**MP1 Report**

組別:os21team34

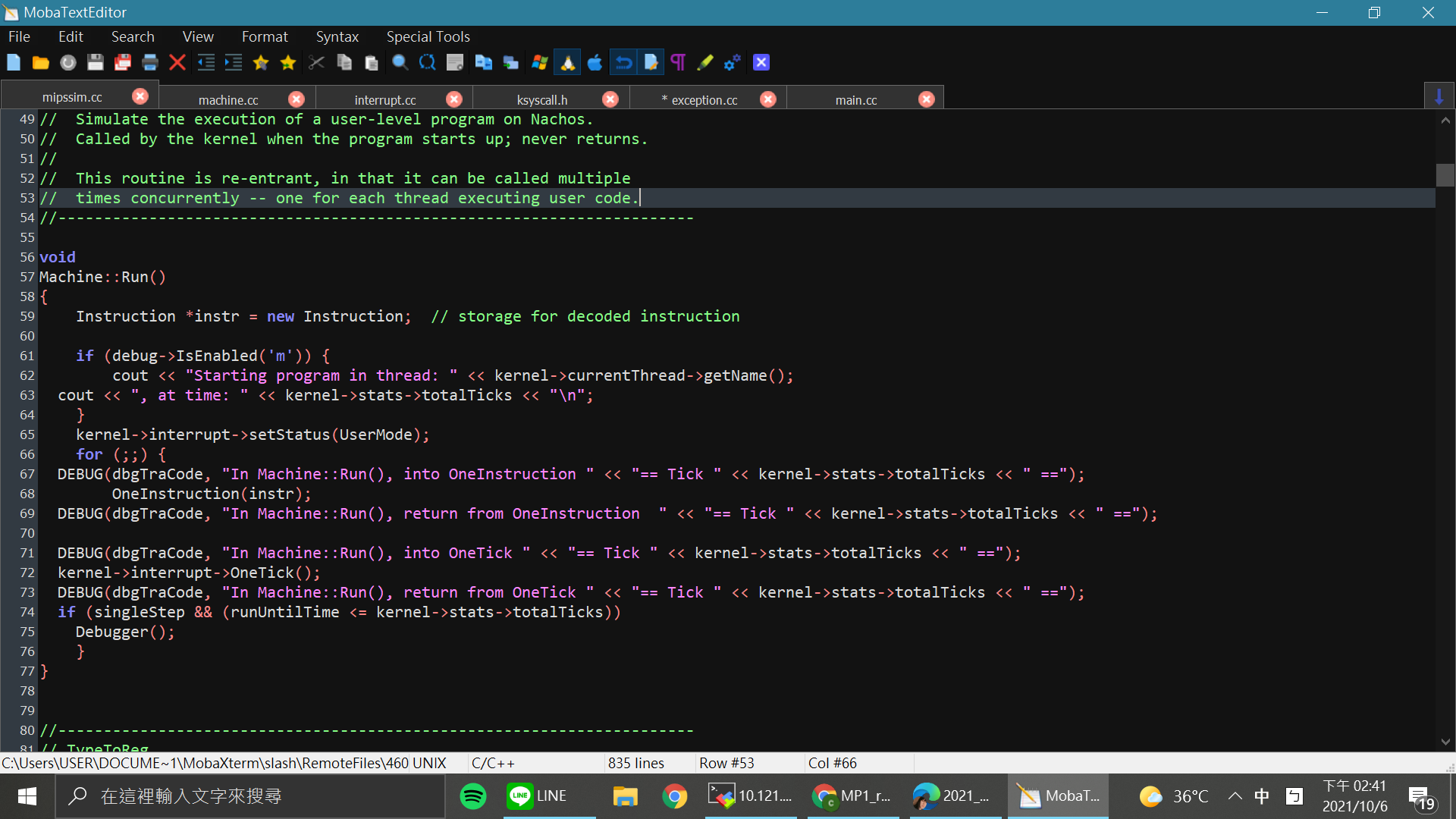
成員:江咏宸108030001、蔡懿晨107070035

| Contribution | | | |
| --- | --- | --- | --- |
| Tracecode(a)(b)與相關report | 蔡懿晨 | Tracecode(c)與相關report | 江咏宸 |
| Implement(a)(d)與相關report | 江咏宸 | Implement(b)(c)與相關report | 蔡懿晨 |

