ใบงานที่ 9

Real-time operating system (RTOS)

สิ่งที่ต้องใช้ในการทดลอง

1. สาย USB
2. บอร์ด ESP32
3. เครื่องคอมพิวเตอร์

Create Task and Delete Task

1. โค้ดในการสร้าง task จำนวน 2 task ผลัดกันในการแสดงข้อความออกทางหน้าจอ

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"

#define STACK_SIZE 2048
#define TASK_PRIORITY 10

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

void vTaskFunction(void *pvParameters)
{
  while(1)
  {
    printf("%s", (char*)pvParameters );
    vTaskDelay( 1000 / portTICK_PERIOD_MS );
  }
}

void app_main(void)
{
  xTaskCreate( vTaskFunction,         //Task function to process
               "Task 1",              //Task name
               STACK_SIZE,            //Stack size
               (void*)pcTextForTask1, //Parameter to pass throug each task
               TASK_PRIORITY,         //Task Priority
               NULL                   //Pointer to task handler
     );

  xTaskCreate( vTaskFunction, "Task 2", STACK_SIZE, (void*)pcTextForTask2, TASK_PRIORITY, NULL );

  while(1)
  {
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
```

2. ทำการ build และโปรแกรมลงบอร์ด พร้อมทั้งบันทึกผลที่ได้จากการ monitor

พื้นที่ในการบันทึกผล

```
Task 1 is runing
Task 2 is runing
Task 1 is runing
Task 2 is runing
Task 1 is runing
Task 2 is runing
Task 1 is runing
Task 2 is runing
Task 1 is runing
Task 2 is runing
Task 1 is runing
Task 2 is runing
Task 1 is runing
Task 2 is runing
Task 1 is runing
Task 2 is runing
```

13/9/66

Queue Management

1. โค้ดในส่วนต้นเพื่อใช้งาน queue (ลักษณะการทำงานของ task ที่สร้างจะเป็น producer & consumer)

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"

uint32_t ulVar = 0;
QueueHandle_t queue;

void taskProducer(void *pvParameter)
{
  while(1)
  {
    ulVar++;
    xQueueSend(queue, (void *)&ulVar, (TickType_t)10 );
    vTaskDelay( 1000 / portTICK_PERIOD_MS );
  }
}

void taskConsumer(void *pvParameter)
{
  uint32_t data;
  while(1)
  {
    if(xQueueReceive(queue, &data, portMAX_DELAY))
    {
      printf("[Queue] Recv, val: %ld\r\n", data);
    }
  }
}
```

2. โค้ดในส่วนของการสร้าง queue และ task

```c
void app_main(void)
{
    queue = xQueueCreate( 10, sizeof( uint32_t) );

    xTaskCreate(taskProducer, "Producer", 2048, NULL, 10, NULL);

    xTaskCreate(taskConsumer, "Consumer", 2048, NULL, 10, NULL);

    while(1)
    {
        vTaskDelay(1000 / portTICK_PERIOD_MS );
    }
}
```

3. ทำการ build และสังเกตข้อความที่แสดงจาก monitor จากนั้นอธิบายเหตุการณ์ที่เกิดขึ้น

Product สร้าง even ส่งไปยัง Queue ทำงานไดยไม่สนใจ Consumer ว่าจะทำงานทันหรือไม่ทัน

```
[QUEUE] Recv, val: 9
[QUEUE] Recv, val: 10
[QUEUE] Recv, val: 11
[QUEUE] Recv, val: 12
[QUEUE] Recv, val: 13
[QUEUE] Recv, val: 14
[QUEUE] Recv, val: 15
[QUEUE] Recv, val: 16
[QUEUE] Recv, val: 17
[QUEUE] Recv, val: 18
[QUEUE] Recv, val: 19
[QUEUE] Recv, val: 20
[QUEUE] Recv, val: 21
[QUEUE] Recv, val: 22
[QUEUE] Recv, val: 23
[QUEUE] Recv, val: 24
[QUEUE] Recv, val: 25
[QUEUE] Recv, val: 26
[QUEUE] Recv, val: 27
[QUEUE] Recv, val: 28
[QUEUE] Recv, val: 29
[QUEUE] Recv, val: 30
[QUEUE] Recv, val: 31
```

13/9/66

Resource Management

1. โค้ดส่วนต้นเพื่อใช้ mutex ในการจัดการทรัพยากร (resource; ในที่นี้คือการแสดงข้อความออกทางหน้าจอ)

```c
#include <stdio.h>
#include <stdlib.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"

SemaphoreHandle_t xMutex = NULL;
const TickType_t xMaxBlockTimeTicks = 0x20;

static void vPrintString(const char *pcString)
{
  //take a semaphore for useing resource (only 1 task to print string to display)
  xSemaphoreTake( xMutex, portMAX_DELAY );

  printf("%s", pcString);
  fflush( stdout );

  //Leave resource back for another task
  xSemaphoreGive( xMutex );
}

static void prvPrintTask(void *pvParameter)
{
  char *pcString = (char *)pvParameter;
  const TickType_t xSlowDownDelay = pdMS_TO_TICKS( 5UL );

  while(1)
  {
    vPrintString( pcString );
    vTaskDelay( rand() % xMaxBlockTimeTicks );
    vTaskDelay( xSlowDownDelay );
  }
}
```

