1. 專案09\_semaphore:

Linux是一個多任務操作系統，肯定會存在多個任務共同操作同一段記憶體或者設備的情況，多個任務甚至中斷都能訪問的資源叫做共享資源。在驅動開發中要注意對共享資源的保護，也就是要處理對共享資源的並發訪問。

1. 併發與競爭:

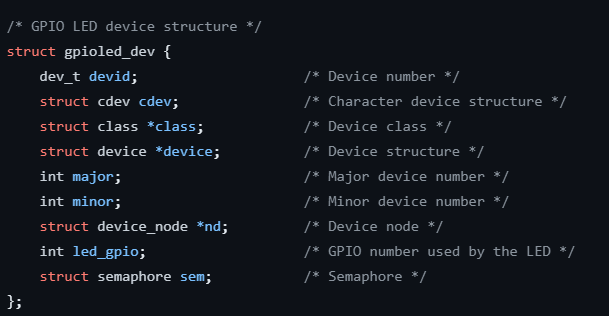
Linux系統是個多任務操作系統，會存在多個任務同時訪問同一片記憶體區域，這些任務可能會相互覆蓋這段記憶體中的數據，造成記憶體數據混亂。大致上原因有以幾項:

1. 多線程並發訪問，Linux是多任務（線程）的系統，所以多線程訪問是最基本的原因。
2. 搶佔式並發訪問，從2.6版本內核開始，Linux內核支持搶佔，也就是說調度程序可以在任意時刻搶佔正在運行的線程，從而運行其他的線程。
3. 中斷程序並發訪問，因為硬體中斷的權利是很大的。
4. SMP（多核）核間並發訪問，現在ARM架構的多核SOC很常見，多核CPU存在核間並發訪問。"
5. 保護的內容與方法:

我們實際要保護的內容就是共享數據，透故過防止併發的訪問來進行保護。

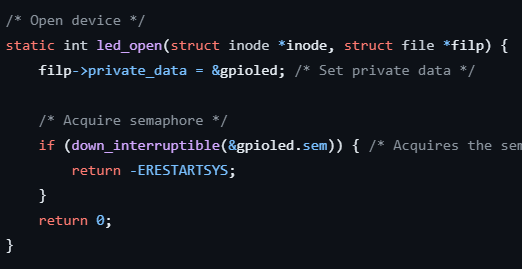
* 保護的方法有:
  + 原子操作(atomic):指的是不能再進一步分割的操作。而Linux Kernel中定義了atomic\_t的結構體於include/linux/types.h 文件中。(已於專案07\_atomic討論過)
  + 自旋鎖(Spinlock): (已於專案08\_spinlock討論過)
* 先前提到的原子操作(atomic)只支援整數類型的變數，但是在實際專案中不可能只需要保護整數類型的變數，因此需要有它方式來對數據進行保護。這裡的方法便是自旋鎖。
* 當一個線程要訪問某個共享資源的時候，首先要先獲取相應的鎖。鎖只能被一個線程持有，只要此線程不釋放持有的鎖，那麼其他的線程就不能獲取此鎖。對於自旋鎖而言，如果自旋鎖正在被線程 A 持有，線程 B 想要獲取自旋鎖，那麼線程 B 就會處於等待狀態。
* 這裡我們可以看到自旋鎖的一個缺點：那就是等待自旋鎖的線程會一直處於自旋狀態，這樣會浪費處理器時間，降低系統性能，所以自旋鎖的持有時間不能太長。自旋鎖適用於短時期的輕量級加鎖，如果遇到需要長時間持有鎖的場景，那就需要換其他的方法了
  + - 訊號量(Semaphore):
* Linux 內核也提供了信號量機制，信號量常常用於控制對共享資源的訪問(除了Linux系統FreeRTOS也有提供Semaphore的使用機制)。
* 比於自旋鎖，信號量可以使線程進入休眠狀態。可以看出，使用信號量會提高處理器的使用效率，畢竟Thread不用一直在那裡‘自旋’等待。但是，信號量的開銷要比自旋鎖大，因為信號量使Thread進入休眠狀態後會進行線程切換，而切換線程會產生開銷。(此處的開銷指系統資源和時間的消耗)，而Semaphore有以下特點:
* 因為信號量可以使等待資源的線程進入休眠狀態，因此適用於那些佔用資源比較久的場合。
* 因此信號量不能用於中斷中，因為信號量會引起休眠，而中斷不能休眠。
* 如果共享資源的持有時間比較短，那就不適合使用信號量了，因為頻繁的休眠、切換線程引起的開銷要遠大於信號量帶來的那點優勢。
  + - 互斥鎖(Mutex): 之後專案10\_mutex會提到。

1. 實作Semaphore:
   1. 首先先將semaphore結構體變數定義在設備結構體中。(如下圖一)



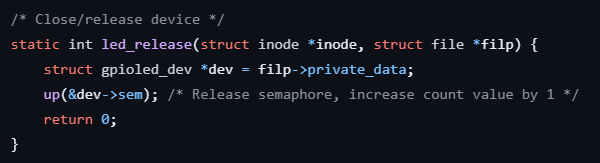
(圖一: 設備結構體成員變數)

* 1. 在open()函式中透過down\_interruptible(&gpioled.sem)來獲取信號量，此函數進入休眠以後是可以被信號打斷的。(如下圖二)



(圖二:open()函式內容)

* 1. 在release()函式中使用up(&dev->sem)，釋放信號量。(如下圖三)



(圖三:release()函式內容)

* 1. 最後在驅動入口函式init()中透過sema\_init()函式來初始化semaphore。(如下圖四)



(圖四: 驅動入口函式init()內容)

※總結:

由於spinlock可能會造成CPU效能的浪費，因此semaphore可以讓等待資源的線程進入休眠狀態，因此適用於那些佔用資源比較久的場合。

如果共享資源的持有時間比較短，那就不適合使用信號量了，因為頻繁的休眠、切換線程引起的開銷要遠大於信號量帶來的那點優勢。