

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

```

234
235 #ifndef configUSE_TIME_SLICING
236     #define configUSE_TIME_SLICING 1
237 #endif

```

```

137 #define configUSE_PREEMPTION 1
138 #define configUSE_IDLE_HOOK 1
139 #define configUSE_TICK_HOOK 1
140 #define configRECORD_STACK_HIGH_ADDRESS 1
141 #define configTICK_RATE_HZ ( CONFIG_FREERTOS_HZ )

```

Search results for 'configUSE_T':

- configUSE_T

Code:

```

#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 X C FreeRTOSConfig.h C main.c M ≡ cmake_
C: > Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > fre

233 #endif
234
235 #ifndef configUSE_TIME_SLICING
236     #define configUSE_TIME_SLICING 0
237 #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: > Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > esp_additions > freertos > FreeRTOSConfig.h > configUSE_PREEMPTION
134 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB
135 *-----
136
137 #define configUSE_PREEMPTION 0
138 #define configUSE_IDLE_HOOK 1
139 #define configUSE_TICK_HOOK 1
140 #define configRECORD_STACK_HIGH_ADDRESS 1
141 #define configTICK_RATE_HZ ( CONFIG_FREERTOS_HZ )
```

Code:

```
#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

```
5. #include <stdio.h>
6. #include "sdkconfig.h"
7. #include "freertos/FreeRTOS.h"
8. #include "freertos/task.h"
9. #include "esp_system.h"
10. #include "esp_spi_flash.h"
11. #include "freertos/FreeRTOSConfig.h"
12.
13.
14. void vTaskFunction_Idle(void *pvParameters)
15. {
16.     char *pcTaskName;
17.     const TickType_t xDelay250ms = pdMS_TO_TICKS(250);
18.     TickType_t count1000ms = pdMS_TO_TICKS(0);
```

```

19.      /* The string to print out is passed in via the parameter
20.      .
21.      Cast this to a character pointer . */
22.      pcTaskName = (char *)pvParameters;
23.      /* As per most tasks , this task is implemented in
24.      an infinite loop . */
25.      for (;;)
26.      {
27.          /* Print out the name of this task . */
28.          printf(pcTaskName);
29.          // Hook IDLE TASK
30.          /* Print chip information */
31.          esp_chip_info_t chip_info;
32.          esp_chip_info(&chip_info);
33.          printf("This is %s chip with %d CPU core(s), Wi-Fi%s%s,
34.          ",
35.          CONFIG_IDF_TARGET,
36.          chip_info.cores,
37.          (chip_info.features & CHIP_FEATURE_BT) ? "/BT" :
38.          "",
39.          (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" :
40.          "");
41.          printf("silicon revision %d, ", chip_info.revision);
42.          printf("%dMB %s flash\n", spi_flash_get_chip_size() /
43.          (1024 * 1024),
44.          (chip_info.features & CHIP_FEATURE_EMB_FLASH) ?
45.          "embedded" : "external");
46.          printf("Minimum free heap size: %d bytes\n",
47.          esp_get_minimum_free_heap_size());
48.          /* Delay for a period . This time a call to vTaskDelay
49.          ()
50.          is used which places the task into the Blocked state
51.          until the delay period has expired . The parameter
52.          takes
53.          a time specified in " ticks " , and the pdMS_TO_TICKS
54.          () macro
55.          is used ( where the xDelay250ms constant is declared
56.          ) to
57.          convert 250 milliseconds into an equivalent time in
58.          ticks .*/
59.          vTaskDelay(xDelay250ms);
60.          if (count1000ms == pdMS_TO_TICKS(1000)) {
61.              taskYIELD();
62.          }
63.          else count1000ms += pdMS_TO_TICKS(250);
64.      }

```

```

59.     vTaskDelete(NULL);
60. }
61. void vTaskFunction2(void *pvParameters)
62. {
63.     char *pcTaskName;
64.     const TickType_t xDelay250ms = pdMS_TO_TICKS(250);
65.     /* The string to print out is passed in via the parameter
        .
66.     Cast this to a character pointer . */
67.     pcTaskName = (char *)pvParameters;
68.     /* As per most tasks , this task is implemented in
69.     an infinite loop . */
70.     for (;;)
71.     {
72.         /* Print out the name of this task . */
73.         printf(pcTaskName);
74.         printf("Hello \n");
75.         /* Delay for a period . This time a call to vTaskDelay
            ()
76.             is used which places the task into the Blocked state
77.             until the delay period has expired . The parameter
            takes
78.             a time specified in " ticks " , and the pdMS_TO_TICKS
            () macro
79.             is used ( where the xDelay250ms constant is declared
            ) to
80.             convert 250 milliseconds into an equivalent time in
            ticks .*/
81.         vTaskDelay(xDelay250ms);
82.
83.     }
84.     vTaskDelete(NULL);
85. }
86. void vTaskFunction1(void *pvParameters)
87. {
88.     char *pcTaskName;
89.     const TickType_t xDelay500ms = pdMS_TO_TICKS(500);
90.
91.     /* The string to print out is passed in via the parameter
        .
92.     Cast this to a character pointer . */
93.     pcTaskName = (char *)pvParameters;
94.     /* As per most tasks , this task is implemented in
95.     an infinite loop . */
96.     for (;;)
97.     {
98.         /* Print out the name of this task . */
99.         printf(pcTaskName);
100.        printf("Hello\tYo yo\n");
101.
102.        /* Delay for a period . This time a call to vTaskDelay
            ()
103.            is used which places the task into the Blocked state

```

```

104.         until the delay period has expired . The parameter
           takes
105.         a time specified in " ticks " , and the pdMS_TO_TICKS
           () macro
106.         is used ( where the xDelay250ms constant is declared
           ) to
107.         convert 500 milliseconds into an equivalent time in
           ticks .*/
108.         vTaskDelay(xDelay500ms);
109.     }
110.     vTaskDelete(NULL);
111. }
112. static const char *pcTextForTask1 = "Task 1 is running \r\n";
113. static const char *pcTextForTask2 = "Task 2 is running \r\n";
114. static const char *pcTextForTask0 = "Task 0 is running \r\n";
115.
116. void app_main(void)
117. {
118.
119.
120.     printf("=====\n");
121.
122.     xTaskCreate(vTaskFunction_Idle, " Task 0 Ilde", 2048,
123.                (void *)pcTextForTask0, 0, NULL);
124.     /* Create the first task at priority 1.
125.      The priority is the second to last parameter . */
126.
127.
128.     /* Create the second task at priority 2 ,
129.      which is higher than a priority of 1.
130.      The priority is the second to last parameter . */
131.
132.
133.     //Delay add Task 1 after 1000ms
134.
135.     vTaskDelay(1000 / portTICK_PERIOD_MS);
136.
137.     xTaskCreate(vTaskFunction1, " Task 1", 2048,
138.                (void *)pcTextForTask1, 1, NULL);
139.     xTaskCreate(vTaskFunction2, " Task 2", 2048,
140.                (void *)pcTextForTask2, 0, NULL);
141.     /* Start the scheduler so the tasks start executing . */
142.     //vTaskStartScheduler();
143.
144.     /* Will not reach here . */
145. }

```

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.