REPORT LAB

EMBEDDED SYSTEM - CO3054

Group: **CC02**

Student: **Dương Gia An – 1952163**

I. LAB 3 - Preemptive with Slicing

1. Prioritized Pre-emptive Scheduling with Time Slicing.

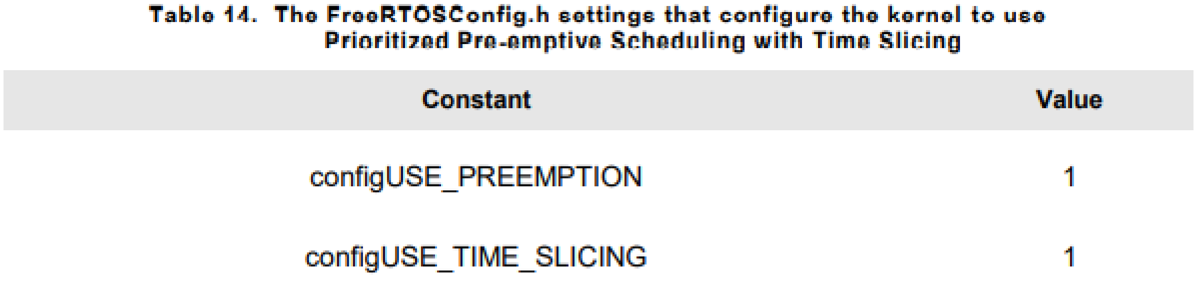
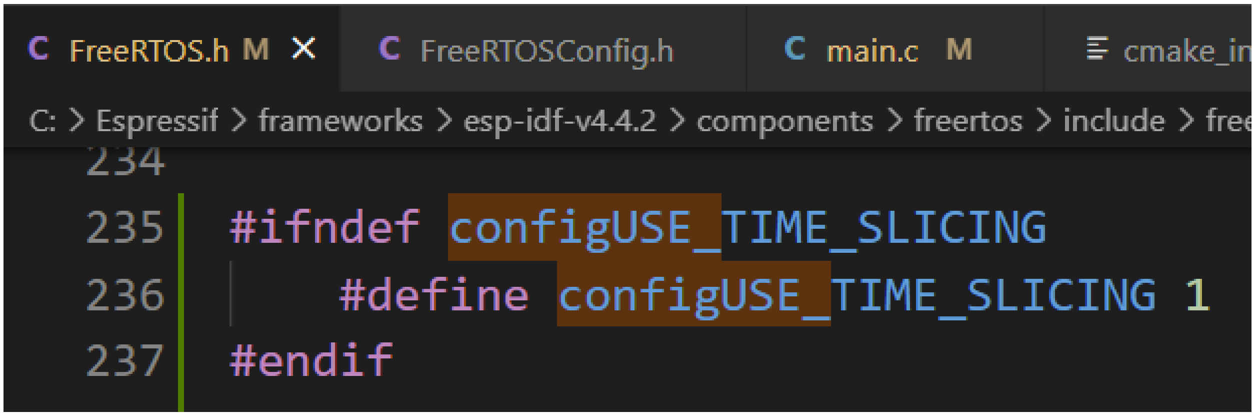
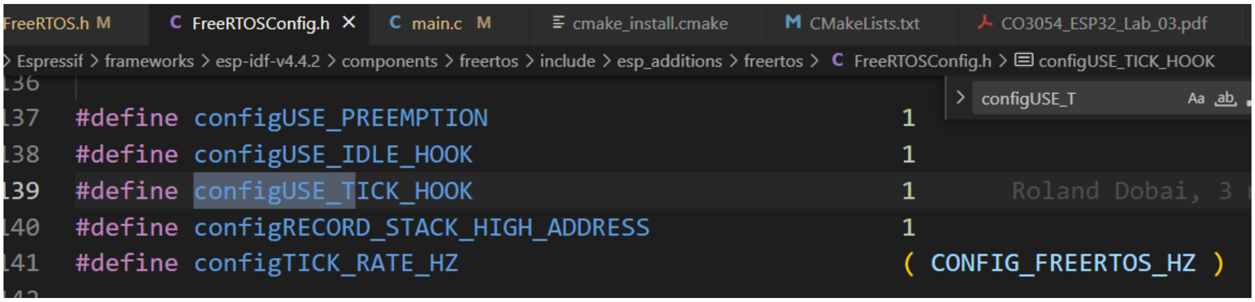


