1. 專案13\_irq:

* 中斷，不論是在MCU，SOC開發中使用非常頻繁使用的功能，Linux系統也因此幫我們提供了非常完善的中斷框架API。我們只需要申請中斷，然後註冊中斷處理函數，就可以直接使用非常方便。
* 此外這個專案會把功能模塊化，也就是將各個功能以函式方式獨立出來，目的是要簡化驅動入口函數內容也增加整體程式碼的可讀性。

1. Linux系統中斷概述:

* 首先回顧一下Bare-Metal環境下(例:MCU)如何使用中斷:

1. Enable IRQ，並初始化相應暫存器。
2. 註冊中斷處理函式(IRQ Handler)，也就是向irqTable加入中斷處理函式。
3. 當中斷發生後會透過irqTable去查找相應中斷服務函數並執行。

* 硬中斷:

每個硬體設備都有一個中斷號，硬中斷是由硬體設備觸發的信號，用來通知CPU有事件需要立即處理。這些中斷是異步的，可以在任何時刻發生，並會暫停當前正在執行的程序，轉而執行中斷處理程序。

* 軟中斷:

由軟體生成的中斷信號，用於處理非緊急事件或需要延遲處理的任務。與硬中斷不同，軟中斷不由硬體設備直接觸發，而是由軟體（如內核或應用程序）生成並排入中斷隊列，等待合適的時機進行處理。

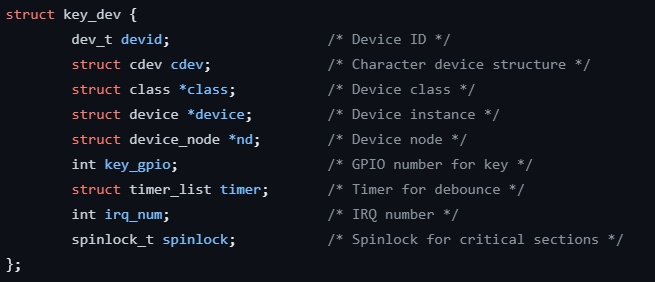
* 上半部和下半部:
* 只要中斷觸發就會執行中斷處理函數，而中斷處理函數執行時間越短越好，但是往往有些中斷比較費時，就必須針對這類中斷進行處理。於是就分成了上半部和下半部:

上半部：上半部就是中斷處理函數，那些處理過程比較快，不會占用很長時間的處理就可以放在上半部完成。

下半部：如果中斷處理過程比較耗時，那麼就將這些比較耗時的代碼提取出來，交給下半部去執行，這樣中斷處理函數就會快進快出。

* 如果處理的任務不希望被打斷，且對時間較敏感，或與硬體有關可以放在上半部，除此之外可以考慮放在下半部。

1. 中斷實作:
   1. 設備結構體，多宣告一個irq\_num成員變數來存放IRQ Number。(如圖一)



(圖一: 設備結構體內容)

* 1. 設備操作函式(file\_operations)，以及read(), write(), open(), release()函數內容。而read()函數則使用spinlock來保護驅動程式及應用程式的共享數據

