OSTeam19\_Report

Team\_member：鍾昀諠，陳麒懋

Contribution：

鍾昀諠：Part1，code report

陳麒懋：Part2，trace report

Part I：Console I/O

Syscall.h



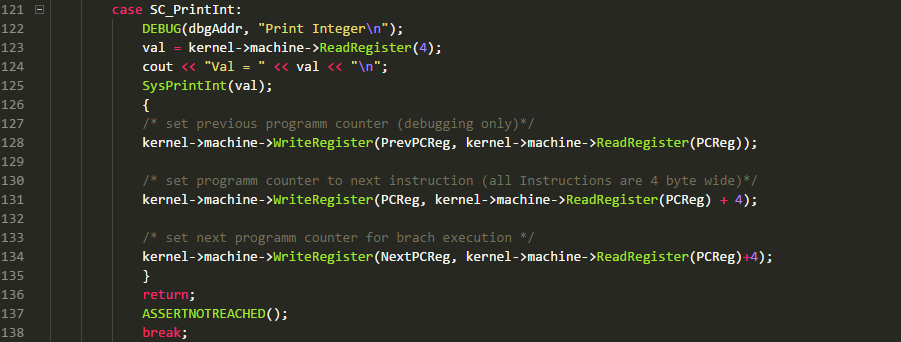
因為Syscall.h只是整個system call的介面，所以在這裡只需要宣告要新增的system call函數和常數讓接下來的exception.cc用於判斷被呼叫的是哪一個system call。

Start.S



這邊負責的是以MIPS指令把資料寫進register裡的動作，把$2寫入代表這是哪個system call的常數。

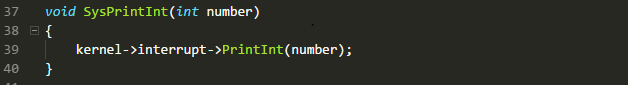
Exception.cc



Exception.cc負責查看$2裡面存放的是代表哪一個system call的常數，並把他map到對應的函數進行處理。

這裡我們新增了處理PrintInt的case，在裡面把要印出來的值從$4取出來傳下去，並處理Program Counter。

Ksyscall.h

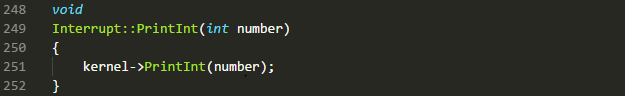


Exception.cc把取出來要印的數字傳遞到SysPrintInt來，SysPrintInt再把參數傳到kernel處理，因為要轉換成kernel mode執行，所以必須先傳到interrupt修改user mode，再從interrput 呼叫 kernel來執行。

Interrupt.h/cc



在.h檔宣告函數，然後在.cc實作，是比較好的一種coding style。

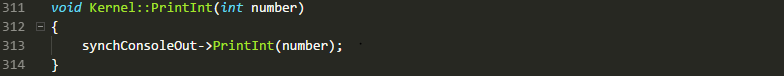


在interrupt.cc就可以直接呼叫kernel進行處理。

Kernel.h/cc



宣告函數。

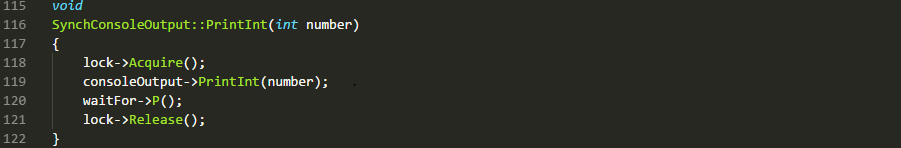


因為顯示設備是asynchronous的，所以不能直接call console.cc把他印出來，而是要先到synchConsoleOut去把device鎖起來不讓其他人使用，確保不會被干擾再印。