Table 14. The FreeRTOSConfig.h settings that configure the kernel to use Prioritized Pre-emptive Scheduling with Time Slicing

Constant	Value
configUSE_PREEMPTION	1
configUSE_TIME_SLICING	1


C FreeRTOS.h M ×
C FreeRTOSConfig.h
C main.c M
cmake_in
C: > Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > free

```c
#ifndef configUSE_TIME_SLICING
    #define configUSE_TIME_SLICING 1
#endif
```

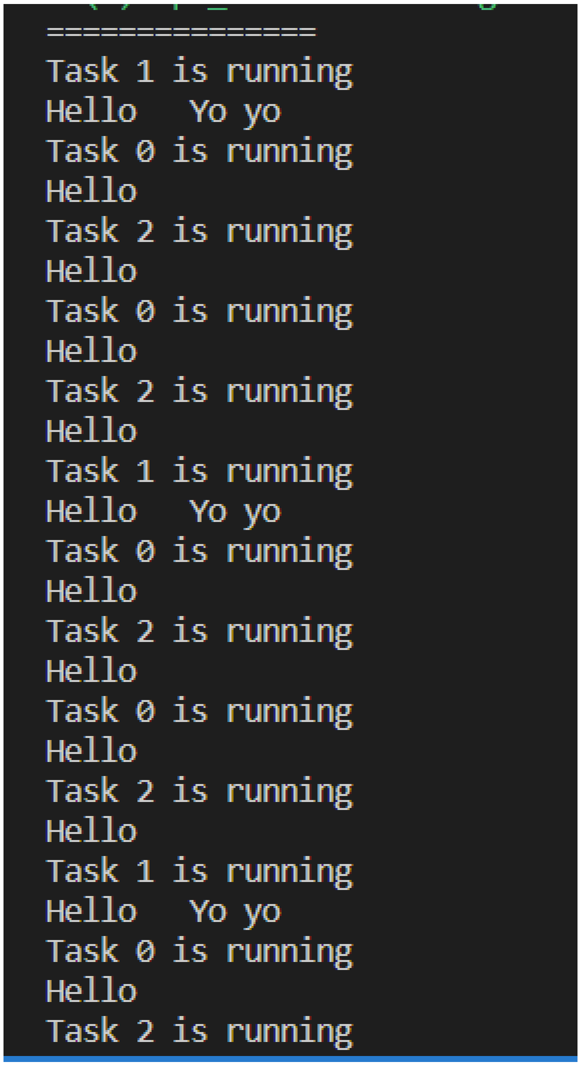

FreeRTOS.h M
C FreeRTOSConfig.h ×
C main.c M
cmake_install.cmake
M CMakeLists.txt
CO3054_ESP32_Lab_03.pdf
> Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > esp_additions > freertos > C FreeRTOSConfig.h > configUSE_TICK_HOOK
configUSE_T
Aa
ab
Roland Dobai, 3

```c
#define configUSE_PREEMPTION                            1
#define configUSE_IDLE_HOOK                             1
#define configUSE_TICK_HOOK                             1
#define configRECORD_STACK_HIGH_ADDRESS                 1
#define configTICK_RATE_HZ                              ( CONFIG_FREERTOS_HZ )
```

