01076001
วิศวกรรมคอมพิวเตอร์เบื้องต้น
Introduction to Computer Engineering

Arduino #9

Realtime OS #2

FreeRTOS : xTaskCreate

- ฟังก์ชัน xTaskCreate

```
BaseType_t xTaskCreate(     TaskFunction_t pvTaskCode,
                            const char * const pcName,
                            configSTACK_DEPTH_TYPE usStackDepth,
                            void *pvParameters,
                            UBaseType_t uxPriority,
                            TaskHandle_t *pxCreatedTask
                          );
```

xTaskCreate มีพารามิเตอร์ ดังนี้

- ตัวที่ 1 ตำแหน่งฟังก์ชันที่จะเรียกขึ้นมาทำงานเป็น Task หนึ่งในระบบ
- ตัวที่ 2 เป็นชื่อเรียกของ Task จะเห็นว่ามีลักษณะเป็นข้อความ
- ตัวที่ 3 เป็นขนาดของ Stack ที่จะจองให้ Task นั้นใช้งาน
- ตัวที่ 4 เป็นพารามิเตอร์ที่จะส่งเข้าไปใน Task
- ตัวที่ 5 เป็น Priority หรือลำดับความสำคัญของงาน ซึ่งสามารถจะกำหนดให้แต่ละงานมีลำดับความสำคัญไม่เท่ากันก็ได้

FreeRTOS : Task Handle

- pxCreatedTask เป็นพารามิเตอร์ตัวสุดท้ายของ xTaskCreate โดย *pxCreatedTask เป็นพารามิเตอร์แบบส่งกลับ คือ เมื่อเรียกใช้ฟังก์ชัน xTaskCreate แล้วฟังก์ชันจะส่งกลับข้อมูลชุดหนึ่ง ที่ทำหน้าที่เป็นตัวชี้ไปยัง Task เอง (Pointer) โดยจะต้องกำหนดตัวแปรขึ้นมารับดังนี้

```
TaskHandle_t blueHandle;
```

- โดย TaskHandle_t นั้นเป็นชนิดข้อมูล (Data Type) ของตัวแปรที่ใช้รับ Pointer ที่ชี้ไปยังแต่ละ Task โดยการใช้ตัวแปรที่จะรับ Pointer มีตัวอย่างการใช้ดังนี้

```
xTaskCreate(blueLedTask, "BLUE LED Task", 128, NULL, 1, &blueHandle);
```

FreeRTOS : Task Handle

- หลังจากที่เรียกฟังก์ชัน xTaskCreate แล้ว ฟังก์ชันจะส่งคืน Pointer ที่ชี้ไปยัง Task blueLedTask คืนมาให้ ซึ่งสามารถใช้ตัวแปร Pointer นี้ในการอ้างอิง blueLedTask ได้ เช่น หากเราจะเปลี่ยนค่า Priority ของ blueLedTask เราสามารถอ้างถึง Task ได้ดังนี้

```
vTaskPrioritySet(blueHandle,2);
```

- โดยฟังก์ชัน vTaskPrioritySet จะใช้ในการกำหนดค่า Priority ของ Task
- เมื่อเรียกใช้ฟังก์ชันดังกล่าว ค่า Priority ของ blueLedTask จะเปลี่ยนเป็น 2 และหากต้องการตรวจสอบค่า Priority ของ Task สามารถใช้ฟังก์ชัน

```
blueTaskPriority = uxTaskPriorityGet(blue_Handle);
```

FreeRTOS : Example 1

- ตัวอย่างนี้จะมีการสร้าง Task1 ด้วย Priority 1
- Task 1 จะสร้าง Task 2 ด้วย Priority 2 ทำให้ Task 2 ได้รับ CPU Time ไปทั้งหมด โดยมีการใช้ Task Handle โดยเก็บไว้ที่ Global Variable

```
#include <Arduino_FreeRTOS.h>
#include "task.h"

TaskHandle_t TaskHandle_2; // handler for Task2

void setup()
{
  Serial.begin(9600); // Enable serial communication library.
  pinMode(7, OUTPUT);  // define LED1 pin as a digital output
  pinMode(8, OUTPUT);  // define LED2 pin as a digital output

  xTaskCreate(Task1, "LED1", 100, NULL, 1, NULL);
}

void loop() {}
```

FreeRTOS : Example 1

```
void Task1(void* pvParameters)
{
  while(1)
  {
    Serial.println("Task1 Running");
    digitalWrite(7, HIGH); // sets the digital pin 7 off
    digitalWrite(8, LOW); // sets the digital pin 8 on
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    xTaskCreate(Task2, "LED2", 100, NULL, 2, &TaskHandle_2);
  }
}

void Task2(void* pvParameters)
{
    digitalWrite(7, LOW); // sets the digital pin 7 on
    digitalWrite(8, HIGH); // sets the digital pin 8 off
    Serial.println("Task2 is runnig and about to delete itself");
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    vTaskDelete(TaskHandle_2);
}
```

FreeRTOS : Example 2

- ตัวอย่างนี้จะมีการสร้าง Task1 ด้วย Priority 3 และ Task2 ด้วย Priority 2 ซึ่งทำให้ Task1 มีความสำคัญสูงกว่า
- จากนั้น ใน Task 1 มีการปรับค่า Priority ของ Task 2 + 1 ทำให้มี Priority 3 เท่ากัน ทำให้ Task 2 มีโอกาสทำงาน

```
#include <Arduino_FreeRTOS.h>
#include <task.h>
void Task1( void *pvParameters );
void Task2( void *pvParameters );

TaskHandle_t TaskHandle_1; // handler for Task1
TaskHandle_t TaskHandle_2; // handler for Task2

void setup()
{
  Serial.begin(9600); // Enable serial communication library.

  xTaskCreate(Task1, "LED1", 100, NULL, 3, &TaskHandle_1);
  xTaskCreate(Task2, "LED2", 100, NULL, 2, &TaskHandle_2);
}

void loop(){}
```