SynConsole.h/cc



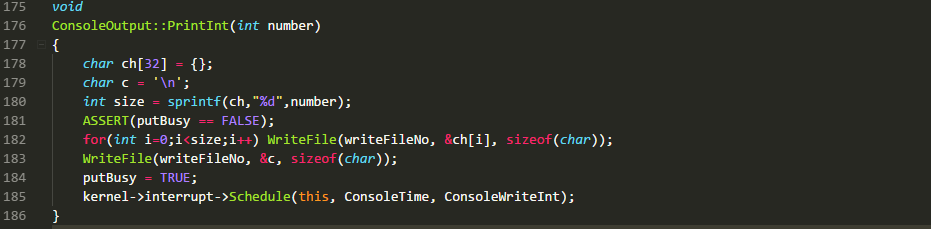
宣告函數



第118行call threads/synch.cc 裡面的lock函數把output device鎖起來，接著119行呼叫consoleOutput把數字輸出，120行負責判斷output是否已經結束，121行重新開放device供其他程式使用。

Console.h/cc





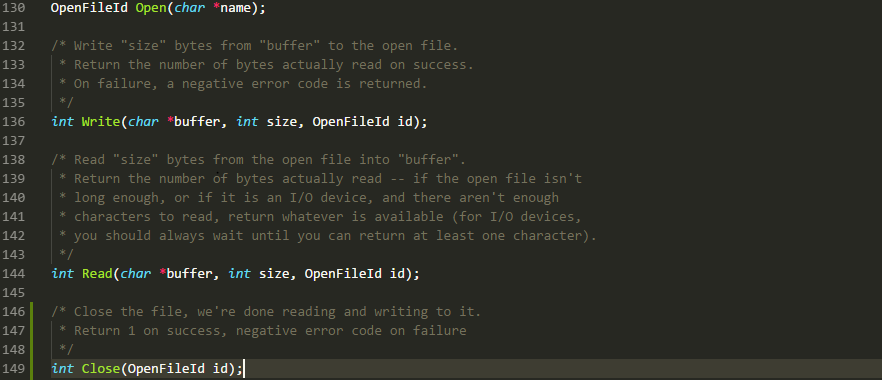
最後終於可以實作輸出數字的部分，先用sprintf把正負皆可的數字轉成由字元組成的array，然後一個一個字元輸出，然後把這個interrupt放進scheduler 放進執行序列阜排程執行。