Code:

```c
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "freertos/FreeRTOSConfig.h"

void vTaskFunction(void *pvParameters)
{
        char *pcTaskName;
        const TickType_t xDelay250ms = pdMS_TO_TICKS(250);

        /*The string to print out is passed in via the parameter .
        Cast this to a character pointer . */
        pcTaskName = (char*) pvParameters;
        /*As per most tasks, this task is implemented in
        an infinite loop . */
        for (;;)
        {
                /*Print out the name of this task . */
                printf(pcTaskName);
                printf("Hello\n");

                /*Delay for a period . This time a call to vTaskDelay
()
                 is used which places the task into the Blocked state
                 until the delay period has expired . The parameter
takes
                 a time specified in " ticks ", and the pdMS_TO_TICKS
() macro
                 is used (where the xDelay250ms constant is declared)
to
                convert 250 milliseconds into an equivalent time in
ticks .*/
                vTaskDelay(xDelay250ms);
        }

        vTaskDelete(NULL);
}

void vTaskFunction1(void *pvParameters)
{
        char *pcTaskName;
        const TickType_t xDelay500ms = pdMS_TO_TICKS(500);

        /*The string to print out is passed in via the parameter .
        Cast this to a character pointer . */
        pcTaskName = (char*) pvParameters;
        /*As per most tasks, this task is implemented in
        an infinite loop . */
        for (;;)
        {
                /*Print out the name of this task . */
                printf(pcTaskName);
                printf("Hello\tYo yo\n");

                /*Delay for a period . This time a call to vTaskDelay
()
                 is used which places the task into the Blocked state
                 until the delay period has expired . The parameter
takes
                 a time specified in " ticks ", and the pdMS_TO_TICKS
() macro
                 is used (where the xDelay250ms constant is declared)
to
                convert 250 milliseconds into an equivalent time in
ticks .*/
                vTaskDelay(xDelay500ms);
        }

        vTaskDelete(NULL);
}

static
const char *pcTextForTask1 = "Task 1 is running \r\n";
static
const char *pcTextForTask2 = "Task 2 is running \r\n";
static
const char *pcTextForTask0 = "Task 0 is running \r\n";
```

```
void app_main(void)
{
        printf("===============\n");

        xTaskCreate(vTaskFunction, " Task 0 Ilde", 2048,
                (void*) pcTextForTask0, 0, NULL);

        /*Create the first task at priority 1.
 The priority is the second to last parameter . */
        xTaskCreate(vTaskFunction1, " Task 1", 2048,
                (void*) pcTextForTask1, 1, NULL);

        /*Create the second task at priority 2,        which is higher
than a priority of 1.
         The priority is the second to last parameter . */
        xTaskCreate(vTaskFunction, " Task 2", 2048,
                (void*) pcTextForTask2, 0, NULL);


        /*Start the scheduler so the tasks start executing . */
        //vTaskStartScheduler();

        /*Will not reach here . */
}
```



```
===============
Task 1 is running
Hello   Yo yo
Task 0 is running
Hello
Task 2 is running
Hello
Task 0 is running
Hello
Task 2 is running
Hello
Task 1 is running
Hello   Yo yo
Task 0 is running
Hello
Task 2 is running
Hello
Task 0 is running
Hello
Task 2 is running
Hello
Task 1 is running
Hello   Yo yo
Task 0 is running
Hello
Task 2 is running
```

Explain result: Task Idle , task 1, task 2 add to schedule same time. And task 1 has Largest Priotirity is 1 so it run first (print Hello Yo yo). Then task 0 (Idle task) and task 2 run.

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

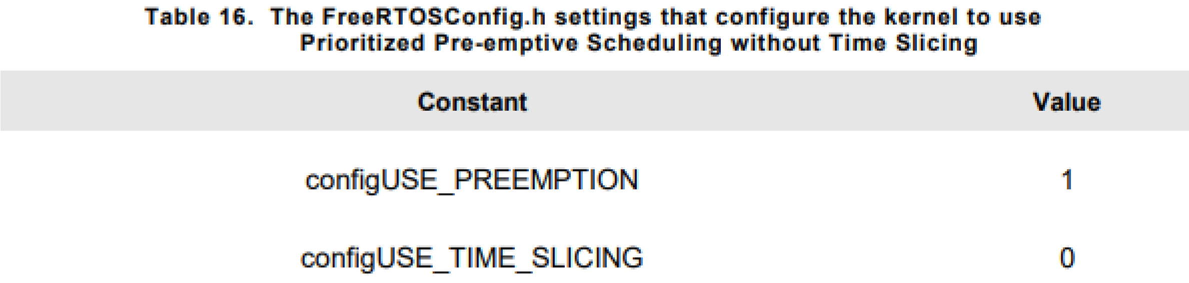
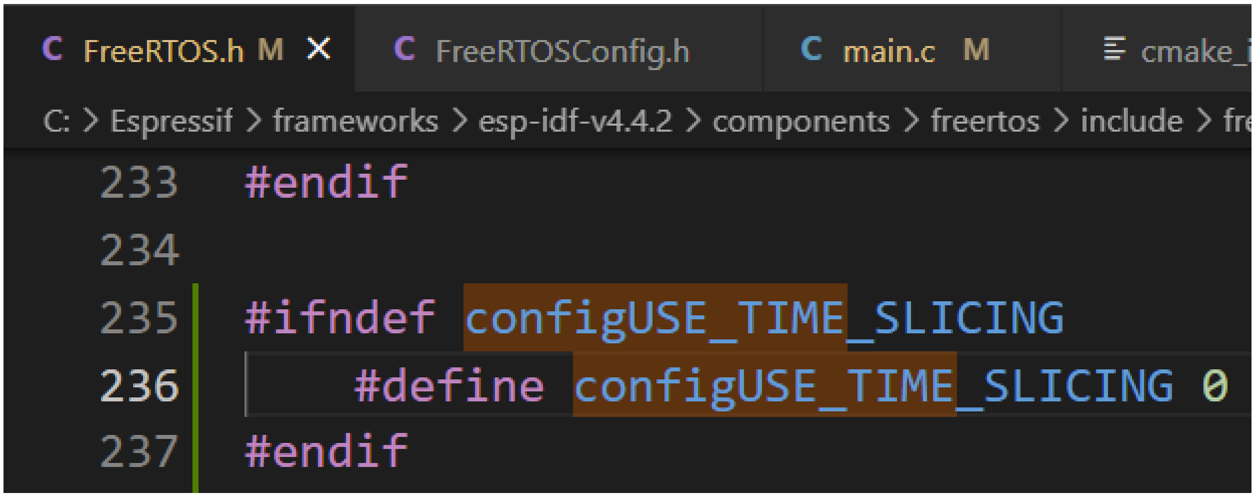


