Exercise 11 Semaphore Signalling

In this exercise we will look at the configuration of a semaphore and use it to signal between two threads.

In the Pack Installer select “Ex 11 Interrupt Signals” and copy it to your tutorial directory.

First, the code creates a semaphore called sem1 and initialises it with zero tokens and a maximum count o five tokens.

```
osSemaphoreId_t sem1;

static const osSemaphoreAttr_t semAttr_SEM1 = {
.name = "SEM1",
};

void app_main (void *argument) {

    sem1 = osSemaphoreNew(5, 0, &semAttr_SEM1 );
```

The first task waits for a token to be sent to the semaphore.

```
__NO_RETURN void led_Thread1 (void  *argument) {

for (;;) {
    osSemaphoreAcquire(sem1, osWaitForever);
    LED_On(1);
    osSemaphoreAcquire(sem1, osWaitForever);
    LED_Off(1);
}
```

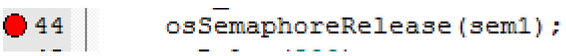
While the second task periodically sends a token to the semaphore.

```
__NO_RETURN void led_Thread2 (void *argument) {

for (;;) {
    osSemaphoreRelease(sem1);
    LED_On(2);
    osDelay(500);
    osSemaphoreRelease(sem1);
    LED_Off(2);
    osDelay(500);
}}
```

Build the project and start the debugger

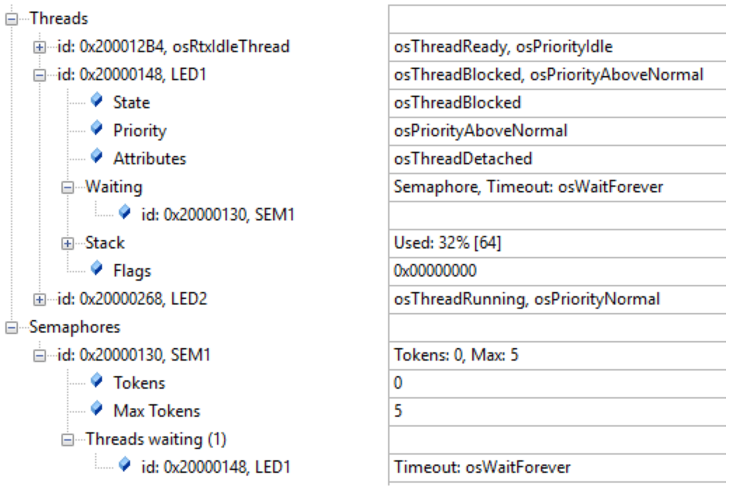
Set a breakpoint in the led_Thread2 task



```
    osSemaphoreRelease(sem1);
```

Fig 46 Breakpoint on the semaphore release call in led_Thread2

Run the code and observe the state of the threads when the breakpoint is reached.



```
Threads
  id: 0x200012B4, osRtxIdleThread     osThreadReady, osPriorityIdle
  id: 0x20000148, LED1                osThreadBlocked, osPriorityAboveNormal
      State                           osThreadBlocked
      Priority                        osPriorityAboveNormal
      Attributes                      osThreadDetached
    Waiting                           Semaphore, Timeout: osWaitForever
        id: 0x20000130, SEM1
    Stack                             Used: 32% [64]
      Flags                           0x00000000
  id: 0x20000268, LED2                osThreadRunning, osPriorityNormal
Semaphores
  id: 0x20000130, SEM1                Tokens: 0, Max: 5
      Tokens                          0
      Max Tokens                      5
    Threads waiting (1)
        id: 0x20000148, LED1          Timeout: osWaitForever
```

Fig 47 Led_Thread1 is waiting to acquire a semaphore

Now led_thread1 is blocked waiting to acquire a token from the semaphore. led_Thread1 has been created with a higher priority than led_thread2 so as soon as a token is placed in the semaphore it will move to the ready state and pre-empt the lower priority task and start running. When it reaches the osSemaphoreAcquire() call it will again block.

Now block step the code (F10) and observe the action of the threads and the semaphore.