ฟังก์ชัน vPrintString จะเป็นฟังก์ชันที่มีการใช้งานทรัพยากร (แสดงข้อความออกทางหน้าจอ) จึงมีการ lock mutex ไว้เพื่อป้องกัน การเรียกใช้งานซ้ำก่อนที่การ print จะเสร็จสิ้น

ส่วนฟังก์ชัน prvPrintTask จะเป็น task ที่คอยเรียกใช้งานทรัพยากร โดยในที่นี้ในแต่ละครั้งการทำงาน เวลาที่ถูกหน่วงจะถูกสุ่มขึ้นมาทำให้ การทำงานในแต่ละรอบของ task ไม่เท่ากัน ก่อให้เกิดบางช่วงเวลา task ที่มี priority ต่ำได้ใช้งานทรัพยากร ก่อให้เกิดการลด priority ของ task ที่สูงกว่าได้

2. โค้ดในส่วนของการสร้าง task ขึ้นมาจำนวน 2 task เพื่อร้องขอทรัพยากรในการแสดงข้อความออกทางหน้าจอ

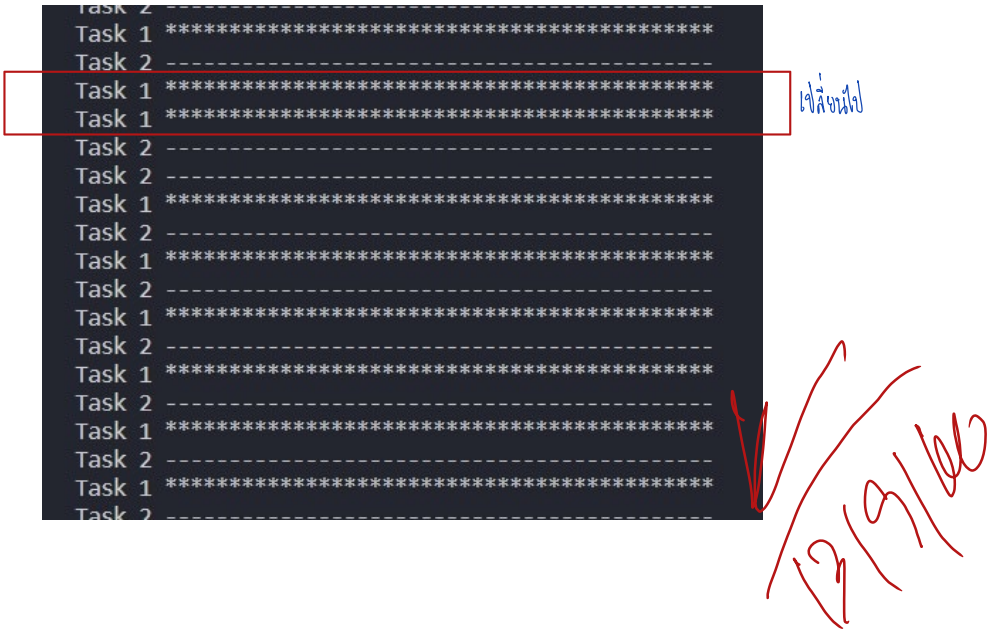
```c
const char *pvStringForTask1 = "Task 1 ***********************************************\r\n";
const char *pvStringForTask2 = "Task 2 -----------------------------------------------\r\n";

void app_main(void)
{
  xMutex = xSemaphoreCreateMutex();

  xTaskCreate(prvPrintTask, "Task 1", 2048, (void*)pvStringForTask1, 1, NULL);
  xTaskCreate(prvPrintTask, "Task 2", 2048, (void*)pvStringForTask2, 2, NULL);

  while(1)
  {
    vTaskDelay( 1000 / portTICK_PERIOD_MS );
  }
}
```

3. ทำการ build และบันทึกผลจากการ monitor ในช่วงที่การแสดงข้อความที่แตกต่างออกไปจากปกติ



ใบงานท้ายการทดลอง

ให้นิสิตเขียนโปรแกรมโดยสร้าง task ขึ้นมาดังนี้

Task 1 (Producer):

- การทำงานจะวนรอบทุกๆ 0.5 วินาที
- อ่านการกดปุ่มจากผู้ใช้งาน หากมีการกดปุ่ม ให้ทำการเพิ่มค่าในตัวแปรทีละ 1 ค่า (ตามรอบการวนของ task)
- เมื่อผู้ใช้ปล่อยปุ่มให้ส่งค่าไปให้กับ task ที่ 2 จากนั้นจึง clear ค่ากลับ

Task 2 (consumer):

- ไม่มีการหน่วงเวลาในการทำงานของ task (no delay)
- รับค่าที่ได้จาก task ที่ 1 เอามาแสดงผลที่หน้าจอ

ใน idle task ให้นิสิตทำการ delay ไว้ที่ 1 วินาทีป้องกันตัว wdt จะเกิด error

กดปุ่ม

```
Producer: BUTTON: 0
Producer: BUTTON: 0
Producer: BUTTON: 1
Consumer: Button pressed: 1

Producer: BUTTON: 2
Consumer: Button pressed: 2
```

ปล่อยปุ่ม

```
Producer: BUTTON: 3
Consumer: Button pressed: 3

Producer: BUTTON: 0
Producer: BUTTON: 0
```