Table 16. The FreeRTOSConfig.h settings that configure the kernel to use Prioritized Pre-emptive Scheduling without Time Slicing

Constant	Value
configUSE_PREEMPTION	1
configUSE_TIME_SLICING	0

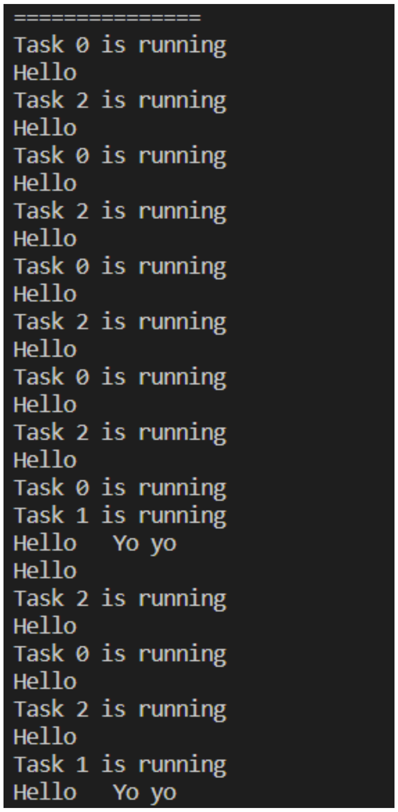

C FreeRTOS.h M ×
C FreeRTOSConfig.h
C main.c M
cmake_i
C: > Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > fre

```
#endif

#ifndef configUSE_TIME_SLICING
    #define configUSE_TIME_SLICING 0
#endif
```

Code:

```
void app_main(void)
{
        printf("===============\n");

    xTaskCreate(vTaskFunction, " Task 0 Ilde", 2048,
                (void *)pcTextForTask0, 0, NULL);
    /* Create the first task at priority 1.
 The priority is the second to last parameter . */


    /* Create the second task at priority 2 ,
     which is higher than a priority of 1.
     The priority is the second to last parameter . */
    xTaskCreate(vTaskFunction, " Task 2", 2048,
                (void *)pcTextForTask2, 0, NULL);

    //Delay add Task 1 after 1000ms

    vTaskDelay(1000 / portTICK_PERIOD_MS);

    xTaskCreate(vTaskFunction1, " Task 1", 2048,
                (void *)pcTextForTask1, 1, NULL);

    /* Start the scheduler so the tasks start executing . */
    //vTaskStartScheduler();

    /* Will not reach here . */
}
```

Result:



```
===============
Task 0 is running
Hello
Task 2 is running
Hello
Task 0 is running
Hello
Task 2 is running
Hello
Task 0 is running
Hello
Task 2 is running
Hello
Task 0 is running
Hello
Task 2 is running
Hello
Task 0 is running
Task 1 is running
Hello   Yo yo
Hello
Task 2 is running
Hello
Task 0 is running
Hello
Task 2 is running
Hello
Task 1 is running
Hello   Yo yo
```

Explain Result:

I add Task Idle and Task 2 first and they run. (delay 250ms each)

After 1000ms (4 times for Idle and Task 2 runed) , I add task 1 to schedule. Task 1 has Highest Priority, so it run.