FreeRTOS : Example 2

```
void Task1(void* pvParameters)
{
  UBaseType_t uxPriority = uxTaskPriorityGet( NULL );
  while (1)
  {
    Serial.print("Task1 is running and about to raise Task2 Priority (");
    int TaskPriority = uxTaskPriorityGet(TaskHandle_2);
    Serial.print(TaskPriority);
    Serial.println(")");
    vTaskPrioritySet( TaskHandle_2, ( uxPriority + 1 ) );
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
  }
}

void Task2(void* pvParameters)
{
  UBaseType_t   uxPriority = uxTaskPriorityGet( NULL );
  while (1)
  {
    Serial.print("Task2 is running and about to lower Task2 Priority (");
    int TaskPriority = uxTaskPriorityGet(TaskHandle_2);
    Serial.print(TaskPriority);
    Serial.println(")");
    vTaskPrioritySet( TaskHandle_2, ( uxPriority - 2 ) );
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
  }
}
```

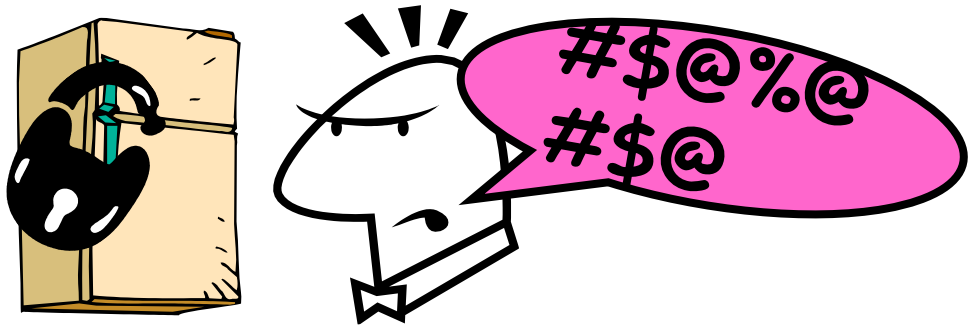
Analogy : Too Much Milk

- สมมติสถานการณ์ : A และ B อยู่บ้านเดียวกัน โดยมีข้อตกลงว่า ให้มีนมในตู้เย็นไว้กิน ถ้าใครเห็นตู้เย็นว่าง ให้ซื้อนมมาด้วย

Time	Person A	Person B
3:00	Look in Fridge. Out of milk	
3:05	Leave for store	
3:10	Arrive at store	Look in Fridge. Out of milk
3:15	Buy milk	Leave for store
3:20	Arrive home, put milk away	Arrive at store
3:25		Buy milk
3:30		Arrive home, put milk away

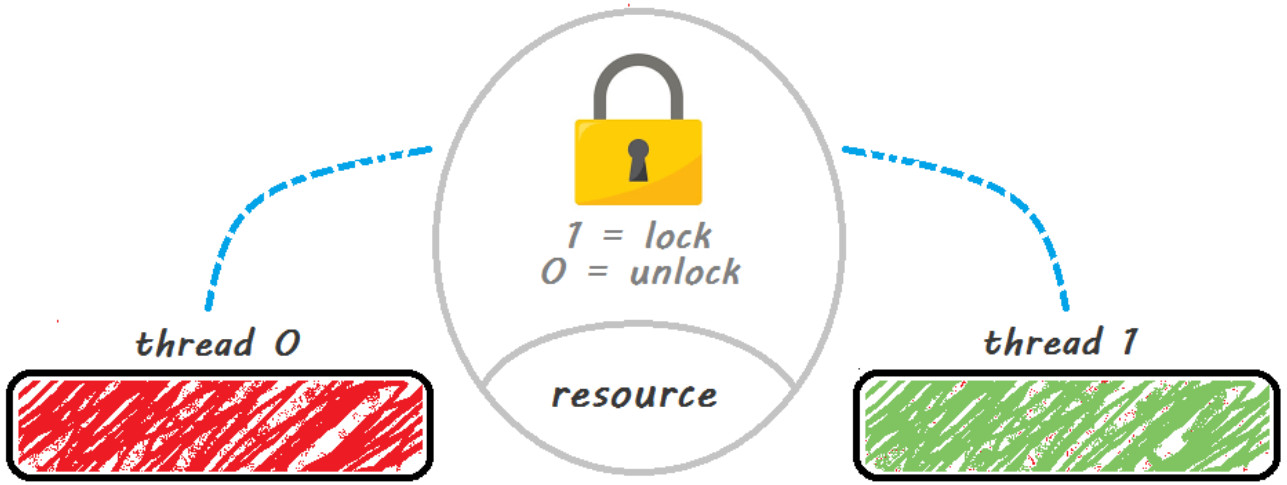
FreeRTOS : Synchronization

- ปัญหาที่ยกตัวอย่าง เป็นปัญหาของ Synchronization ซึ่งในระบบปฏิบัติการก็มีปัญหานี้เช่นกัน ในกรณีที่มีการใช้ทรัพยากรร่วมกัน
- ตัวอย่าง คือ มี LED 1 ดวง โดยให้ Task #1 ON ถ้า OFF และ Task #2 OFF ถ้า ON
- การแก้ปัญหานี้ ต้องใช้สิ่งที่เรียกว่า Lock โดยทำหน้าที่ป้องกันไม่ให้คนอื่นทำการกับตำแหน่งที่กำลังใช้งาน
- กรณีโจทย์ Too Much Milk ก็อาจจะ Lock ตู้เย็นในระหว่างที่ออกไปซื้อนม เป็นต้น



