

SEMAPHORE (SEMAFOR) NEDİR?

Semaforlar, birden fazla iş parçacığının (örneğin, task, thread) aynı kaynakları (örneğin, bellek, çevresel birimler) aynı anda kullanmasını engellemek veya sınırlamak için kullanılır.

Semaforlar, iki farklı işlem yapabilir: bir task'ın beklemesini sağlamak veya bir task'ı uyandırmak. Eğer bir kaynak mevcutsa, semafor bir task'ı uyandırarak kaynağın kullanılmasına izin verir. Eğer kaynak mevcut değilse, semafor bekleyen task'ı askıya alır ve kaynak mevcut olduğunda task'ı uyandırır.

Semaforlar, task'lar arasında senkronizasyon ve veri paylaşımı gibi kritik bölgelerin yönetimi için kullanılır. Semaforlar, task'lar arasındaki iletişimi düzenlemek için de kullanılabilir. FreeRTOS gibi bir RTOS (Real-Time Operating System) de semaforlar kullanılabilir.

SEMAFOR ÇEŞİTLERİ NELERDİR?

Semaforlar, bir iş parçacığının belirli bir kaynağın kullanımını yönetmek için kullanılan araçlardır. Semaforlar, bir iş parçacığının kaynağa erişmesine izin verilip verilmeyeceğini belirleyen bir bit değeri taşırlar. Semaforlar genellikle aşağıdaki iki türde kullanılabilir:

Binary Semaphores: Binary semaforlar, bir kaynağın yalnızca bir kez kullanılabileceğini belirlemek için kullanılır. Binary semaforlar sadece 0 veya 1 değerini alabilirler ve bir iş parçacığı bu kaynağı kullanırken semaforun değerini 1 yapar ve bu kaynağı bıraktığında semaforun değerini tekrar 0 yapar.

Counting Semaphores: Counting semaforlar, birden fazla iş parçacığının aynı kaynağı kullanmasına izin verilen durumlar için kullanılır. Bu semaforlar belirli bir değer aralığında sayı değerlerini taşırlar ve bir iş parçacığı kaynağı kullandığında semaforun değerini bir azaltır ve kaynağı bıraktığında ise semaforun değerini bir arttırır.

Her iki semafor tipi de bir iş parçacığının kaynakları nasıl kullanacağını belirlemek için kullanılabilir. Binary semaforlar yalnızca bir iş parçacığının aynı anda kaynağı kullanmasını önlemek için kullanılabilirken, counting semaforlar birden fazla iş parçacığının aynı kaynağı kullanmasına izin verilen durumlarda kullanılabilir.

MUTUAL EXCLUSION NEDİR?

Mutual exclusion, bir sistemdeki bir veya daha fazla iş parçacığının aynı zamanda aynı kaynağa erişmesini önlemek için kullanılan bir terimdir. Mutual exclusion, bir kaynak için yalnızca bir iş parçacığının erişimine izin verir. Bu sayede kaynakta yapılan değişikliklerin birbirine zarar vermesi önlenabilir. Örneğin, bir bellek alanında yapılan değişikliklerin başka bir iş parçacığı tarafından fark edilmemesi, bu alanı kullanan iş parçacıkları arasında uyumsuzluk oluşturabilir. Mutual exclusion, bu tip problemlerin önüne geçmek için kullanılabilir.

FreeRTOS' DA SEMAFOR PROJESİ

PROJENİN AMACI:

Bu projede 2 adet görev olacak Task1 ve Task2, Task2 daha büyük öncelikli iken, Task1 daha küçük öncelikli olacak. (Task2 > Task1)

Task1' Semafor anahtarını verirken Task2 ise semafor anahtarını alacak.

Task1 Anahtarı vermeden önce c1 adlı değişkenin içindeki sayıyı 1 artıracak.

Task2 ise her anahtarı aldığı anda c2 adlı değişkenin içindeki sayıyı 1 artıracak.

PROJE AYARLARI:

Öncelikle FreeRTOS' u STM32CubeIDE' den aktifleştirip ardından Task1 ve Task2' yi tanımlıyoruz task2' nin önceliği task1' den fazla olacak şekilde ayarlıyoruz.

The image shows two side-by-side screenshots of the 'Edit Task' dialog box in STM32CubeIDE. The left dialog is for 'task1' and the right is for 'task2'. Both dialogs have the following fields: Task Name, Priority (osPriorityNormal for task1, osPriorityHigh for task2), Stack Size (Words) (128), Entry Function (task1_func for task1, task2_func for task2), Code Generation Option (Default), Parameter (NULL), Allocation (Dynamic), Buffer Name (NULL), and Control Block Name (NULL). Each dialog has 'OK' and 'Cancel' buttons at the bottom.

Ardından İkili Semaforu (Binary Semaphore) oluşturuyoruz.

The image shows a screenshot of the 'Edit Binary Semaphore' dialog box in STM32CubeIDE. The dialog has the following fields: Semaphore Name (Semaphore), Allocation (Dynamic), and Control Block Name (NULL). There are 'OK' and 'Cancel' buttons at the bottom.

Şimdi CTRL + S basıp otomatik olarak kodlarımızın oluşturulmasını bekliyoruz.

CMSIS_V1 Kütüphanesinin API' lerine bağımlı kalmadan semaforu oluşturmak için yorum satırında olan xSemaphoreCreateBinary() API' sini kullanabiliriz.

```
/* Create the semaphores(s) */
/* definition and creation of Semaphore */
osSemaphoreDef(Semaphore);
SemaphoreHandle = osSemaphoreCreate(osSemaphore(Semaphore), 1);
//SemaphoreHandle = xSemaphoreCreateBinary();
```

c1 ve c2 adında 2 adet değişken oluşturuyoruz

```
/* USER CODE BEGIN 4 */
uint32_t c1 = 0;
uint32_t c2 = 0;
/* USER CODE END 4 */
```

Şimdi ise task1 ve task2 görevlerinin fonksiyonlarını aşağıdaki gibi dolduruyoruz.

```
/* USER CODE END Header_task1_func */
void task1_func(void const * argument)
{
    /* USER CODE BEGIN 5 */
    /* Infinite loop */
    for(;;)
    {
        c1++;
        osSemaphoreRelease(SemaphoreHandle);
        //xSemaphoreGive(SemaphoreHandle);
        osDelay(1000);
    }
    /* USER CODE END 5 */
}
/* USER CODE END Header_task2_func */
void task2_func(void const * argument)
{
    /* USER CODE BEGIN task2_func */
    /* Infinite loop */
    for(;;)
    {
        //if(xSemaphoreTake(SemaphoreHandle,pdMS_TO_TICKS(portMAX_DELAY)) == pdTRUE)
        if(osSemaphoreWait(SemaphoreHandle, osWaitForever) == osOK)
        {
            c2++;
        }
    }
    /* USER CODE END task2_func */
}
```

NOT:

task1_func()' da bulunan **osSemaphoreRelease()** API' si yerine **xSemaphoreGive()** API' sini **task2_func()**' da bulunan **osSemaphoreWait()** API' si yerine **xSemaphoreTake()** API' sini kullanabiliriz.

SONUÇ:

Projeyi mikrodenetleyiciye debug modda yükleyip **Live Expressions** kısmından c1 ve c2 değişkenlerini incelediğimizde c1 ve c2 değişkenlerinin sırayla arttığını görürüz.

(x)= Variables Breakpoints Expressions Registers Live Expressions SFRs		
Expression	Type	Value
(x)= c1	uint32_t	58
(x)= c2	uint32_t	59
+ Add new expression		

Projenin videosu github' da.

