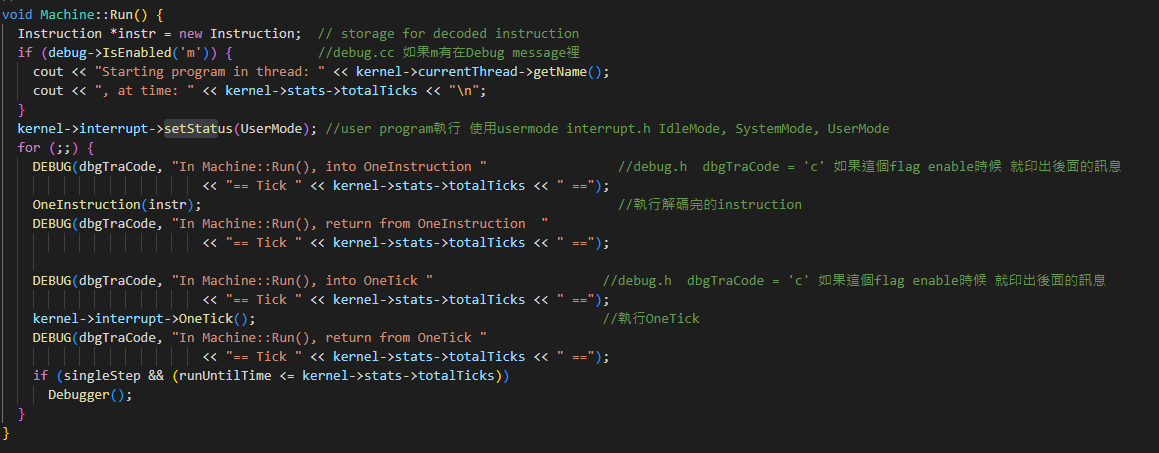
## (1)SC\_Halt:

**Machine::Run()**

把解碼後的instruction如ADD,BEQ丟進OneInstruction(instr) 做運算

### Machine::OneInstruction()

模擬CPU執行各個指令,其中還有Exception的opcode,遇到異常會丟出RaiseException訊息

### Machine:RaiseException()

### 

### 當發生Exception,印出訊息,跳到該Debug的ISR,並且進入Kernel Mode,解決完之後跳回User Mode

### ExceptionHandler()

### 

### System call的值在Reg2,以type儲存起來,接著根據這個type值看是哪種system call,此處是SC\_Halt.

### SysHalt()

### 進入到ksyscall.h,系統的system call都寫在這了

### 

### 此實作的system call寫在interupt.cc的Halt()裡了

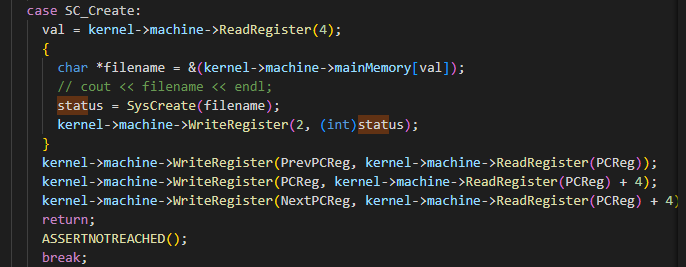
### Halt()

### 

### 系統print出訊息後,把整個kernel結束掉,程式就停止了

## (2)SC\_Create

### ExceptionHandler()



### 把要創立的file名字放入到val變數(原先儲存在Reg4),SysCreate詳細在ksyscall.h,並且把status更新到Reg2(system call的值),最後更新program counter的值.

### SysCreate()

### 

### 此function在ksyscall.h,System call的實作在filesys.h

### FileSystem::Create()

### 再往下追蹤OpenForWrite可發現在lib的sysdep.cc,可以看到會呼叫Unix系統的open() system call,也就是stub檔案的來由.最後Create(),根據回傳的fileDescriptor建立檔案,成功為1,失敗為-1

## (3)SC\_PrintInt

### ExceptionHandler():

### 

### 一樣會判斷是甚麼樣的system call類型,此處是PrintInt,讀出傳入的參數進入SysPrintInt()

### SysPrintInt():

### 進到ksyscall.h 看到必須進入synchConsoleOut.cc

### q

### SynchConsoleOutput::PutInt() && SynchConsoleOutput::PutChar()

### 

### 

### 使用lock,只有獲得鎖的process可以進入同步區,其他process必須等待,

### 將輸入的整數轉成string並在最後加入換行以及終止字元,

### consoleOutput->PutChar()主要是控制硬體輸出在螢幕上,

### waitFor->P() 讓後面還沒做完的字元先等待

### ConsoleOutput::PutChar():

### 

### 寫入file,當寫入時候flag為True,此時不能處理其他事情.

### Interrupt::Schedule()

### 

### 在Interrupt.cc底下,tocall是interrupt執行的對象,when是指interrupt發生的時間,type是指產生interrupt的硬體設備,最後把此interrupt放入pending queue裡.

### Machine::Run()

### OneInstrucion會把解碼好的instruction放入MIP模擬CPU執行指令

### Interrupt::OneTick()

### 

### 目的是要模擬系統往前一個時刻

### 檢查是否有等待中斷：程式碼會關閉中斷（turn off interrupts），然後檢查是否有等待的中斷需要處理。如果有，就會開啟中斷（re-enable interrupts）以處理中斷。

### 如果當前的Thread用完了time slice,kernel會釋放現在的Thread並調度新的Thread

### Interrupt::CheckIfDue()

### 檢查所有pending queue裡的interrupt是否全部解決了,解決完return True.