FreeRTOS : Semaphore

- กรณี ตัวอย่าง คือ มี LED 1 ดวง โดยให้ Task #1 ON ถ้า OFF และ Task #2 OFF ถ้า ON ต้องมีการ lock ทรัพยากรก่อนจะสั่งให้ติดหรือดับ
- กรณีนี้เราจะใช้เครื่องมือที่เรียกว่า Semaphore เพื่อให้เวลาใดๆ มีเพียง 1 Task ที่สามารถใช้งาน LED ได้



FreeRTOS : Semaphore

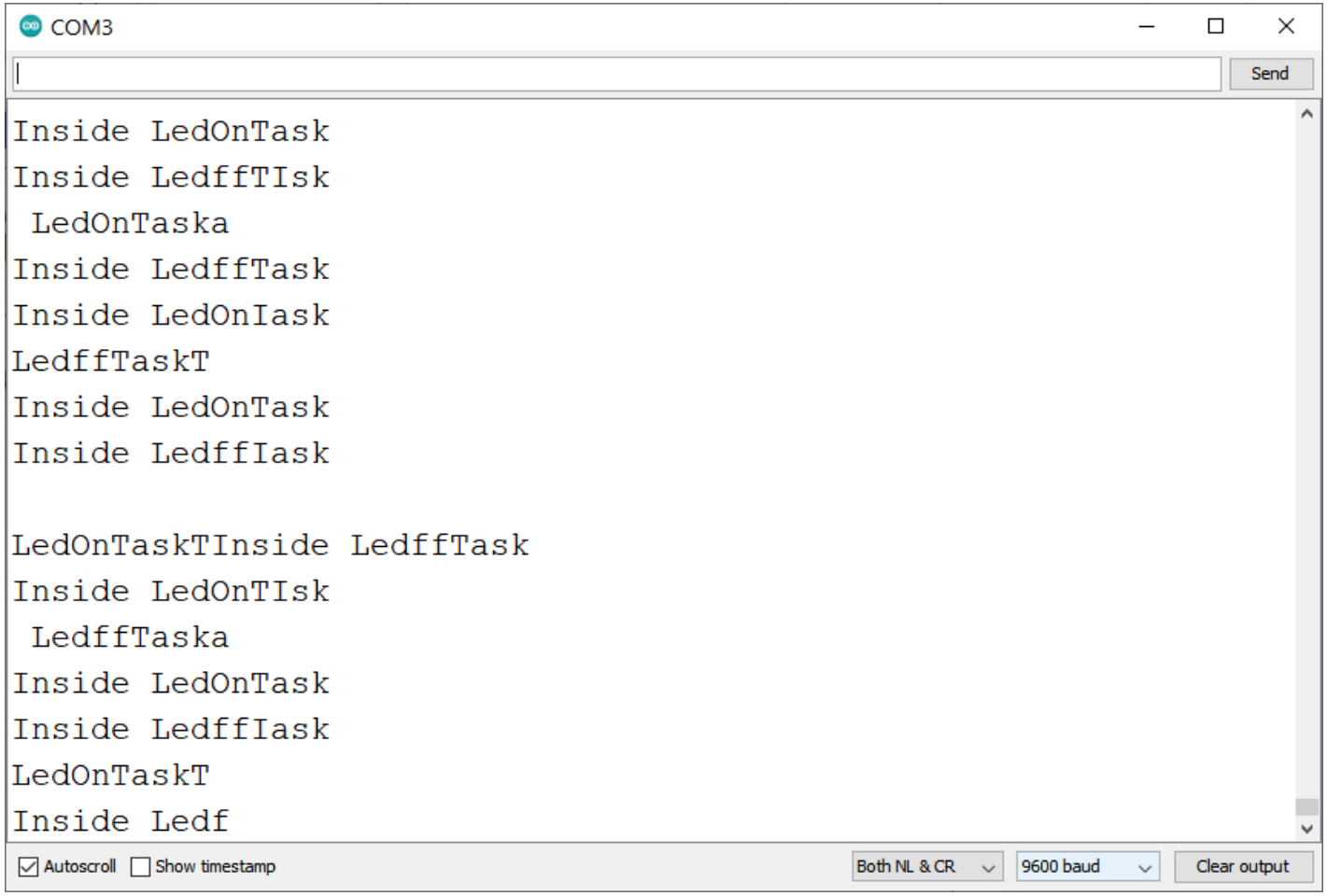
```
#include <Arduino_FreeRTOS.h>
#include "semphr.h"
#define  LED  7
SemaphoreHandle_t xBinarySemaphore;
void setup()
{
  Serial.begin(9600);
  pinMode(LED ,OUTPUT);
  xBinarySemaphore = xSemaphoreCreateBinary();
  xTaskCreate(LedOnTask, "LedON",100,NULL,1,NULL);
  xTaskCreate(LedoffTask, "LedOFF", 100,NULL,1,NULL);
  xSemaphoreGive(xBinarySemaphore);
}

void loop(){}
```

FreeRTOS : Semaphore

```
void LedOnTask(void *pvParameters)
{
  while(1)
  {
   xSemaphoreTake(xBinarySemaphore,portMAX_DELAY);
   Serial.println("Inside LedOnTask");
   digitalWrite(LED,LOW);
   xSemaphoreGive(xBinarySemaphore);
   vTaskDelay(1);
  }
}
void LedoffTask(void *pvParameters)
{
  while(1)
  {
    xSemaphoreTake(xBinarySemaphore,portMAX_DELAY);
    Serial.println("Inside LedffTask");
    digitalWrite(LED,HIGH);
    xSemaphoreGive(xBinarySemaphore);
    vTaskDelay(1);
  }
}
```