Part II：File I/O

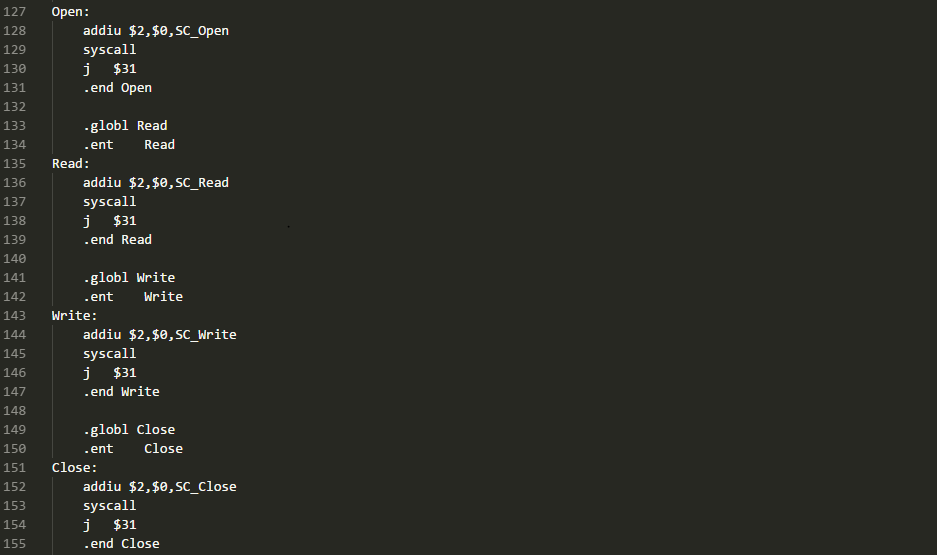
Syscall.h



代表system call的常數，和PartI一樣是為了讓OS知道是哪一個system call被呼叫。



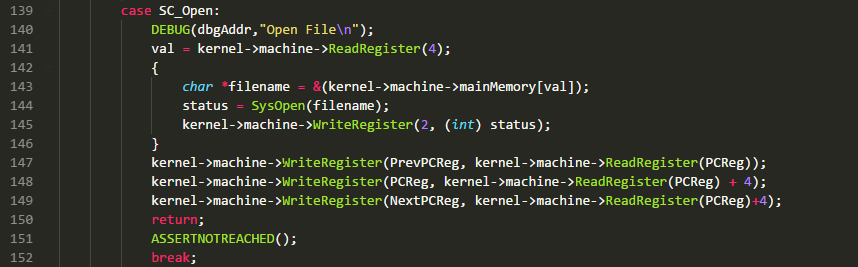
Start.S



把syscall.h定義的syscall常數放進$2裡面。

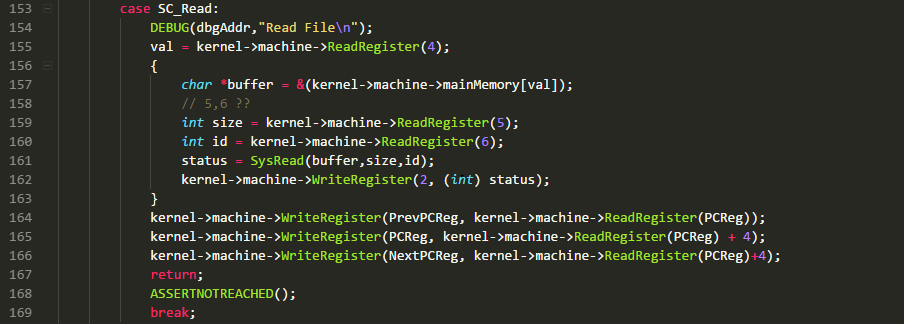
Exception.cc

Open



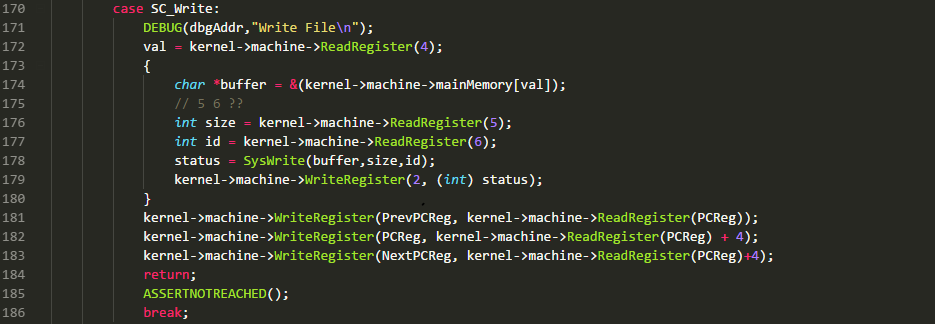
把要開啟的file的位址從$4取出來傳下去，並把開啟檔案的結果寫到$2，接著處理PC後return該檔案的File Descriptor或是代表失敗的-1。

Read



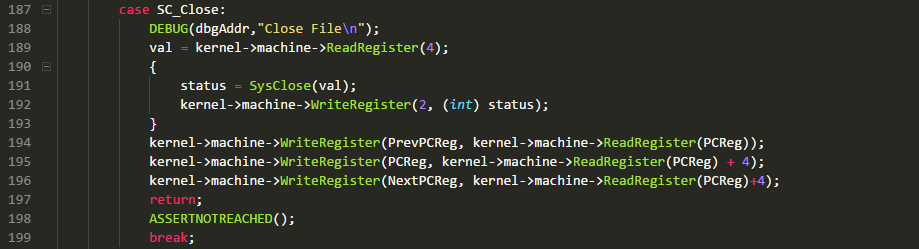
從registers讀出要寫入的buffer的位址、資料的大小以及要讀檔案的fileID並傳下去，回傳讀入的byte數並寫到$2，接著處理PC後return。

