

NachOS-4.0

MP1



第21組

106062173吳東弦(一起trace code，負責整理report)

106062271郭子豪(一起trace code，負責實作coding)

# Trace code

## SC\_Halt

### Machine::Run()

　　當機器開機（執行main）後，作業系統會被載入（實體化kernel），接著在Run()中會跑一個無窮迴圈，不斷執行OneInstruction(instr)和OneTick()，前者就像在監聽是否有instruction要被執行，後者則持續計算經過的時間。

### Ｍachine::OneInstruction()

　　在OneInstruction中首先會看是否有指令要被執行（ReadMem），如果有，就會去解讀這個指令要做什麼（decode），接著再根據指令的opcode，執行相對應的動作。比如在執行Halt函式時，addiu會把後兩個register的值加到前面的暫存器，j會跳到暫存器所指向的位址，而syscall則呼叫RaiseException，準備進行模式切換。

### Machine::RaiseException()

　　執行Halt時，原本是在UserMode中，在RaiseException中，由於要執行系統呼叫（透過ExceptionHandler），會先切換到SystemMode，等執行完後，再切換回Usermode（透過setStatus）。

### ExceptionHandler()

　　由於MIPS中$v0暫存器，也就是$2暫存器，其中一個作用為存放系統呼叫的類型，因此，在ExceptionHandler中，會依照傳進來的參數與$2暫存器的值，來決定要執行哪一種系統呼叫，以本題Halt為例，根據$2暫存器的值要執行的是SC\_Halt，所以會去呼叫SysHalt。

### SysHalt()

　　在SysHalt中會去執行作業系統中Halt的interrupt服務。

### Interrupt::Halt()

　　在Halt中，最後會將作業系統移出記憶體（delete kernel），模擬虛擬機關機的樣子。

## SC\_Create

### ExceptionHandler()

　　在ExceptiopHandler中，根據$2暫存器的值要執行的是SC\_Create。由於MIPS預設將函數參數依序放在$4~$7暫存器，所以先從$4暫存器將檔名存入主記憶體，再作為參數傳入SysCreate。接著，又因為$2暫存器另一個作用為存放函數的返回值，因此，當建檔成功後回傳的布林值的會讀入$2暫存器。最後，再將PC暫存器遞進。

### SysCreate()

　　在SysCreate中會去執行作業系統中fileSystem的Create服務。

### FileSystem::Create()

　　在Create中，呼叫了OpenForWrite，並利用c語言函式庫的open進行開檔，由於我們只要創建檔案，開檔後就直接close了。

## SC\_PrintInt

### ExceptionHandler()

　　在ExceptiopHandler中，根據$2暫存器的值要執行的是SC\_PrintInt。先從$4暫存器將要印出來的數字存入主記憶體，並作為參數傳入SysPrintInt，最後，再將PC暫存器遞進。

### SysPrintInt()

　　在SysPrintInt中會去執行作業系統中synchConsoleOut的PutInt服務。

### SynchConsoleOutput::PutInt() （同PutChar()）

#### lock->Acquire()

　　當要印出數字時，如果同時有其執行緒也同時需要此I/O，會導致交錯印出，產生錯誤的結果，因此，必須指定當前執行緒（lockholder），不讓別的執行緒進入，而上鎖（P）、解鎖（V）則是由Semaphore（在此為semaphore）負責。

　　在semaphore的P中，會先讓當前執行緒執行（value = = 1），然後將其鎖上，避免其他執行緒進入（value = 0）。如果ready queue有別的執行緒也可能用到此I/O（value = = 0），semaphore會將其轉至waiting queue並使其sleep，避免這段期間這些執行緒回到ready queue。

　　接著，如果ready queue有其他與此I/O無關的執行緒，就會讓CPU繼續Run（asynchronized I/O），否則如果ready queue為空，沒有任何執行緒的話，CPU就會ilde。