3.

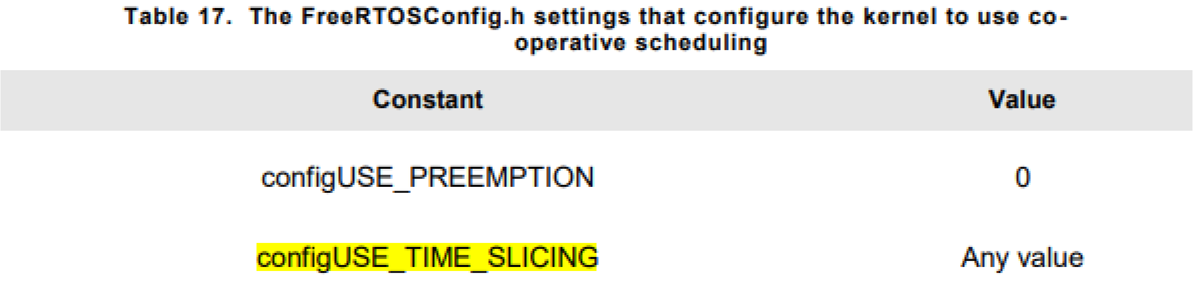
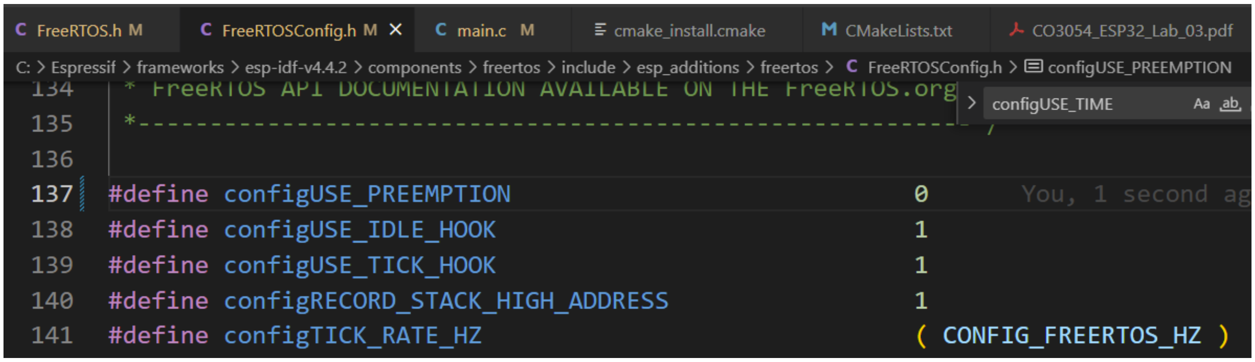


Table 17. The FreeRTOSConfig.h settings that configure the kernel to use co-operative scheduling

Constant	Value
configUSE_PREEMPTION	0
configUSE_TIME_SLICING	Any value



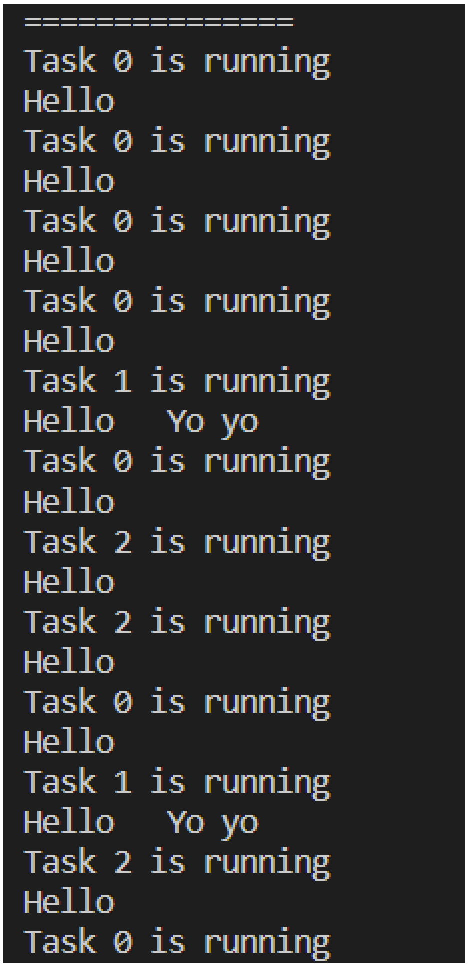
C FreeRTOS.h M
C FreeRTOSConfig.h M ×
C main.c M
cmake_install.cmake
M CMakeLists.txt
CO3054_ESP32_Lab_03.pdf
C: > Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > esp_additions > freertos > C FreeRTOSConfig.h > configUSE_PREEMPTION
configUSE_TIME
Aa ab
You, 1 second ag

```
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org
 *----------------------------------------------------------------------------*/

#define configUSE_PREEMPTION                    0
#define configUSE_IDLE_HOOK                     1
#define configUSE_TICK_HOOK                     1
#define configRECORD_STACK_HIGH_ADDRESS         1
#define configTICK_RATE_HZ                      ( CONFIG_FREERTOS_HZ )
```

