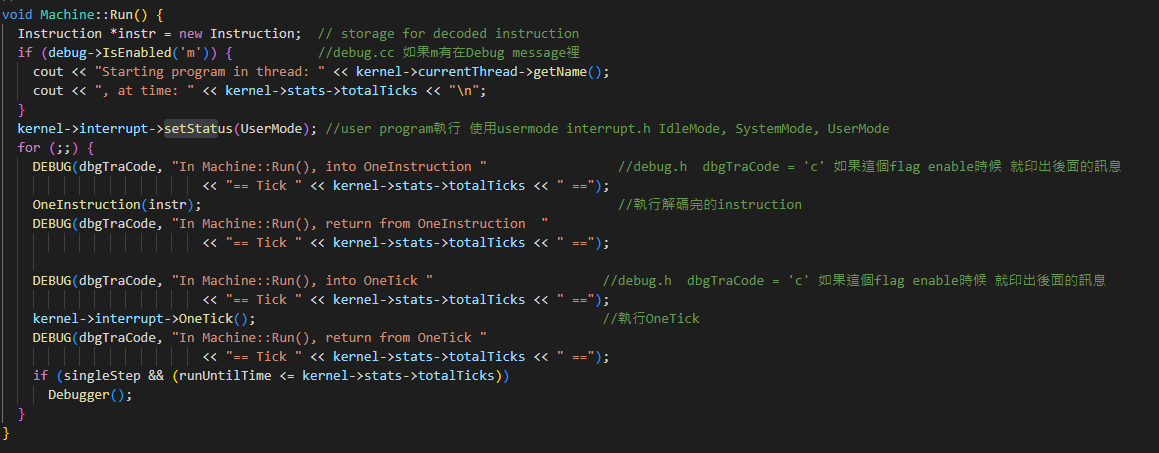
## (1)SC\_Halt:

**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

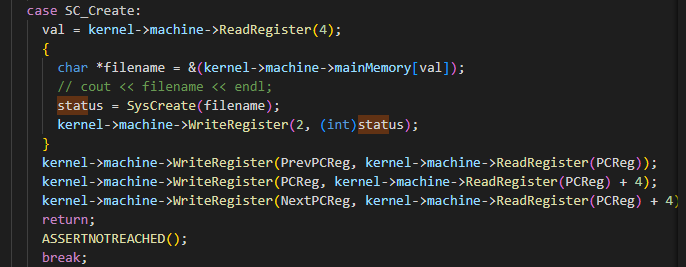
### Halt()

### 

### 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