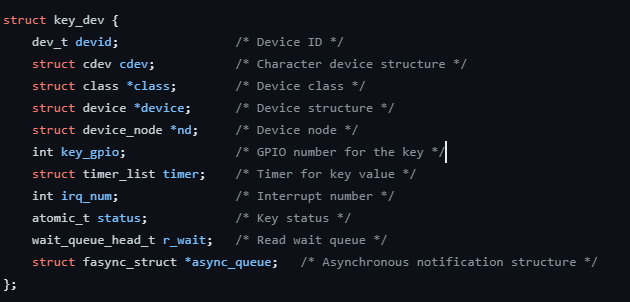
1. 專案16\_asyncnoti:

* Asynchronies Notification，在前兩個專案中使用阻塞或者非阻塞的方式來讀取驅動中按鍵值都是應用程序主動讀取的，對於非阻塞方式來說還需要應用程序通過 poll 函數不斷地輪詢。最好的方式就是驅動程序能主動向應用程序發出通知，報告自己可以訪問，然後應用程序再從驅動程序中讀取或寫入數據，類似於中斷。Linux 提供了異步通知這個機制來完成此功能，本專案我們就來實現異步通知。

1. 異步通知(Asynchronies Notification):

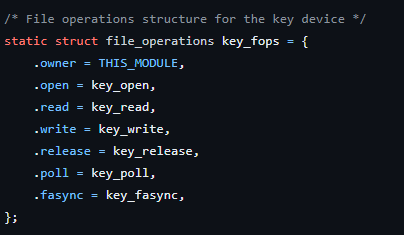
* 我們希望驅動程序能主動向應用程序發出通知，並報告自己可以訪問。Linux系統中恰好提供”信號”的概念來實現異步通知。信號是一種用於通知進程某些事件已經發生的機制，有點類似軟中斷，其定義在Kernel文件中arch/xtensa/include/uapi/asm/signal.h的位置。

1. Asynchronies Notification實作:
2. 首先第一步要先在設備結構體中定義fasync\_struct成員變數(如下圖一)



(圖一:設備結構體內容)

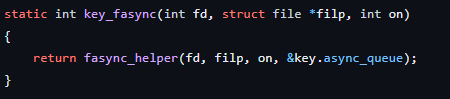
1. 接著在file\_operations中定義fasync()函數。(如圖二)



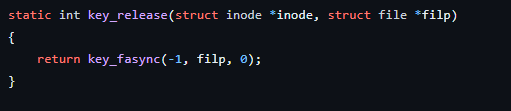
(圖二:file\_operations操作函式結構體)

1. 實現fasync()函式(如下圖三)，在fasync()函式中透過fasync\_helper()函式來初始化先前設備結構體中的fasync\_struct成員變數。並在release()函式中呼叫。

(如下圖四)



(圖三: fasync()函式內容)



(圖四:release()函式內容)

1. 最後剩餘部分就是很熟悉的字元設備驅動實現。此處驅動出入口函數都不需要對fsync進行處理，因為這部分只攸關於Kernel Space和User Space之間的溝通而已，所以不須做任何動作只需完善剩餘字元設備驅動框架即可。

※總結:

應用程序對異步通知的處理 應用程序對異步通知的處理包括以下三步：

1. 註冊信號處理函數:

* 應用程式根據驅動程式所使用的信號來設置信號的處理函數，應用程式使用signal函數來設置信號的處理函數。

1. 將本應用程序的進程號告訴給內核:

* 使用fcntl(fd, F\_SETOWN, getpid())將本應用程序的進程號告訴給Kernel。

1. 開啟異步通知:

* 使用如下兩行程序開啟異步通知:
* flags = fcntl(fd, F\_GETFL); /\* 獲取當前的進程狀態 \*/
* fcntl(fd, F\_SETFL, flags | FASYNC); /\*開啟當前進程異步通知功能 \*/

重點就是通過fcntl函數設置進程狀態為FASYNC，經過這一步，驅動程序中的 fasync函數就會執行。