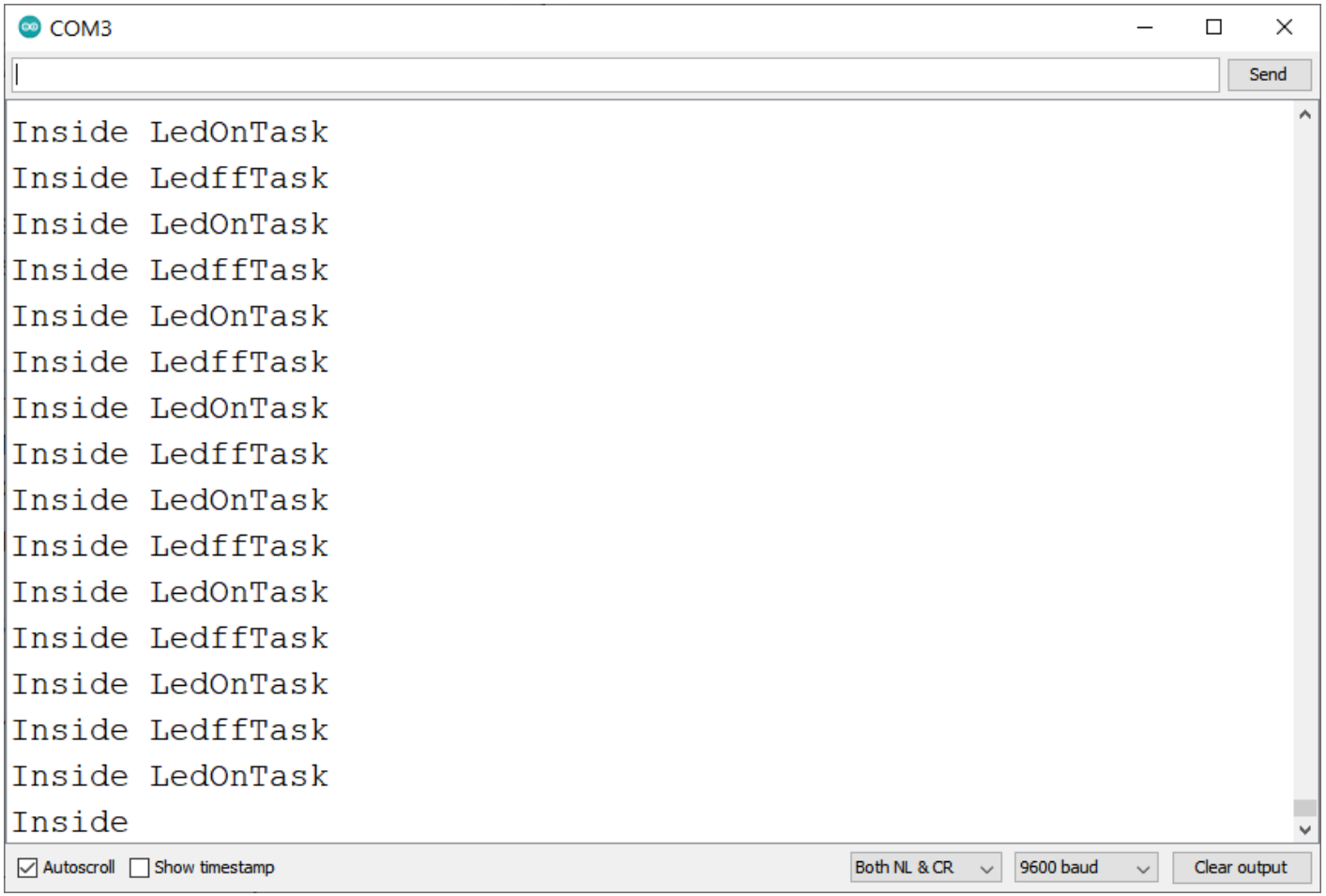
FreeRTOS : Semaphore

- ไม่ใช้ Semaphore



FreeRTOS : Semaphore

- ใช้ Semaphore



FreeRTOS : Semaphore

- จะเห็นว่า กรณีที่ไม่ใช่ semaphore การ switch task อาจเกิดขึ้นในเวลาใดก็ได้ ทำให้การแสดงผลไม่เป็นไปตามลำดับ
- แต่เมื่อมีการใส่ semaphore การ switch task จะเกิดขึ้นได้ ก่อน หรือ หลัง การ Lock เท่านั้น
  - กรณีที่ Task 1 มีการเรียกใช้ xSemaphoreTake ซึ่งทำให้ Task 1 ได้ semaphore ไป เมื่อ Task 2 ต้องการ semaphore จึงต้องรอ ทำให้ Task 2 ไปอยู่ในสถานะ Block เมื่อ Task 1 มีการเรียกใช้ xSemaphoreGive จึงทำให้ semaphore ว่าง
  - จากนั้น Task 2 จึงหลุดออกจากสถานะ Block และเรียกใช้ xSemaphoreTake ซึ่งทำให้ Task 2 ได้ semaphore ไป เมื่อถึงรอบการทำงานของ Task 1 เมื่อมีการร้องขอ semaphore จึงไม่ได้ และเข้าสู่สถานะ Block จนกว่า Task 2 จะปล่อย semaphore
  - จึงทำให้การแสดงผลเป็นไปตามลำดับ เพราะจะรันได้ทีละ 1 Task ทั้งช่วงการทำงาน

FreeRTOS : Semaphore - LED sequence

- ขอยกอีกตัวอย่าง โดยกำหนด LED จำนวน 8 ดวง โดยกำหนดให้แต่ละดวง ควบคุมโดย 1 Task และมีเงื่อนไข คือ ให้แสดงผลเรียงตามลำดับ จาก 1-8