#### do { PutChar && waitFor->P() } while ()

　　接著進入do-while迴圈，將字元一個一個印出來，以完成整個數字的輸出。在這個執行緒內，每個輸出都是一個動作，為了避免字元交錯印出，同樣需要一個Semaphore（在此為waitFor）控制，也就是waitFor的P，而P的執行過程與前項相同。

#### lock->Release()

　　當整個數字印完後，就可以將當前執行緒釋放（lockholder），接著，在semaphore的V中，會將其他sleep中的執行緒喚醒，放回ready queue。

### ConsoleOutput::PutChar()

　　PutChar中，模擬將字元輸出到I/O設備的情況（WriteFile），將putBusy設為TRUE，表示正在I/O中，並將輸出後要發出的中斷加入排程（Schedule），也就是對pending插入toOccur指標所指向的callOnInterrupt。

### Machine::Run()

　　如同第一題所示，Run中會跑無窮迴圈，不斷執行OneInstruction(instr)和OneTick()，從semaphore的P和waitFor的P可以知道，這時其實I/O仍然在進行中，此即asynchronized的特性。

### Interrupt::OneTick()

　　OneTick的呼叫，表示時間的推移，每一個Tick可看成時間的最小單位。單純執行一個使用者的指令時，會遞進一個UserTick（1Tick），若是執行系統呼叫或是中斷，則會遞進一個SystemTick（10Tick）。

### Interrupt::CheckIfDue()

　　在CheckIfDue中，會去檢查pending是否有中斷訊號（callOnInterrupt），也就是剛剛在PutChar所做的排程（schedule），如果有的話，代表己經做完一次I/O，準備執行接下來的CallBack。另外在這裡可以發現，在整份code中，CheckIfDue會出現在兩個地方，其一為Idle，其一為Run，也就是不論CPU在執行或是在閒置中，都持續在偵測是否有中斷訊號，這也是asynchronized的特性。

### ConsoleOutput::CallBack()

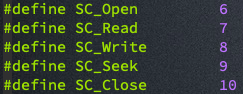
　　這個CallBack將putBusy設為FALSE，表示I/O已經結束，接著呼叫下一個CallBack。

### SynchConsoleOutput::CallBack()

　　這個CallBack呼叫了waitFor的P，也就是表示這個字元已經印完了，要去喚醒其他sleep的字元，準備進行輸出。

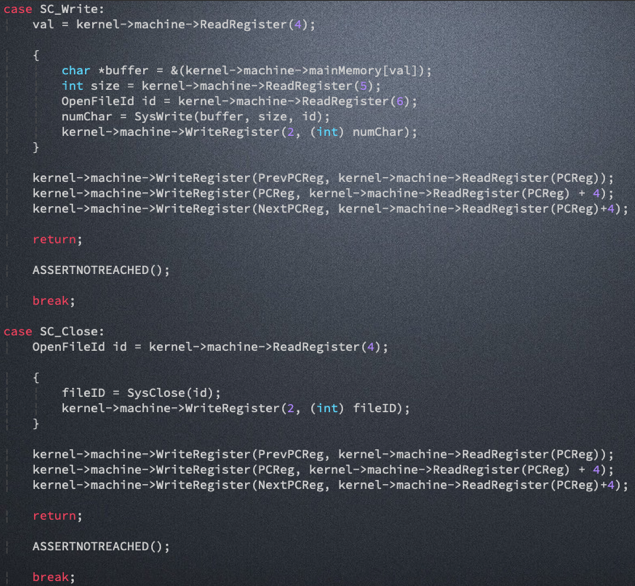
# Implement four I/O system calls in NachOS

## define



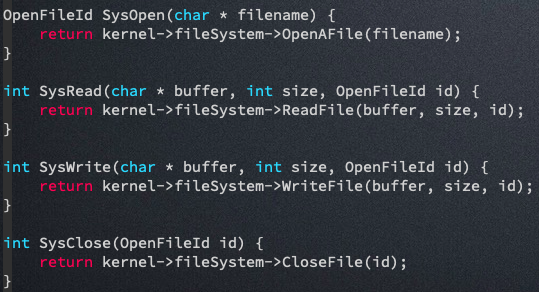
　　這裡將Macro的註解取消，ExceptionHandler才抓得到這些case。

