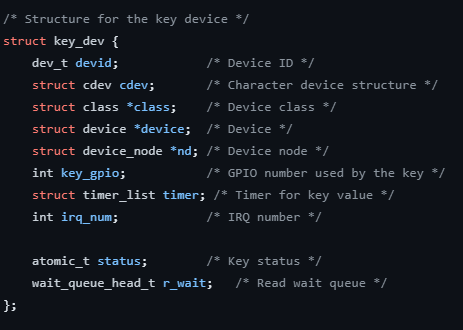
1. 專案14\_blockio:

* 阻塞和非阻塞 IO 是 Linux 驅動開發裡面很常見的兩種設備訪問模式，本專案沿用13\_irq來示範如何在驅動程序中使用等待隊列和 poll 機制。

1. 阻塞和非阻塞**IO:**

* 這裡的IO指的是Input/Output，也就是輸入/輸出，是應用程式對驅動設備的輸入/輸出操作。當應用程式對設備驅動進行操作的時候，如果無法獲取到設備資源，那麼阻塞式IO就會將應用程式對應的線程掛起(休眠)，直到設備資源可以獲取為止(喚醒)。對於非阻塞IO，應用程式對應的線程不會掛起，它要麼一直輪詢等待，直到設備資源可以使用，當設備不可用或數據未準備好的時候，會立即向內核返回一個錯誤碼，表示數據讀取失敗。

1. wait\_queue\_head:
   * 阻塞訪問最大的優點就是當設備不可用時進入休眠狀態讓出CPU資源。但是當設備可以操作時記得將其喚醒，而Linux Kernel中提供wait\_queue來實現喚醒的工作。
   * 原理很簡單，當有Thread要進入休眠狀態就必須加入wait\_queue中，當Thread要被喚醒時再從wait\_queue移除。
2. blockio實作:
   1. 先在設備結構體中定義wait\_queue\_head\_t成員變數。(如下圖一)



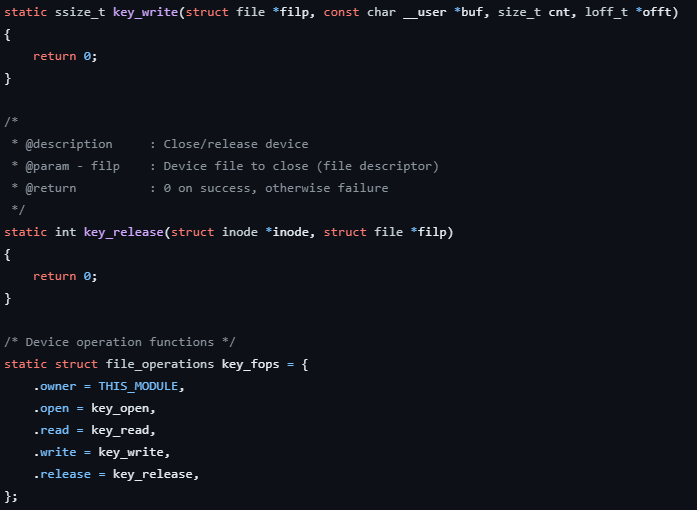
(圖一: 設備結構體)

* 1. 接著定義file\_operations結構體及其內部函數。(如下圖二)我們只有在read()函式新增wait\_event\_interruptable(wq\_head, condition)內容(如下圖二)，當condition條件滿足時，喚醒wq\_head進程。此處沒有特別定義open()函式內容。(如下圖二)



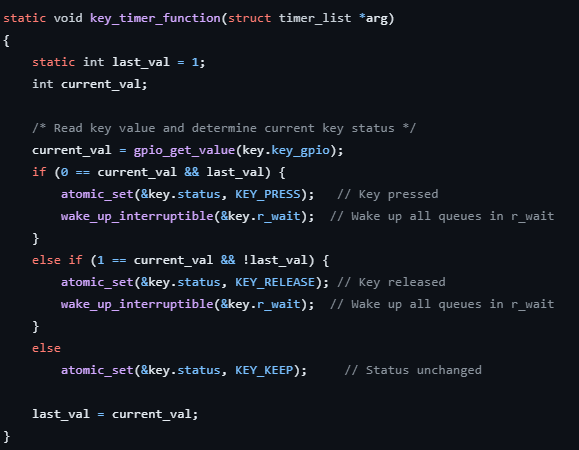
(圖二:open()及read()函式內容)

剩餘的操作函式內容，並沒有特別定義內容。(如下圖三)



(圖三:release()函式，write()函式)

* 1. 接著，在key\_timer\_function()中使用wake\_up\_interruptable()來喚醒read()函式中休眠的進程。(如下圖四)



(圖四: key\_timer\_function()函式內容)

* 1. 當然在具體使用wait\_queue\_head\_t之前需要在驅動入口函數以init\_waitqueue\_head()函式來初始化。wait\_queue\_head\_t(如下圖五)



(圖五:驅動入口函式部分內容)

(\*\*\*剩餘的部分和字元設備驅動一樣，由於前面許多專案都說明過本章不贅述。)

※總結:

由於應用程式對驅動設備的輸入/輸出操作時，不一定有設備可以操作，所以出現阻塞和非阻塞IO，而本專案的阻塞IO主要優點是當沒有設備可以操作時自動休眠讓出CPU資源，當設備可以操作時在自動喚醒。

Linux Kernel則是提供相應的API函是讓我悶驅動開發人員更加方便實現阻塞和非阻塞IO。

大致上流程都差不多，都是先在設備結構體先定義成員變數並在驅動入口函數中初始話就可以直接使用了，舉凡嫌起個專案的atomic, spinlock, gpio, IRQ, mutex都是如此。