Code:

```c
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "freertos/FreeRTOSConfig.h"
void vTaskFunction(void *pvParameters)
{
    char *pcTaskName;
    const TickType_t xDelay250ms = pdMS_TO_TICKS(250);
    TickType_t count1000ms = pdMS_TO_TICKS(0);
    /* The string to print out is passed in via the parameter .
    Cast this to a character pointer . */
    pcTaskName = (char *)pvParameters;
    /* As per most tasks , this task is implemented in
    an infinite loop . */
    for (;;)
    {
        /* Print out the name of this task . */
        printf(pcTaskName);
        printf("Hello\n");

        /* Delay for a period . This time a call to vTaskDelay ()
         is used which places the task into the Blocked state
         until the delay period has expired . The parameter takes
         a time specified in " ticks " , and the pdMS_TO_TICKS ()
macro
         is used ( where the xDelay250ms constant is declared ) to
        convert 250 milliseconds into an equivalent time in ticks .*/
        vTaskDelay(xDelay250ms);
        if (count1000ms == pdMS_TO_TICKS(1000)){
            taskYIELD();
        }
        else count1000ms += pdMS_TO_TICKS(250);
    }
    vTaskDelete(NULL);
}

void vTaskFunction1(void *pvParameters)
{
    char *pcTaskName;
    const TickType_t xDelay500ms = pdMS_TO_TICKS(500);

    /* The string to print out is passed in via the parameter .
    Cast this to a character pointer . */
    pcTaskName = (char *)pvParameters;
    /* As per most tasks , this task is implemented in
    an infinite loop . */
    for (;;)
    {
        /* Print out the name of this task . */
        printf(pcTaskName);
        printf("Hello\tYo yo\n");

        /* Delay for a period . This time a call to vTaskDelay ()
         is used which places the task into the Blocked state
         until the delay period has expired . The parameter takes
         a time specified in " ticks " , and the pdMS_TO_TICKS ()
macro
         is used ( where the xDelay250ms constant is declared ) to
        convert 250 milliseconds into an equivalent time in ticks .*/
        vTaskDelay(xDelay500ms);
    }
    vTaskDelete(NULL);
}
static const char *pcTextForTask1 = "Task 1 is running \r\n";
static const char *pcTextForTask2 = "Task 2 is running \r\n";
static const char *pcTextForTask0 = "Task 0 is running \r\n";

void app_main(void)
{

    printf("===============\n");

    xTaskCreate(vTaskFunction, " Task 0 Ilde", 2048,
                (void *)pcTextForTask0, 0, NULL);
    /* Create the first task at priority 1.
 The priority is the second to last parameter . */


    /* Create the second task at priority 2 ,
     which is higher than a priority of 1.
     The priority is the second to last parameter . */


    //Delay add Task 1 and Task 2 after 1000ms

    vTaskDelay(1000 / portTICK_PERIOD_MS);

    xTaskCreate(vTaskFunction1, " Task 1", 2048,
                (void *)pcTextForTask1, 1, NULL);
    xTaskCreate(vTaskFunction, " Task 2", 2048,
                (void *)pcTextForTask2, 0, NULL);
    /* Start the scheduler so the tasks start executing . */
    //vTaskStartScheduler();

    /* Will not reach here . */
}
```