## ExceptionHandler

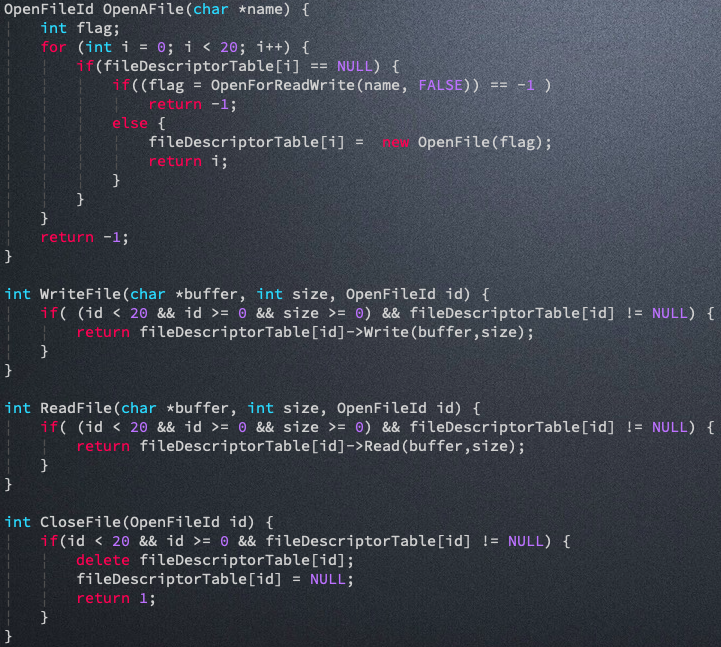
　　從$2取得對應中斷訊號的case後，會依序從$4、$5、$6取得對應的參數，接著呼叫對應的SysCall，執行完後，將PC暫存器遞進。

## SysCall



　　接著SysCall會去呼叫kernel對應的實作。

## FileSystem



#### OpenAFile

　　由於fileDescripterTable最大只有20，所以必須先檢查FDT是否有空位，如果滿了就無法開檔，即使有空位，還要再進入OpenForReadWrite檢查這個檔案是否能被開啟，比如filename是否存在，如果不存在一樣無法開檔，都檢查完確定能開後，就會在FDT下一個空位開啟。

#### ReadFile

　　由於fileDescripterTable最大只有20，所以必須先檢查要讀取的檔案id是否在FDT範圍內，接著檔案必須開啟後才能讀取，也就是還要檢查FDT對應位置是否為NULL，都檢查完後就可以讀取檔案了。

#### ＷriteFile

　　由於fileDescripterTable最大只有20，所以必須先檢查要寫入的檔案id是否在FDT範圍內，接著檔案必須開啟後才能寫入，也就是還要檢查FDT對應位置是否為NULL，都檢查完後就可以寫入檔案了。

#### CloseFile

　　由於fileDescripterTable最大只有20，所以必須先檢查要關閉的檔案id是否在FDT範圍內，接著檔案必須在開啟狀態下才能關閉，也就是還要檢查FDT對應位置是否為NULL，都檢查完後就可以關閉檔案了。

# Ｓummary & FeedBack

　　這次作業由於整份專案非常大，要在許多檔案跳來跳去，再加上有些函數在不同class下名稱很相似，甚至一樣，導致經常迷路，經過多方嘗試，學到了在vscode如何快速跳轉到想去的位置，以及在終端下透過grep查詢，需要的函數定義、呼叫在哪個檔案。

　　其中trace第三題我們認為是本次作業最難的部分，首先得了解在MIPS的規範下，$2與$4~7暫存器有什麼特定的作用，接著理解真正的作業系統中，lock與semaphore如何透過P、Ｖ控制threads執行的順序與安全性，最後是執行中斷訊號時，兩次callback如何被呼叫。