(如下圖二，圖三)。

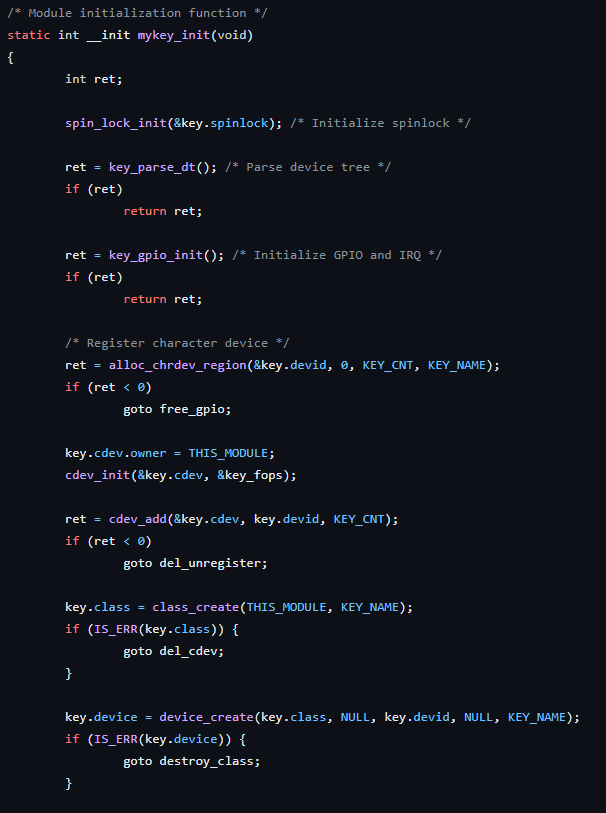


(圖二:read()和open()函式內容)

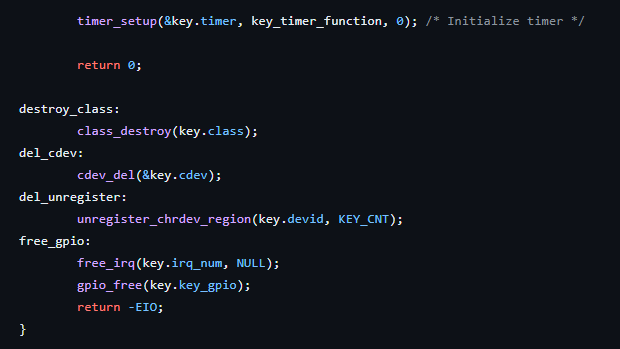


(圖三:write()，release()函式及file\_operations)

* 1. 接著式驅動出入口函式的註冊與實現(如圖四，圖五)。如此一來，整體驅動入口函式就簡潔許多。
     + - 此處key\_parse\_dt()函式打包先前of函數獲取Device Tree的功能(如圖六)。
* 透過irq\_of\_parse\_and\_map ()函數以便從設備樹節點中讀取中斷描述並將其轉換為內核可以使用的 IRQ 號。
  + - * 透過key\_gpio\_init()函式初始化GPIO及IRQ(如圖七)。
        + 透過request\_irq()函數請求中斷號並綁定一個中斷處理函式
        + 透過irq\_get\_trigger\_type()函數設定中斷號的觸發類
        + 當有錯誤發生時需要透過free\_irq()來釋放IRQ。
      * 再以timer\_setup()函式初始化定時器並綁定key\_timer\_function()為定時器處理函數(如圖八)。



(圖四: 驅動入口函式內容)



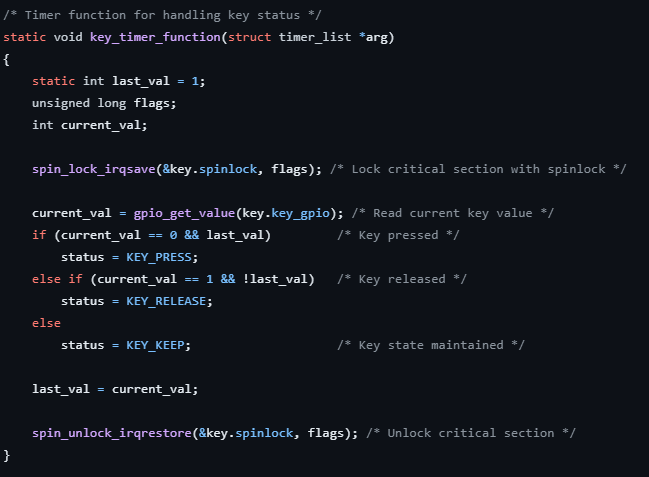
(圖五: 驅動入口函式內容)



(圖六: key\_parse\_dt(void)函式內容)



(圖七: key\_gpio\_init(void)函式內容)



(圖八: key\_timer\_function()函式內容)

* 1. 最終，驅動出口函式的註冊和實現，單存釋放驅動入口函數中所引用的設備計時器，及中斷。外加一些作者訊息(如圖九)



(圖九: 驅動出口函式的註冊和實現)

※總結:

如此一來就完整實現，字元設備驅動，中斷應用，Kernel Timer使用，及共享數據保護。整體的開發流程大致上就是這樣，尤其字元設備驅動的開發框架也很熟悉了。