Result:



```
===============
Task 0 is running
Hello
Task 0 is running
Hello
Task 0 is running
Hello
Task 0 is running
Hello
Task 1 is running
Hello   Yo yo
Task 0 is running
Hello
Task 2 is running
Hello
Task 2 is running
Hello
Task 0 is running
Hello
Task 1 is running
Hello   Yo yo
Task 2 is running
Hello
Task 0 is running
```

Explain Result:

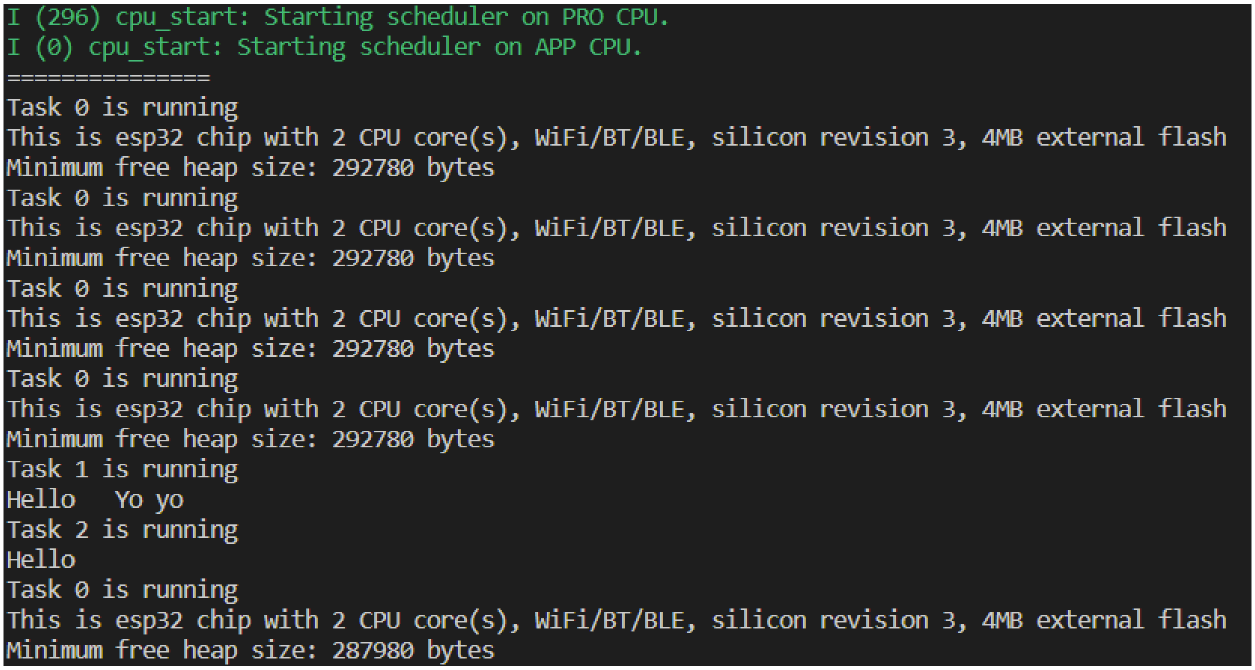
I add task 0 (Idle first).
After 1000ms, I add task 1 and 2. As the same time, TaskYIELD() in Idle run so Task 1 has Largest Priority run then.

