**CS342301: Operating System**

**MP1: System Call**

**Team31**

**111060013 劉祐廷 111060019 黃子恩**

|  |  |  |
| --- | --- | --- |
|  | **劉祐廷** | **黃子恩** |
| **Contribution** |  |  |

1. **Trace Code**
   1. **SC\_Halt**
      1. **Machine::Run()**

這個function負責模擬NachOS中的CPU執行程序的過程，他會先使用kernel->interrupt->setStatus(UserMode);，將模式設定成usermode，接著用後面的無限迴圈for(;;){}模擬CPU執行時一直去fetch instruction的動作，在迴圈裡面會先呼叫OneInstruction()來執行instruction，在每次OneInstruction執行之後會呼叫OneTick()來檢查有無interrupt發生並處理。

* + 1. **Machine::OneInstruction()**

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

先去讀取registers[PCReg]裡面的那個instruction，讀不到則return，有讀到的話就做decode。

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

接著使用opCode取找尋這個指令所代表的動作。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

以OP\_ADD這個動作為例，若遇到exception的狀況發生，會把錯誤的代碼當參數傳給RaiseException()處理，沒有exception發生的話就把各自的操作做完。

* + 1. **Machine::RaiseException()**



把遇到問題的virtual address放進專門放bad virtual address的register裡面。

一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

自動產生的描述

切到system mode執行ExceptionHandler()，執行完成後切回usermode。

* + 1. **ExceptionHandler()**

一張含有 文字, 字型, 螢幕擷取畫面, 數字 的圖片

自動產生的描述

從start.S裡面可以發現，這些exception的編碼都被放在了r2這個記憶體裡面。



因此可以透過讀取r2的值去獲取system call code。（system call code存放在r2）

一張含有 字型, 文字, 圖形, 螢幕擷取畫面 的圖片

自動產生的描述

先檢查這個system call code是否存在。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

接著依照system call code執行不同的動作，由於傳入的system call code為SC\_Halt，因此會執行這段程式碼去呼叫SysHalt()。

* + 1. **SysHalt()**



呼叫定義在kernel可接受的interrupt操作裡的Halt()

* + 1. **Interrupt::Halt()**

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

解構整個kernel釋放資源，停止整個system。

* 1. **SC\_Create**

1. **ExceptionHandler()**

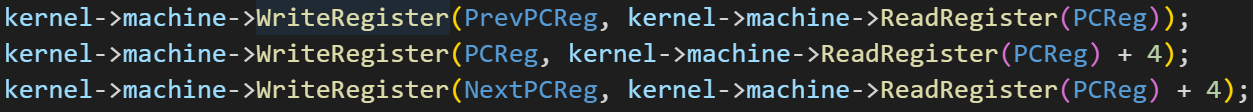


從r4（通常是arg1）讀取欲新增檔案檔名字串的位址。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

把filename提取出來傳給SysCreate()來創建檔案，並把要return的值寫入r2。



更新PC準備執行下一行指令。

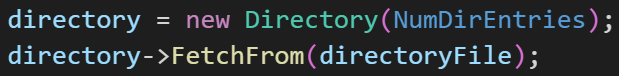
1. **SysCreate()**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

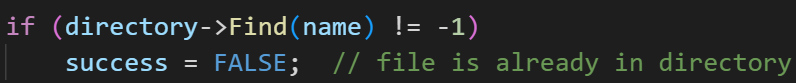
自動產生的描述

呼叫fileSystem裡面的Create()，並把filename傳給他。

1. **FileSystem::Create()**



獲取對應的directory。



確認檔案是否存在，若存在則回傳false。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

檢查是否有足夠的空間創建檔案，若不夠則回傳false。

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

嘗試allocate空間給這個檔案，若空間不夠則回傳false。

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

成功創建檔案，將變更後的資料寫回disk。

一張含有 文字, 字型, 螢幕擷取畫面, 圖形 的圖片

自動產生的描述

解構這些指標以釋放記憶體資源。

* 1. **SC\_PrintInt**
     1. **ExceptionHandler()**

一張含有 文字, 字型, 螢幕擷取畫面, 數字 的圖片

自動產生的描述

從start.S裡面可以發現，這些exception的編碼都被放在了r2這個記憶體裡面。



因此可以透過讀取r2的值去獲取system call code。（system call code存放在r2）

一張含有 字型, 文字, 圖形, 螢幕擷取畫面 的圖片

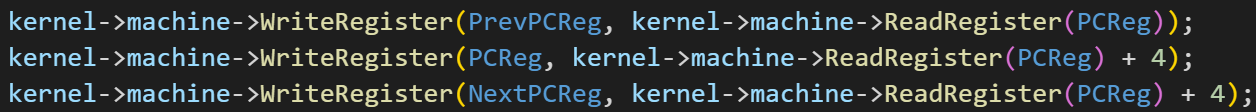
自動產生的描述

先檢查這個system call code是否存在。

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

接著依照system call code執行不同的動作，由於傳入的system call code為SC\_PrintInt，因此會執行這段程式碼去呼叫，首先會去r4（arg1）讀取要輸出的東西存在val，接著把val當參數傳給SysPrintInt()。



更新PC準備執行下一行指令。

* + 1. **SysPrintInt()**

一張含有 文字, 字型, 螢幕擷取畫面, 印刷術 的圖片

自動產生的描述

把要輸出的東西（val）傳給已經定義在kernel的synchConsoleOut裡面的PutInt()。

* + 1. **SynchConsoleOutput::PutInt()**

**一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述**

把要輸出的數字轉換成字串，並用迴圈一個字元一個字元傳給PutChar()。

* lock->Acquire();