**Part1:Trace code**

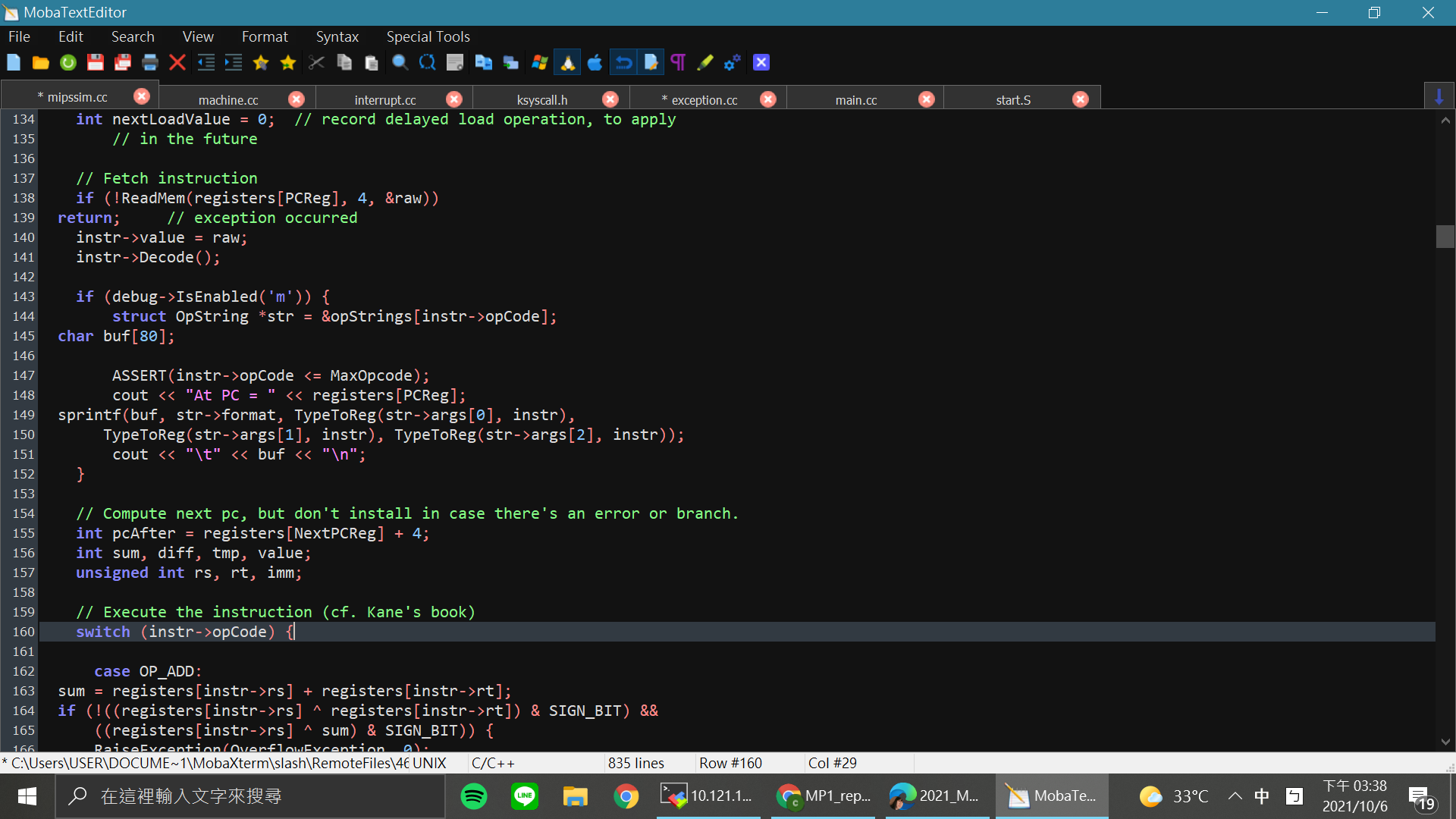
**(a)SC\_Halt(Linux系統關閉)**

* Machine::Run()