FreeRTOS : Semaphore - LED sequence 1

```
#include <Arduino_FreeRTOS.h> // tested on Uno
#include <task.h>
#include <semphr.h>

#define LED_ON     LOW
#define LED_OFF    HIGH
#define NUM_LEDS   (8)
const byte LED_PINS[] = { 5,6,7,8,9,10,11,12 };

TaskHandle_t taskHandles[ NUM_LEDS ];
SemaphoreHandle_t  semaphores[ NUM_LEDS ];

void task( void *pvParameters ); // task function prototype

void setup() {
  randomSeed( analogRead(0) );
  char sbuf[4];  int  priority = (tskIDLE_PRIORITY + 2);
  for ( int id=0; id < NUM_LEDS; id++ ) {
     semaphores[id] = xSemaphoreCreateBinary();
     if ( id == 0 ) {
        xSemaphoreGive( semaphores[0] );
     }
     sprintf( sbuf, "T%d", id );
     xTaskCreate( task, (const char *)sbuf, 60, (void *)id, priority,
&taskHandles[id] );
}
```

FreeRTOS : Semaphore - LED sequence 1

```
void loop() {} // do nothings

void task( void *pvParameters ) {
   int id = (int)pvParameters;
   int ledPin = LED_PINS[id];
   pinMode( ledPin, OUTPUT );
   digitalWrite( ledPin, LED_OFF );
   vTaskDelay( pdMS_TO_TICKS( random(100,1000) ) );
   while(1) {
      if (xSemaphoreTake( semaphores[id], portMAX_DELAY )==pdTRUE){
         digitalWrite( ledPin, LED_ON );
         vTaskDelay( pdMS_TO_TICKS(100) );
         digitalWrite( ledPin, LED_OFF );
         xSemaphoreGive( semaphores[ (id+1) % NUM_LEDS ] );
         taskYIELD();
      }
   }
}
```

FreeRTOS : Semaphore - LED sequence 2

```
#include <Arduino_FreeRTOS.h>
#include <task.h>
#include <semphr.h>

#define LED_ON      LOW
#define LED_OFF     HIGH
#define NUM_LEDS    (8)
const byte LED_PINS[] = { 5,6,7,8,9,10,11,12 };

TaskHandle_t        taskHandles[ NUM_LEDS ];
SemaphoreHandle_t   semaphores[ NUM_LEDS ];// task function prototypes

void task( void *pvParameters );
void sequencer(  void *pvParameters );

void setup() {
  char sbuf[4]; int  priority = (tskIDLE_PRIORITY + 2);
  for ( int id=0; id < NUM_LEDS; id++ ) {
     semaphores[id] = xSemaphoreCreateBinary();
     sprintf( sbuf, "T%d", id );
     xTaskCreate(task, (const char *)sbuf, 60, (void *)id, priority, &taskHandles[id] );
  }
  xTaskCreate( sequencer, "sequencer", 60, NULL, priority, NULL );
}
void loop() {} // do nothings
```

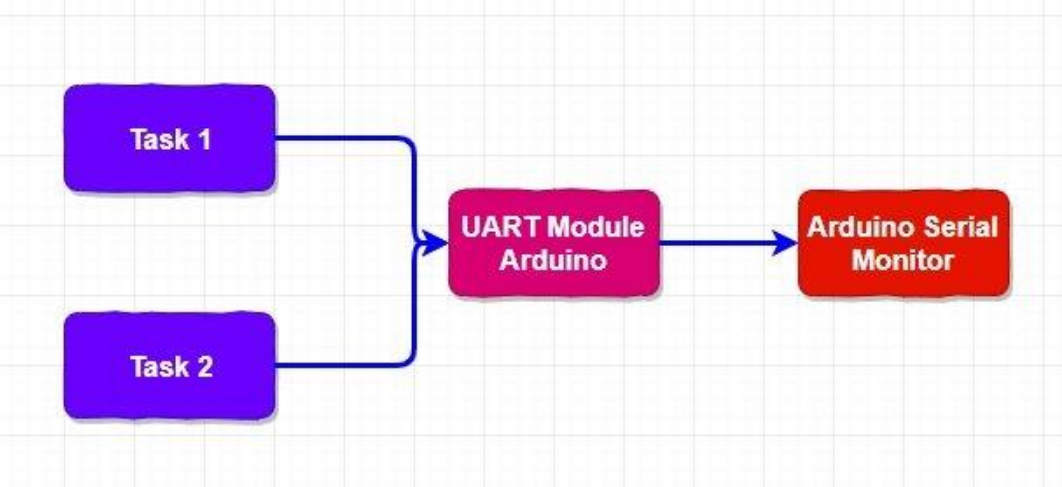
FreeRTOS : Semaphore - LED sequence 2

```
void sequencer( void *pvParameters ) {
  int index = 0;
  while (1) {
      xSemaphoreGive( semaphores[ index ] );
      index = (index+1) % NUM_LEDS;
      vTaskDelay( pdMS_TO_TICKS(100) );
  }
}
```

