Kernel Masters

FREE RTOS Lab Experiments

Embedded C & RTOS

Kernel Masters
2/27/2022

Contents

FREE RTOS Lab Experiments

1. Semaphore

Semaphore Introduction

In multitasking operating system, a semaphore is a variable used to control the access to a common resource by multiple processes. The value of semaphore represents the number of available resources. Semaphores are equipped with two operations: **P** and **V**.

Operation V increments the semaphore, while operation P decrements it. When operation P is executed but the semaphore value is zero, the task executing it will be blocked and wait until the semaphore value is bigger than zero.

Imaging that there is a parking lot with ten parking spaces, which means this parking lot can contain ten vehicles. We regard the parking lot as a semaphore with the value 10 and vehicles as the tasks. A vehicle come into the parking lot means a operation P, while a vehicle go out from the parking lot means a operation V. At the beginning, the parking lot is empty. Vehicles come into the parking lot, do operation P and occupy the parking spaces. When the parking lot is full, the value of semaphore is zero. If there is a vehicle wants to come into the parking lot and do the operation P, it has to wait outside until there is a vehicle goes out from the parking lot and does operation V.

Semaphores which allow an arbitrary resource count are called counting semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores and are used to implement mutex locks.

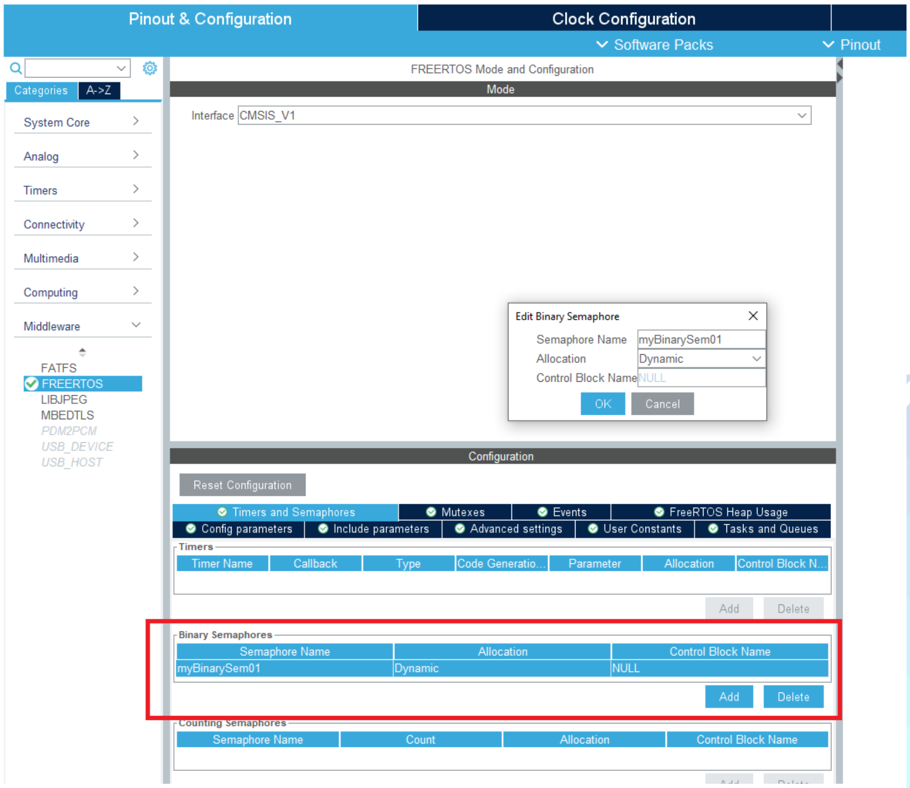
Lab Experiment 1

Write a Program to create two tasks to synchronization using binary semaphore. Task1 release a semaphore to Task2 every 100 msec delay. Task2 keep on waiting for semaphore, once semaphore is received then toggle RED & GREEN LED and again waiting for semaphore.

Name of the Binary Semaphore is "*myBinarySem01*"

Binary Semaphore Configuration on STM32CubeIDE

Go to the Timers and Semaphores tab and add binary semaphores named myBinarySem01.




```
void Task01(void const * argument)
{
 for(;;)
 {
  osDelay(100);
                osSemaphoreRelease(myBinarySem01Handle);
 }
}

void Task02(void const * argument)
{
 /* Infinite loop */
 for(;;)
 {
                osSemaphoreWait(myBinarySem01Handle, osWaitForever);//0x0000FFFF);
                HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_13);
                HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_14);
 }
```

Lab Experiment 2

Write a producer consumer problem synchronization using two binary semaphores.

a. Create producer & consumer tasks and also create "myBinaryEmpty" & "myBinaryFull" binary semaphores.
b. Producer task release a "myBinaryEmpty "semaphore once fill the buffer. And wait for "myBinaryFull" semaphore.
c. Consumer task release a "myBinaryFull" semaphore once fill the buffer. And wait for "myBinaryEmpty" semaphore.

```
void Producer(void const * argument)
{
 /* Infinite loop */
        int in=0,counter=0;
 for(;;)
 {
        osSemaphoreWait(myBinaryFullHandle, osWaitForever);
        while(1) {
        buffer[in] = ch;
        in = (in + 1)%5;

       if(in==0)
        {
           in=0;
          ch='A';
          HAL_UART_Transmit(&huart1, (uint8_t *)"Producer: ", 10,1000);
          HAL_UART_Transmit(&huart1, (uint8_t *)buffer, 5,1000);
          break;
        }
          ch++;
        }
        osSemaphoreRelease(myBinaryEmptyHandle);
        }
}


void Consumer(void const * argument)
{
 int out=0;
 for(;;)
 {
   osSemaphoreWait(myBinaryEmptyHandle, osWaitForever);
    while(1)
    {
      data[out] = buffer[out];
      out = (out + 1) % 5;
```

```
    if(out == 0)
    {
            out=5;
            HAL_UART_Transmit(&huart1, (uint8_t *)"Consumer: ", 10,1000);
            HAL_UART_Transmit(&huart1, (uint8_t *)data, 5,1000);
            break;
     } // if
   } //while
      osSemaphoreRelease(myBinaryFullHandle);
 } // for
}
```

Reference Links:

- FreeRTOS API Reference: https://www.freertos.org/a00106.html
- CMSIS-RTOS API Reference: http://www.keil.com/pack/doc/CMSIS/RTOS/html/group__CMSIS__RTOS.html