tục, không cho task_2 nhảy vào pre-empt. Sau đó task_1 nhảy vào pre-empt task_3 (bởi vì task_1 có priority là 1 cao hơn priority 0 của task_3). Sau khi task_1 thực thi xong, lúc này task_2 sẽ được lấy ra khỏi hàng đợi trước và được thực thi.

```
L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3

L01 - Nhom 3 - TASK 3
L01 - Nhom 3 - TASK 1
L01 - Nhom 3 - TASK 1
L01 - Nhom 3 - TASK 1
L01 - Nhom 3 - TASK 1
L01 - Nhom 3 - TASK 1

L01 - Nhom 3 - TASK 2

L01 - Nhom 3 - TASK 2

L01 - Nhom 3 - TASK 2

L01 - Nhom 3 - TASK 2

L01 - Nhom 3 - TASK 2
```

c) Co-operative Scheduling

Ta có đoạn chương trình sau:

```
#include <stdio.h>
#include <stdlib.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "driver/gpio.h"

#include "freertos/semphr.h"
#include "esp_err.h"


static void task_1(void* arg)
{
    while (1) {
        printf("Task 1 \n");
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

static void task_2(void* arg)
{
    while (1) {
        printf("Task 2 \n");
        vTaskDelay(pdMS_TO_TICKS(4000));
        taskYIELD();
    }
}

static void task_3(void* arg)
{
    while (1) {
        printf("Task 3 \n");
        vTaskDelay(pdMS_TO_TICKS(1000));
        taskYIELD();
    }
}
```

```
void app_main(void)
{
    vTaskDelay(pdMS_TO_TICKS(100));
    xTaskCreatePinnedToCore(task_3, "task 3", 1024, NULL, 0, NULL,
0);
    // vTaskDelay(pdMS_TO_TICKS(4000));
    xTaskCreatePinnedToCore(task_2, "task 2", 1024, NULL, 0, NULL,
0);
    vTaskDelay(pdMS_TO_TICKS(5000));
    xTaskCreatePinnedToCore(task_1, "task 1", 1024, NULL, 1, NULL,
0);
}
```

Ta tiếp tục đổi giá trị của configUSE_PREEMPTION thành 0

```
#define configUSE_PREEMPTION                    0
#define configUSE_TIME_SLICING                  0
```

Sau đó nạp code xuống ESP32 và được kết quả như hình bên dưới. Ta thấy task_3 được khởi tạo và chạy trước, sau đó sẽ đến task_2 (task_3 bị delay 1000ms còn task_2 bị delay 4000ms) sau khi task 3 gọi taskYield. Sau đó task_1 nhảy vào và được thực thi sau khi task_3 hoặc task_2 (task được thực thi trước đó) gọi taskYield. Sau khi task_1 gọi delay thì task kế tiếp (task_2 hoặc task_3) sẽ được thực thi.

```
I (0) cpu_start:
I (214) cpu_star
I (214) cpu_star
I (214) cpu_star
I (218) cpu_star
I (224) cpu_star
I (228) cpu_star
I (234) cpu_star
I (240) cpu_star
I (245) heap_ini
ocation:
I (252) heap_ini
I (258) heap_ini
I (265) heap_ini
I (271) heap_ini
I (277) heap_ini
I (285) spi_flas
I (288) spi_flas
I (293) cpu_star
I (0) cpu_start:
Task 3
Task 3
Task 3
Task 3
Task 2
Task 3
Task 3
Task 3
Task 3
Task 2
Task 3
Task 1
Task 3
Task 1
Task 3
Task 1
Task 3
Task 1
Task 2
Task 3
Task 1
Task 3
```