4. Extra Exercise

```c
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "freertos/FreeRTOSConfig.h"


void vTaskFunction_Idle(void *pvParameters)
{
    char *pcTaskName;
    const TickType_t xDelay250ms = pdMS_TO_TICKS(250);
    TickType_t count1000ms = pdMS_TO_TICKS(0);
    /* The string to print out is passed in via the parameter .
    Cast this to a character pointer . */
    pcTaskName = (char *)pvParameters;
    /* As per most tasks , this task is implemented in
    an infinite loop . */
    for (;;)
    {
        /* Print out the name of this task . */
        printf(pcTaskName);

       // Hook IDLE TASK
        /* Print chip information */
        esp_chip_info_t chip_info;
        esp_chip_info(&chip_info);
        printf("This is %s chip with %d CPU core(s), WiFi%s%s, ",
            CONFIG_IDF_TARGET,
            chip_info.cores,
            (chip_info.features & CHIP_FEATURE_BT) ? "/BT" : "",
            (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" : "");

        printf("silicon revision %d, ", chip_info.revision);

        printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
            (chip_info.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");

        printf("Minimum free heap size: %d bytes\n", esp_get_minimum_free_heap_size());


        /* Delay for a period . This time a call to vTaskDelay()
         is used which places the task into the Blocked state
         until the delay period has expired . The parameter takes
         a time specified in " ticks " , and the pdMS_TO_TICKS() macro
         is used ( where the xDelay250ms constant is declared ) to
        convert 250 milliseconds into an equivalent time in ticks .*/
        vTaskDelay(xDelay250ms);

        if (count1000ms == pdMS_TO_TICKS(1000)){
            taskYIELD();
        }
        else count1000ms += pdMS_TO_TICKS(250);
    }
    vTaskDelete(NULL);
}
void vTaskFunction2(void *pvParameters)
{
    char *pcTaskName;
    const TickType_t xDelay250ms = pdMS_TO_TICKS(250);
    /* The string to print out is passed in via the parameter .
    Cast this to a character pointer . */
    pcTaskName = (char *)pvParameters;
    /* As per most tasks , this task is implemented in
    an infinite loop . */
    for (;;)
    {
        /* Print out the name of this task . */
        printf(pcTaskName);
        printf("Hello \n");
        /* Delay for a period . This time a call to vTaskDelay ()
         is used which places the task into the Blocked state
         until the delay period has expired . The parameter takes
         a time specified in " ticks " , and the pdMS_TO_TICKS () macro
         is used ( where the xDelay250ms constant is declared ) to
        convert 250 milliseconds into an equivalent time in ticks .*/
        vTaskDelay(xDelay250ms);

    }
    vTaskDelete(NULL);
}
void vTaskFunction1(void *pvParameters)
{
    char *pcTaskName;
    const TickType_t xDelay500ms = pdMS_TO_TICKS(500);

    /* The string to print out is passed in via the parameter .
    Cast this to a character pointer . */
    pcTaskName = (char *)pvParameters;
    /* As per most tasks , this task is implemented in
    an infinite loop . */
    for (;;)
    {
        /* Print out the name of this task . */
        printf(pcTaskName);
        printf("Hello\tYo yo\n");

        /* Delay for a period . This time a call to vTaskDelay ()
         is used which places the task into the Blocked state
```

```
          until the delay period has expired . The parameter takes
          a time specified in " ticks " , and the pdMS_TO_TICKS () macro
          is used ( where the xDelay250ms constant is declared ) to
         convert 500 milliseconds into an equivalent time in ticks .*/
         vTaskDelay(xDelay500ms);
    }
    vTaskDelete(NULL);
}
static const char *pcTextForTask1 = "Task 1 is running \r\n";
static const char *pcTextForTask2 = "Task 2 is running \r\n";
static const char *pcTextForTask0 = "Task 0 is running \r\n";

void app_main(void)
{


    printf("===============\n");

    xTaskCreate(vTaskFunction_Idle, " Task 0 Ilde", 2048,
                (void *)pcTextForTask0, 0, NULL);
    /* Create the first task at priority 1.
 The priority is the second to last parameter . */


    /* Create the second task at priority 2 ,
     which is higher than a priority of 1.
     The priority is the second to last parameter . */


    //Delay add Task 1 after 1000ms

    vTaskDelay(1000 / portTICK_PERIOD_MS);

    xTaskCreate(vTaskFunction1, " Task 1", 2048,
                (void *)pcTextForTask1, 1, NULL);
    xTaskCreate(vTaskFunction2, " Task 2", 2048,
                (void *)pcTextForTask2, 0, NULL);
    /* Start the scheduler so the tasks start executing . */
    //vTaskStartScheduler();

    /* Will not reach here . */
}
```

Result:



```
I (296) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
===============
Task 0 is running
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 3, 4MB external flash
Minimum free heap size: 292780 bytes
Task 0 is running
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 3, 4MB external flash
Minimum free heap size: 292780 bytes
Task 0 is running
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 3, 4MB external flash
Minimum free heap size: 292780 bytes
Task 0 is running
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 3, 4MB external flash
Minimum free heap size: 292780 bytes
Task 1 is running
Hello   Yo yo
Task 2 is running
Hello
Task 0 is running
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 3, 4MB external flash
Minimum free heap size: 287980 bytes
```

II. FreeRTOS Queue Management

Not finish yet.