Write



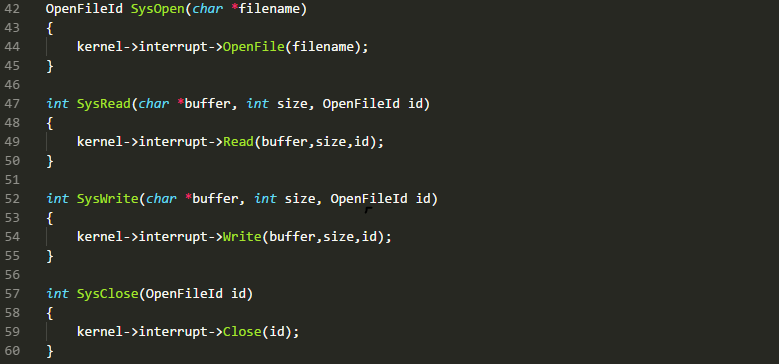
從registers讀出要寫入的檔案的位址、資料的大小以及要讀的buffer的fileID傳下去，回傳寫入的byte數並寫到$2，接著處理PC後return。

Close



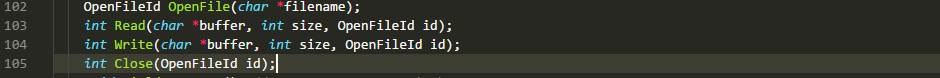
從registers讀出要關掉的檔案的位址傳下去，如果成功關閉回傳1，失敗回傳0，並寫入到$2，接著處理PC後return。

Ksyscall.h

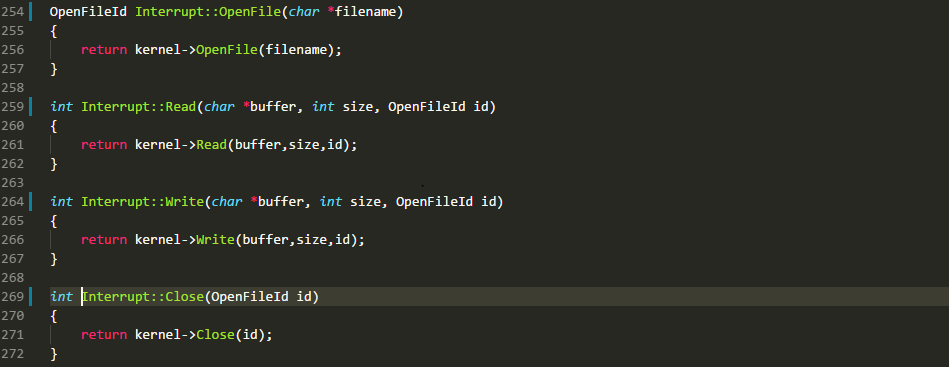


Exception.cc把取出來的參數傳遞到Ksyscall.h裡面來，Ksyscall.h再把參數傳到kernel處理，因為要轉換到kernel mode執行，所以必須先call interrupt修改user mode，再從interrput 呼叫 kernel來執行。