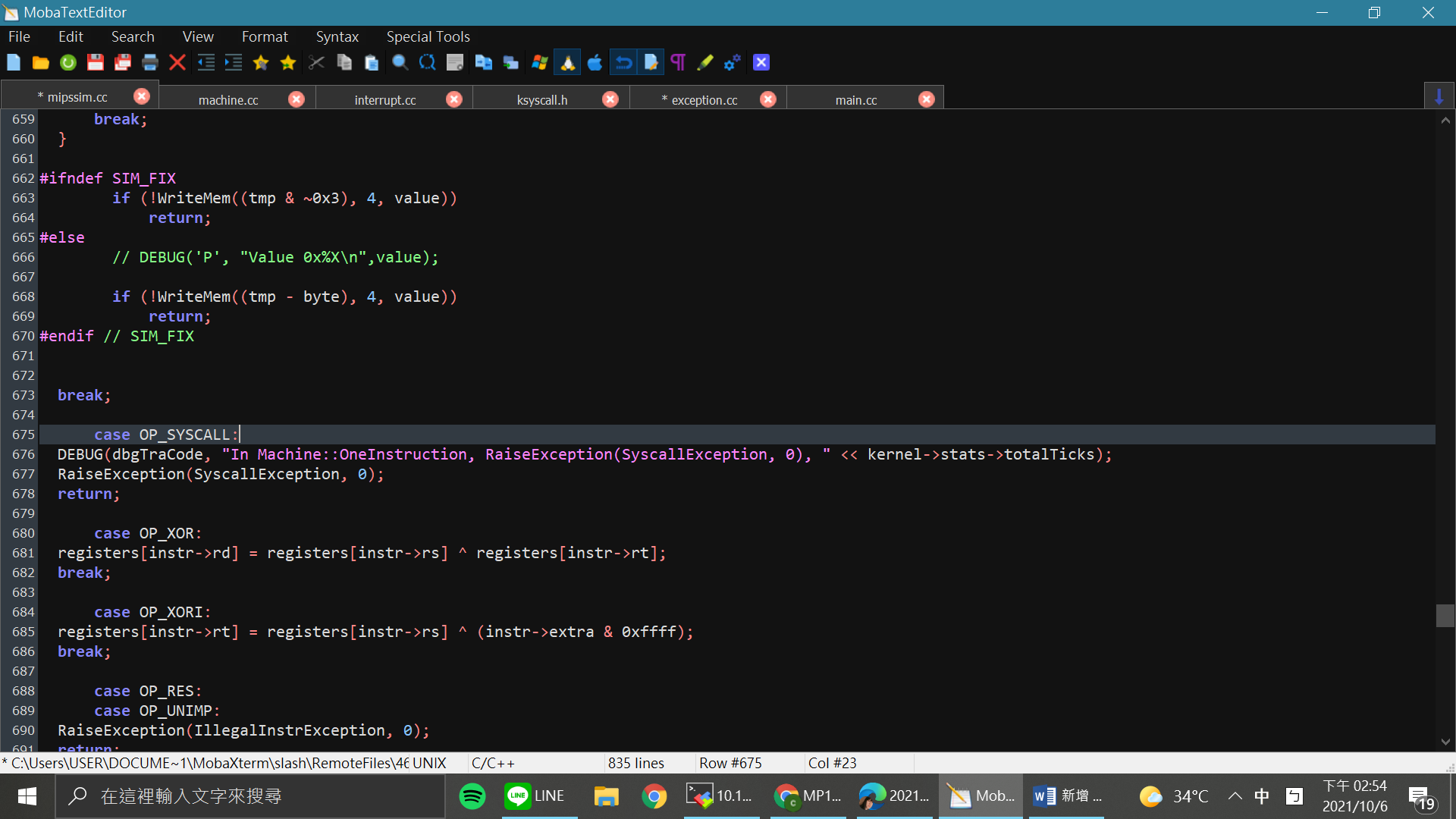


1. 目的：模擬user-level program
2. an user instruction正在執行，將mode改成usermode
3. 在for迴圈裡用呼叫OneInstruction，來模擬逐條執行指令的行為
4. 執行完指令會呼叫OneTick()，更新執行時間並檢查是否有pending interupts (Advance simulated time and check if there are any pending interrupts to be called. )
5. Debugger是user program 的 Primitive debugger

