REPORT LAB EMBEDDED SYSTEM - CO3054

Group: CC02

Student: Dwong 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 X C FreeRTOSConfig.h C main.c M
                                                                     C: > Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > free
     234
              #ifndef configUSE_TIME_SLICING
    235
                     #define configUSE_TIME_SLICING 1
    236
              #endif
    237
          C FreeRTOSConfig.h X C main.c M

    □ cmake_install.cmake

                                                  M CMakeLists.txt
                                                              Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > esp_additions > freertos > \mathbf{C} FreeRTOSConfig.h > \mathbf{\Box} configUSE_TICK_HOOK
                                                             > configUSE_T
    #define configUSE_PREEMPTION
1
L39 #define configUSE_TICK_HOOK
   #define configRECORD STACK HIGH ADDRESS
    #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);
       /*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)
t.o
               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)
t.o
               convert 250 milliseconds into an equivalent time in
ticks .*/
               vTaskDelay(xDelay500ms);
       vTaskDelete(NULL);
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 . ^{\star}/
       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)

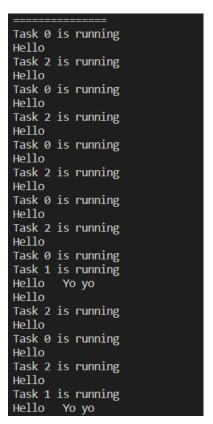
Table 16. The FreeRTOSConfig.h settings that configure the kernel to use Prioritized Pre-emptive Scheduling without Time Slicing

Constant	Value
configUSE_PREEMPTION	1
configUSE_TIME_SLICING	0

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:



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 cooperative scheduling

Constant	Value
configUSE_PREEMPTION	0
configUSE_TIME_SLICING	Any value

```
C FreeRTOS.h M
                                                                               C FreeRTOSConfig.h M X C main.c M
                                                                                                                                                                                                                                                          C: > Espressif > frameworks > esp-idf-v4.4.2 > components > freertos > include > esp_additions > freertos > C FreeRTOSConfig.h > 🗏 configUSE_PREEMPTION
                                                         rameworks Fesp-idf-v4.4.2 Fcomponents Americs America Action of the Freekius organization organization of the Freekius organization of the Freekius organization organiz
      134
      136
                                       #define configUSE_PREEMPTION
      137
                                                                                                                                                                                                                                                                                                                                                                                                    0
      138
                                        #define configUSE_IDLE_HOOK
      139
                                       #define configUSE_TICK_HOOK
                                                                                                                                                                                                                                                                                                                                                                                                    1
                                       #define configRECORD_STACK_HIGH_ADDRESS
       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 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);
        TickType t count1000ms = pdMS TO TICKS(0);
```

```
/* The string to print out is passed in via the parameter
19.
20.
        Cast this to a character pointer . */
        pcTaskName = (char *)pvParameters;
21.
22.
        /* As per most tasks , this task is implemented in
23.
        an infinite loop . */
        for (;;)
24.
25.
26.
            /* Print out the name of this task . */
27.
            printf(pcTaskName);
28.
           // Hook IDLE TASK
           /* Print chip information */
            esp chip info t chip info;
30.
31.
           esp chip info(&chip info);
32.
           printf("This is %s chip with %d CPU core(s), WiFi%s%s,
                CONFIG IDF TARGET,
33.
34.
                chip info.cores,
35.
                (chip info.features & CHIP FEATURE BT) ? "/BT" :
  11 11
                (chip info.features & CHIP FEATURE BLE) ? "/BLE" :
37.
           printf("silicon revision %d, ", chip info.revision);
38.
39.
            printf("%dMB %s flash\n", spi flash get chip size() /
  (1024 * 1024),
                (chip info.features & CHIP FEATURE EMB FLASH) ?
   "embedded" : "external");
42.
           printf("Minimum free heap size: %d bytes\n",
  esp get minimum free heap size());
44.
45.
46.
           /* Delay for a period . This time a call to vTaskDelay
 ( )
             is used which places the task into the Blocked state
47.
             until the delay period has expired . The parameter
48.
  takes
             a time specified in " ticks " , and the pdMS TO TICKS
  () macro
50.
             is used (where the xDelay250ms constant is declared
) to
           convert 250 milliseconds into an equivalent time in
 ticks .*/
52.
           vTaskDelay(xDelay250ms);
54.
            if (count1000ms == pdMS TO TICKS(1000)) {
55.
                taskYIELD();
56.
57.
            else count1000ms += pdMS TO TICKS(250);
58.
```

```
59.
        vTaskDelete(NULL);
60. }
61. void vTaskFunction2(void *pvParameters)
62. {
63.
        char *pcTaskName;
64.
        const TickType t xDelay250ms = pdMS TO TICKS(250);
        /* The string to print out is passed in via the parameter
65.
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");
            /* Delay for a period . This time a call to vTaskDelay
75.
 ( )
76.
             is used which places the task into the Blocked state
77
             until the delay period has expired . The parameter
  takes
             a time specified in " ticks " , and the pdMS TO TICKS
78.
 () macro
             is used (where the xDelay250ms constant is declared
  ) to
           convert 250 milliseconds into an equivalent time in
80.
  ticks .*/
81.
            vTaskDelay(xDelay250ms);
82.
83.
84.
        vTaskDelete(NULL);
85. }
86. void vTaskFunction1(void *pvParameters)
87. {
88.
        char *pcTaskName;
        const TickType t xDelay500ms = pdMS TO TICKS(500);
89.
90.
        /* The string to print out is passed in via the parameter
91.
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
```

```
until the delay period has expired . The parameter
104.
  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 .*/
           vTaskDelay(xDelay500ms);
108.
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";
116. void app main (void)
117. {
118.
119.
       printf("========\n");
120.
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.
       /* Will not reach here . */
144.
145.}
```

Result:

```
(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.