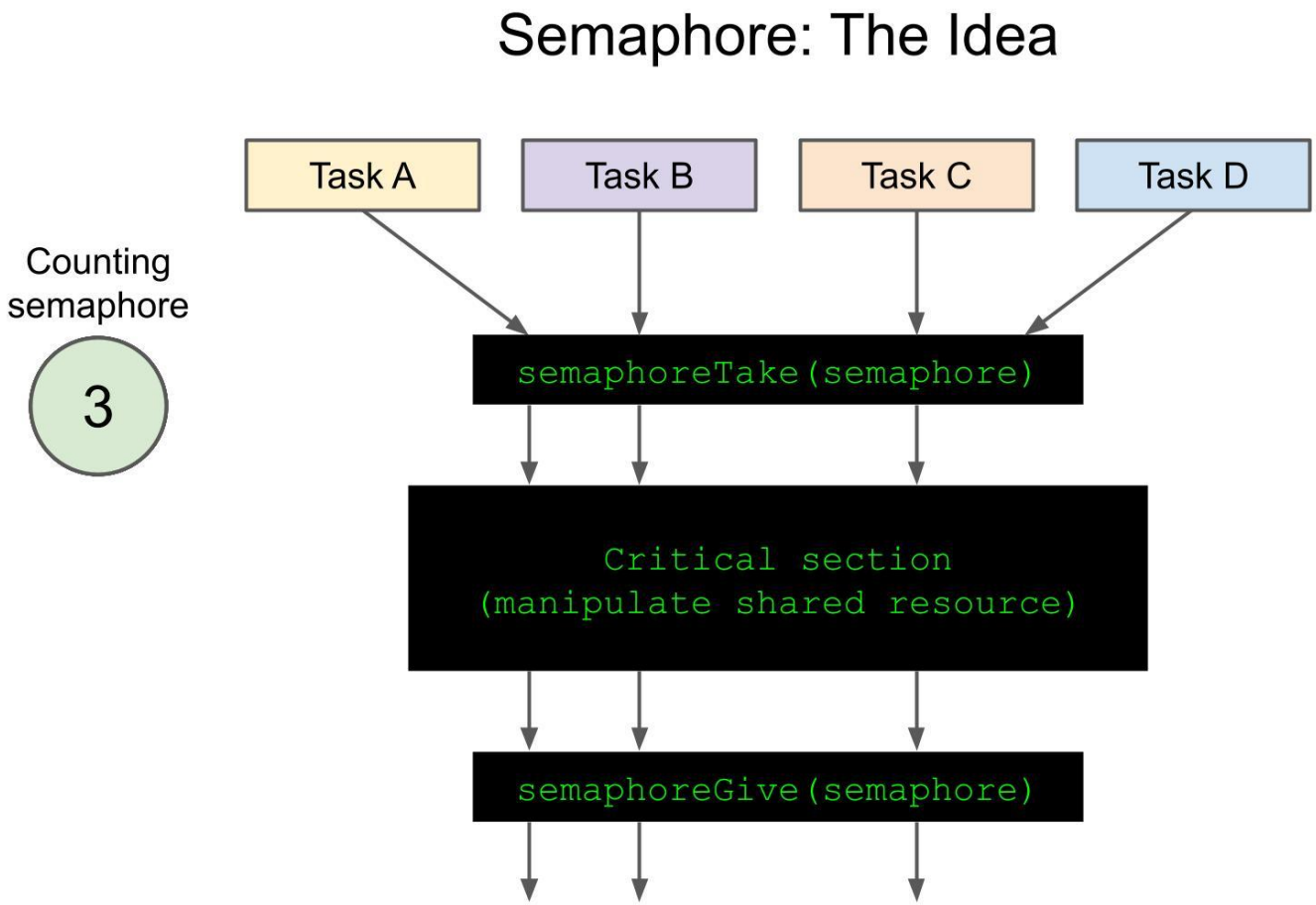
```
void task( void *pvParameters ) {
   int id = (int)pvParameters;
   int ledPin = LED_PINS[id];
   pinMode( ledPin, OUTPUT );
   digitalWrite( ledPin, LED_OFF );
   while(1) {
      if (xSemaphoreTake( semaphores[id], portMAX_DELAY )==pdTRUE){
         digitalWrite( ledPin, LED_ON );
         vTaskDelay( pdMS_TO_TICKS(100) );
         digitalWrite( ledPin, LED_OFF );
         taskYIELD();
      }
   }
}
```

FreeRTOS : Semaphore

- Semaphore ที่ผ่านไปเรียกว่า Binary Semaphore เนื่องจากมีสถานะเพียง Lock กับไม่ Lock แต่มี Semaphore อีกแบบ เรียกว่า Counting Semaphore โดยมีความสามารถในการนับจำนวนครั้งของการ Lock
  - อาจใช้ในการนับจำนวน event โดยทำหน้าที่กำหนดจำนวน event ที่เกิดขึ้นได้
  - ใช้นับจำนวนทรัพยากรที่มีให้ใช้งาน เมื่อครบแล้ว ก็ไม่มีใครสามารถใช้ทรัพยากรนั้นได้ระหว่างที่มีการใช้งาน



FreeRTOS : Semaphore



FreeRTOS : Semaphore

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>  // add the FreeRTOS functions for Semaphores (or Flags).

SemaphoreHandle_t xSerialSemaphore;

void TaskDigitalRead( void *pvParameters );
void TaskAnalogRead( void *pvParameters );

void setup() {

  Serial.begin(9600);

  if ( xSerialSemaphore == NULL )
  {
    xSerialSemaphore = xSemaphoreCreateCounting(1, 1);
    if ( ( xSerialSemaphore ) != NULL )
      xSemaphoreGive( ( xSerialSemaphore ) );
  }
  xTaskCreate(TaskDigitalRead, "DigitalRead", 128, NULL, 2, NULL);
  xTaskCreate(TaskAnalogRead, "AnalogRead", 128, NULL, 1, NULL);
}