這行code在執行共享資源操作之前acquire lock，確保當前的process在進入critical section之前，獨占對該資源的訪問權。這樣可以避免其他process同時訪問 consoleOutput，導致輸出的數據混亂。當lock被某個process持有時，其他嘗試acquire lock的process會被阻塞，直到lock被release。

* lock->Release();

當這段操作完成後release lock，允許其他process進入critical section。這樣能確保在一次完整的輸出操作結束後，其他process可以安全地執行自己的輸出操作。

* + 1. **SynchConsoleOutput::PutChar()**

比起SynchConsoleOutput::PutInt()，只是少了將數字轉換成字串的過程，剩下部分大致上一樣。

* + 1. **ConsoleOutput::PutChar()**



putBusy 表示console output是否忙碌。他是 FALSE時表示console is idle，此時才能開始新的輸出。如果他是 TRUE，表示正在進行輸出，則這次呼叫應該被阻止，這樣可以避免同時輸出多個字符。



模擬將ch輸出到console output中。



把putBusy設為TRUE表示console output忙碌中，可以避免同時輸出多個字符。



呼叫Schedule()以在輸出完成之後raise up interrupt表示輸出完成。

* + 1. **Interrupt::Schedule()**



計算這個interrupt要被觸發的時間

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

建構一個PendingInterrupt，檢查fromNow是否大於0（是否為未來會發生的事件），如果是的話就把這個PendingInterrupt放入sorted的list裡面等待觸發。

* + 1. **Machine::Run()**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Run()會在每次呼叫完OneInstruction()後呼叫OneTick()，目的是在每次執行完一行指令時去檢查有沒有interrupt發生，並更新時間的tick。

* + 1. **Machine::OneTick()**



保存目前的狀態，以便恢復數據時使用。



獲取statistic的指標，以便更新tick。

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

依照目前的模式去更新tick。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

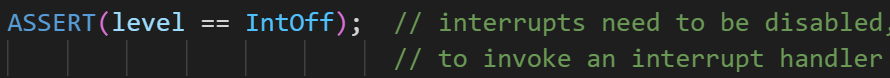
呼叫ChangeLevel()來關掉interrupt，這樣才能讓interrupt handler執行，再來呼叫CheckIfDue()檢查是否有PendingInterrupt到期需要被raise up的，最後再呼叫ChangeLevel()重新開啟interrupt。

一張含有 文字, 字型, 螢幕擷取畫面, 筆跡 的圖片

自動產生的描述

檢查timer是否有請求context switch，有的話就直接切換過去，由於yield是kernel負責的，因此要先切換到system mode，然後讓CPU切換工作，切換完後要把status恢復以繼續執行。

* + 1. **Interrupt::CheckIfDue()**



確保interrupt有被關掉。

一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

自動產生的描述

如果沒有interrupt在等待，則直接return FALSE。



讀取最快要執行的那個interrupt。



若還不到interrupt要發生的時間。

一張含有 字型, 文字, 螢幕擷取畫面, 圖形 的圖片

自動產生的描述

若還沒到觸發時間則return FALSE。

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

advanceClock是true的時候表示目前來沒有準備好其他的process，因此system可以把時間推進到這個interrupt，這也會增加idle ticks，所以也要更新idleTicks。

一張含有 文字, 字型, 螢幕擷取畫面, 筆跡 的圖片

自動產生的描述

若有machine的模擬器存在，則需要執行delay load，這是kernel跟hardware之間的一個步驟。

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

把inHandler設成TRUE，表示正在處理interrupt，使用while loop檢查pending中的interrupt是否到期，到期的話就remove掉並按照next的interrupt type呼叫callOnInterrupt的CallBack()來處理interrupt，最後在interrupt處理完之後，要把inHandler改回FALSE，表示interrupt處理完成。

* + 1. **ConsoleOutput::CallBack()**

一張含有 文字, 字型, 螢幕擷取畫面, 行 的圖片

自動產生的描述

當一個字元輸出完成，putBusy會被設成FALAE，表示可以輸出下一個字元了，更新成功輸出的字元數量，接著呼叫callWhenDone的CallBack()，通知系統可以輸出下一個字了。

* + 1. **SynchConsoleOutput::CallBack()**

一張含有 文字, 字型, 螢幕擷取畫面, 筆跡 的圖片

自動產生的描述

當輸出完成後，用waitfor的V()通知系統解除lock，可以換下一個process輸出了。比起ConsoleOutput::CallBack()多了一個lock的機制，可以確保multiprocess下輸出的正確性。

* 1. **Makefiles**

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

以halt舉例說明，當在終端機輸入make halt時，會跳到halt這個label這裡，檢查冒號後面的檔案，依序遞迴進去將每個檔案做好。

…

CFLAGS是個變數，存放一些指令用來告訴編譯器在編譯過程中使用哪些選項。

* -g：告訴compiler產生調試信息
* -G 0：-G通常用於MIPS架構的compiler，0表示放入global pointer area (GPA)中的變數的大小，0表示不做任何優化。
* -c：告訴compiler只編譯source files，不進行link的動作，這會產生.o檔。
* $(INCDIR)：這是一個變數，存放標頭檔的路徑，當編譯器遇到#include時會查找這個路徑。
* -B/usr/bin/…：-B會讓後面的路徑變成二進制的，讓compiler可以從這個路徑中尋找linker之類的工具。



CC這個變數表示gcc這個程式(compiler)的路徑。



LD這個變數表示ld這個程式(linker)的路徑。



這行負責把halt.c做成halt.o。

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

這一段程式碼負責將halt.o和start.o鏈接起來，做成halt.coff，.coff檔為mips可以執行的檔案，但由於linux的架構是x86，所以要再轉成.noff變成可以在模擬器上面執行mips的檔案。

1. **Implementation of I/O System Calls in NachOS**
2. **Difficulties**
3. **Feedback**