* Machine::OneInstruction()

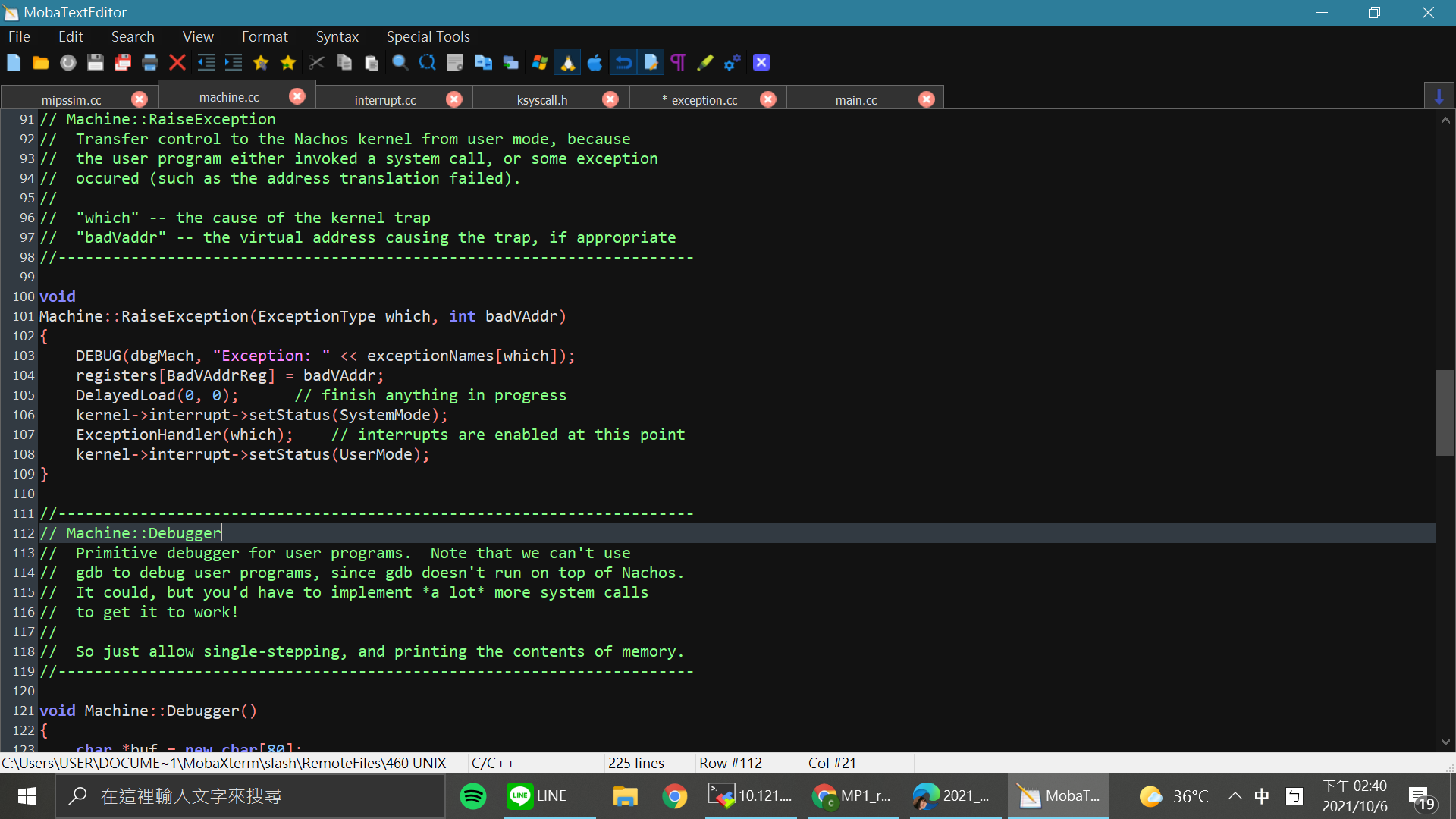


1. 目的：在user的角度執行program的一個指令
2. OneInstruction收到instr後做decode
3. 開始判斷屬於哪一種指令，進入switch