void loop() {}
```

FreeRTOS : Semaphore

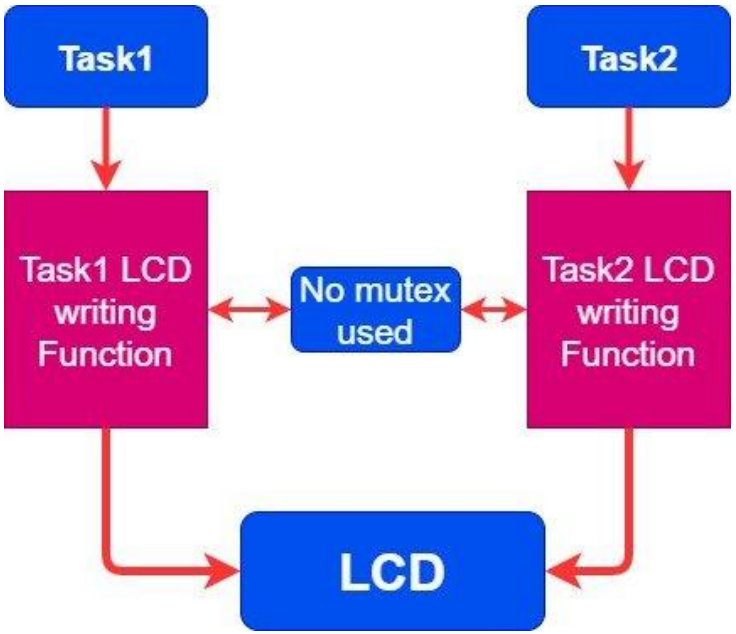
```
void TaskDigitalRead( void *pvParameters __attribute__((unused)) )
{
  uint8_t pushButton = 2;
  pinMode(pushButton, INPUT);
  for (;;)
  {
    int buttonState = digitalRead(pushButton);

    // If the semaphore is not available, wait 5 ticks
    if (xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
    {
      Serial.println(buttonState);
      xSemaphoreGive( xSerialSemaphore );
    }
    vTaskDelay(1);
  }
}
```

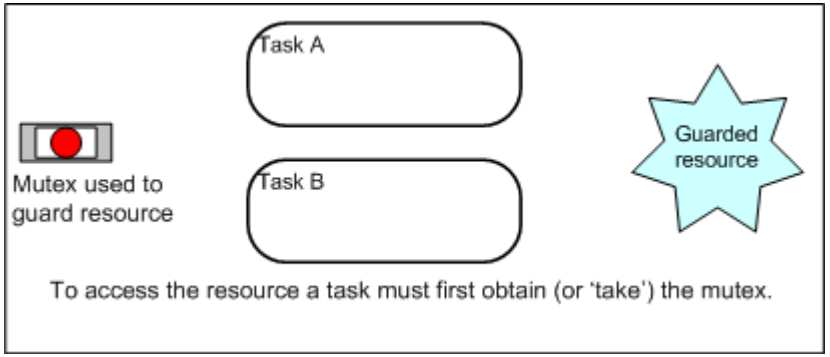
FreeRTOS : Semaphore

```
void TaskAnalogRead( void *pvParameters __attribute__((unused)) )
{
  for (;;)
  {
    int sensorValue = analogRead(A0);

    // If the semaphore is not available, wait 5 ticks
    if ( xSemaphoreTake(xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
    {
      Serial.println(sensorValue);
      xSemaphoreGive( xSerialSemaphore );
    }

    vTaskDelay(1);
  }
}
```

FreeRTOS : Mutual Exclusion

- ในการทำงานของ Task กรณีที่มีทรัพยากรที่ใช้ร่วมกัน จะมีการรับรองว่าไม่มีการเข้าไปใช้ทรัพยากรพร้อมกัน (Mutual Exclusion)
- ใน FreeRTOS จะมีการสร้าง Semaphore ชนิดพิเศษ โดยเป็น Binary Semaphore





FreeRTOS : Mutual Exclusion

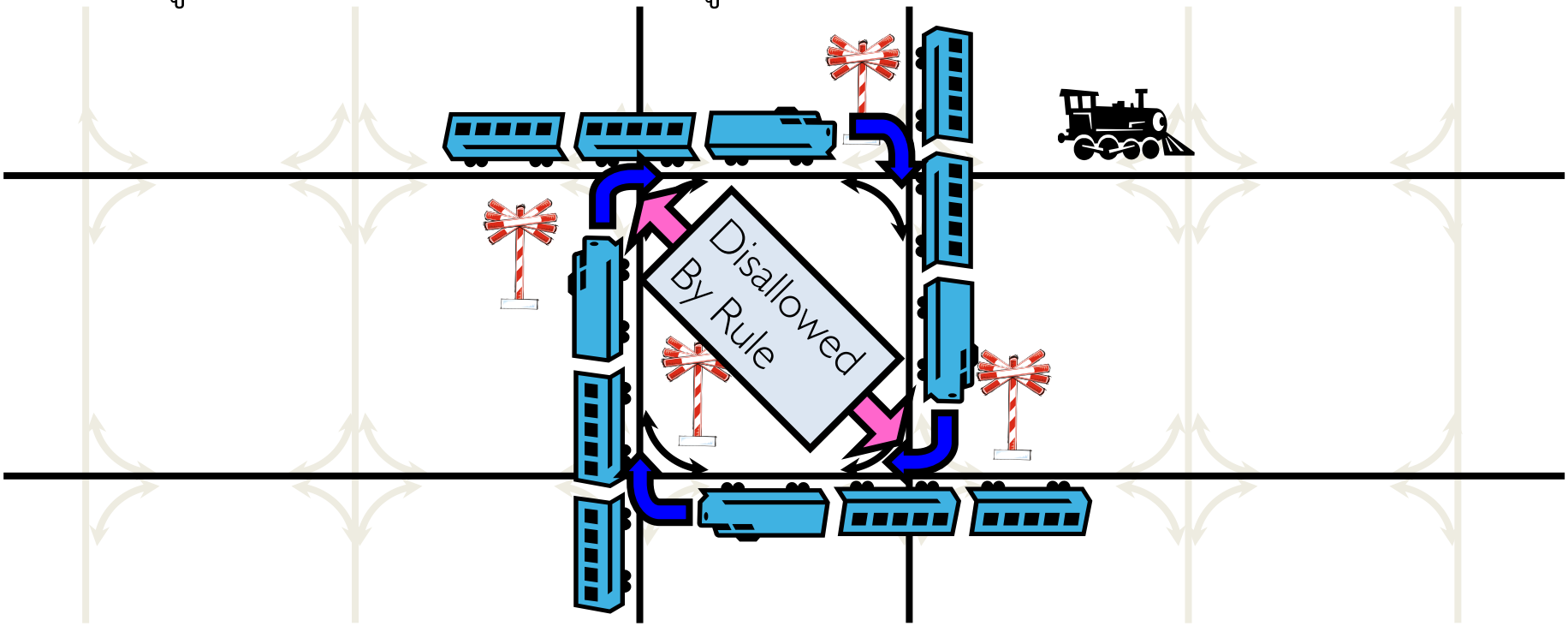
```
#include <Arduino_FreeRTOS.h>
#include "semphr.h"
//create handle for the mutex. It will be used to reference mutex
SemaphoreHandle_t  xMutex;