Interrupt.h/cc

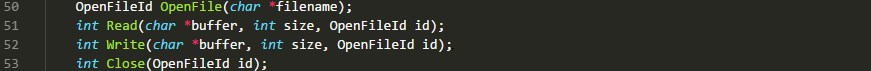


宣告函數

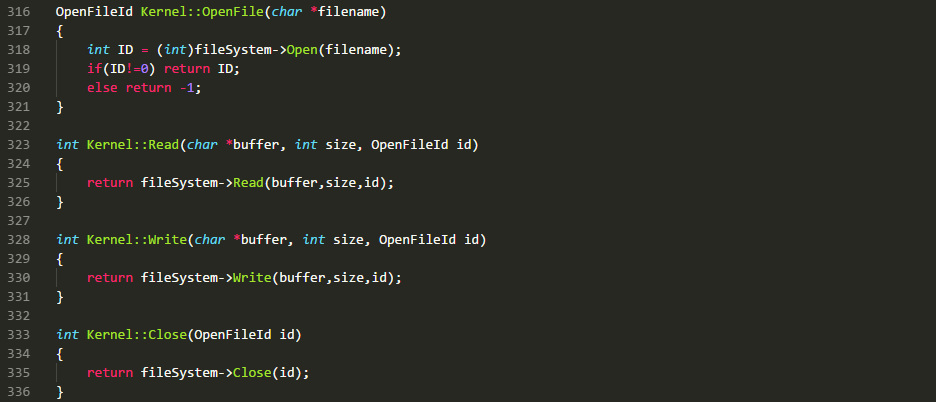


在interrupt因為已經把mode切換成kernel mode，所以就可以直接call kernel執行。

Kernel.h/cc



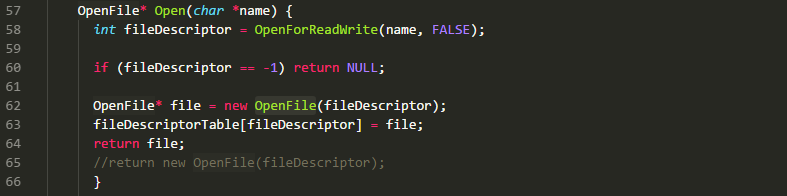
宣告函數



Kernel直接呼叫filesystem，由filesystem裡面的函數去實作。注意在OpenFile中我們把傳回來的指標先轉換成int才傳回去。

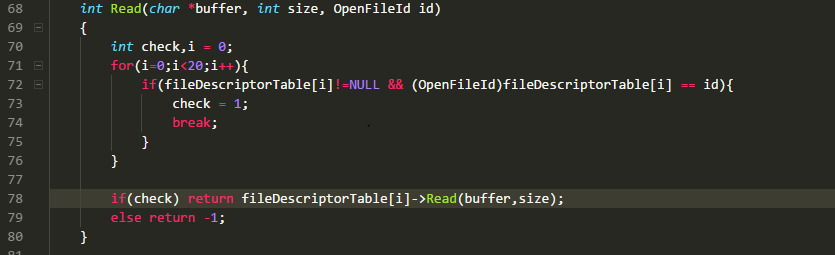
Filesys.h

Open



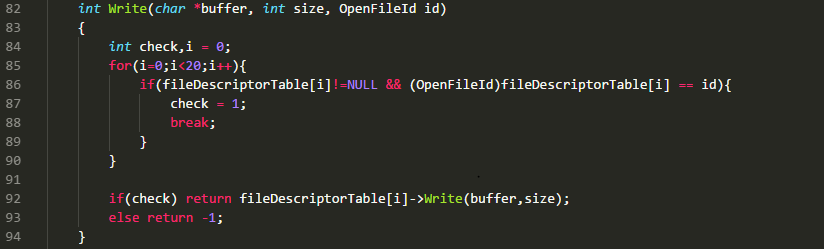
用lib/sysdep的OpenForReadWrite尋找檔案，如果沒有找到的話就回傳null，找到的話就用其index創建一個OpenFile物件，接著以一個指標指向他，加入到FileDescriptorTable並回傳指標。

Read



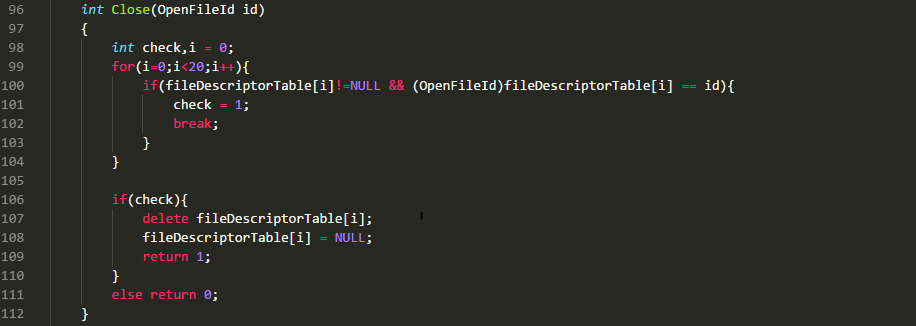
用傳入的FileID在FileDescriptorTable中尋找想讀的檔案是否已經被開啟了。因為table裡面存的是指標，所以需要先做型別轉換才能跟FileID比較。接著78行用openfile物件內涵的public函數Read完成。

Write



用傳入的FileID在FileDescriptorTable中尋找我們想寫入資料的檔案是否已經被開啟了。因為table裡面存的是指標，所以需要先做型別轉換才能跟FileID比較。接著92行用openfile物件內涵的public函數Write完成。

Close



用傳入的FileID在FileDescriptorTable中尋找我們想關閉的檔案是否已經被開啟了，如果有找到就把table內的OpenFile物件delete掉，把table的值設成null然後回傳1，如果沒有找到則回傳0。因為table裡面存的是指標，所以需要先做型別轉換才能跟FileID比較。