void setup()
{

  Serial.begin(9600);

  xMutex = xSemaphoreCreateMutex();

  xTaskCreate(OutputTask,"Printer Task 1", 100,"Task 1 \r\n",1,NULL);
  xTaskCreate(OutputTask,"Printer Task 2", 100,"Task 2 \r\n",2,NULL);
}
```

FreeRTOS : Mutual Exclusion

```
void OutputTask(void *pvParameters)
{
  char *pcStringToPrint;
  pcStringToPrint = (char *)pvParameters;
  while(1)
  {
    printer(pcStringToPrint);
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}
// this printer task send data to arduino serial monitor
//also it is shared resource between both instances of the tasks

void printer(const char* pcString)
{
  // take mutex
  xSemaphoreTake(xMutex, portMAX_DELAY);
  Serial.println(pcString); // send string to serial monitor
  xSemaphoreGive(xMutex); // release mutex
}
void loop(){}
```

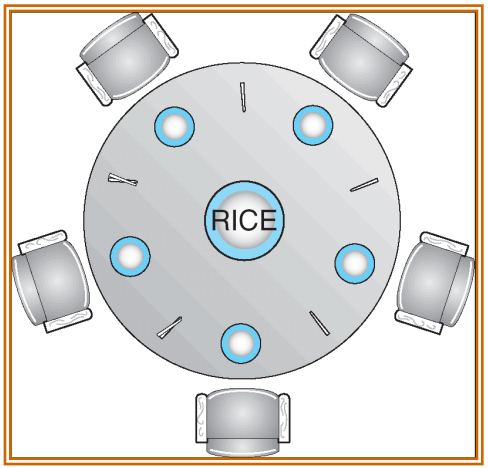
FreeRTOS : Deadlock

- กรณีที่ในแต่ละ Task ต้องการใช้ทรัพยากรมากกว่า 2 ตัว อาจเกิดเหตุการณ์ที่แต่ละ Task เป็นเจ้าของทรัพยากรอยู่ก่อนแล้ว 1 ตัว และ ต้องการจองอีก 1 ตัว เหตุการณ์นี้เรียกว่า Deadlock
- จากรูป รถไฟต้องการเลี้ยวขวา และ ถูกขวางโดยรถไฟคันอื่น (จะแก้อย่างไร?)



FreeRTOS : Deadlock

- ปัญหา : Dining Philosophers เป็นสถานการณ์ที่นำเสนอโดยนักคอมพิวเตอร์ชื่อ Edsger Dijkstra ในปีค.ศ. 1965 เพื่อยกตัวอย่างปัญหาเกี่ยวกับการประสานการทำงานของ Task
- ปัญหานี้กล่าวถึง นักปราชญ์ เช่น 5 คน นั่งล้อมวงทานอาหารเย็น และต้องใช้ตะเกียบมีจำนวนแท่งตะเกียบเท่ากับจำนวนนักปราชญ์ โดยวางตะเกียบหนึ่งแท่ง สลับกับที่นั่งของปราชญ์ เป็นวงกลมรอบโต๊ะ



FreeRTOS : Deadlock

- นักปราชญ์แต่ละคน จะมีเวลาในการคิด (Thinking) ก่อนที่จะสลับไปทานอาหาร (Eating) โดยจะต้องหยิบแท่งตะเกียบ (Chopsticks) ที่อยู่ทางซ้ายและขวามือของตนเองตามลำดับ จากนั้นเมื่อทานแล้วโดยใช้เวลาช่วงสั้น ๆ จะต้องวางตะเกียบลงเพื่อให้คนข้าง ๆ ได้ หยิบใช้บ้าง
- ในการหยิบตะเกียบ นักปราชญ์แต่ละคนจะไม่คุยหรือสื่อสารกัน ถ้าหยิบแท่งตะเกียบทางซ้ายถือไว้แล้ว จะพยายามหยิบแท่งตะเกียบทางขวามือของตนเอง หรือรอจนกว่าคนข้าง ๆ จะวางแท่งตะเกียบ
- นักปราชญ์แต่ละคนเปรียบเสมือนทาส์ก (T0, T1, ..., T4) และแท่งตะเกียบเป็นทรัพยากรที่จะต้องแชร์กันใช้ นักปราชญ์ที่นั่งติดกันจะกินพร้อมกันไม่ได้
- ถ้ามีเหตุการณ์ที่ถือตะเกียบคนละอัน จะเกิด Deadlock

For your attention