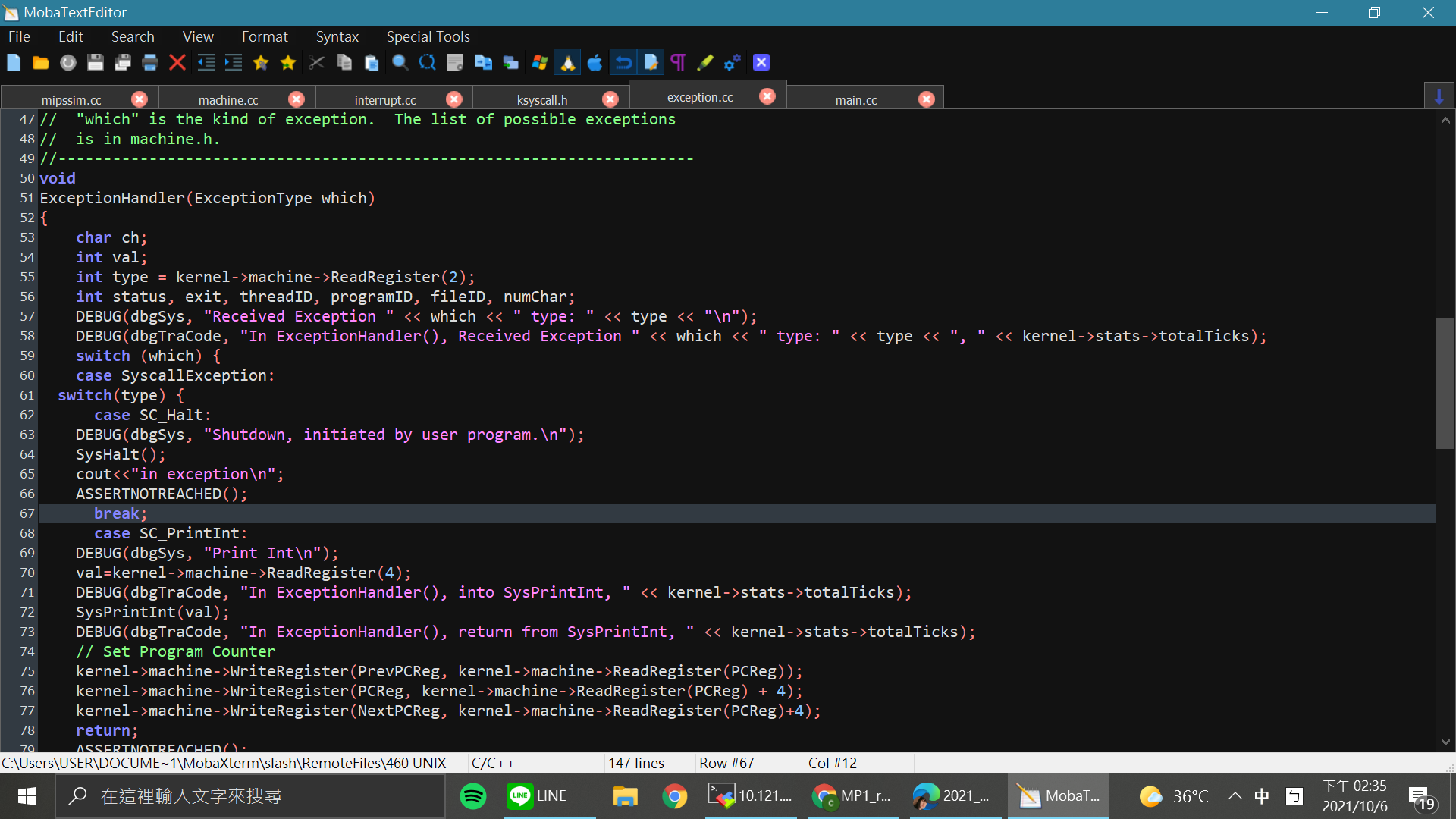


1. 判斷出是syscall類型的指令
2. 呼叫RaiseException並傳遞syscallException的種類跟此trap的the virtual address(此時是0)
3. 傳遞參數的方式是pass by value，因為SyscallException和0都是數值(SyscallException可以從machine.h中看到在enum裡面)

* Machine::RaiseException()

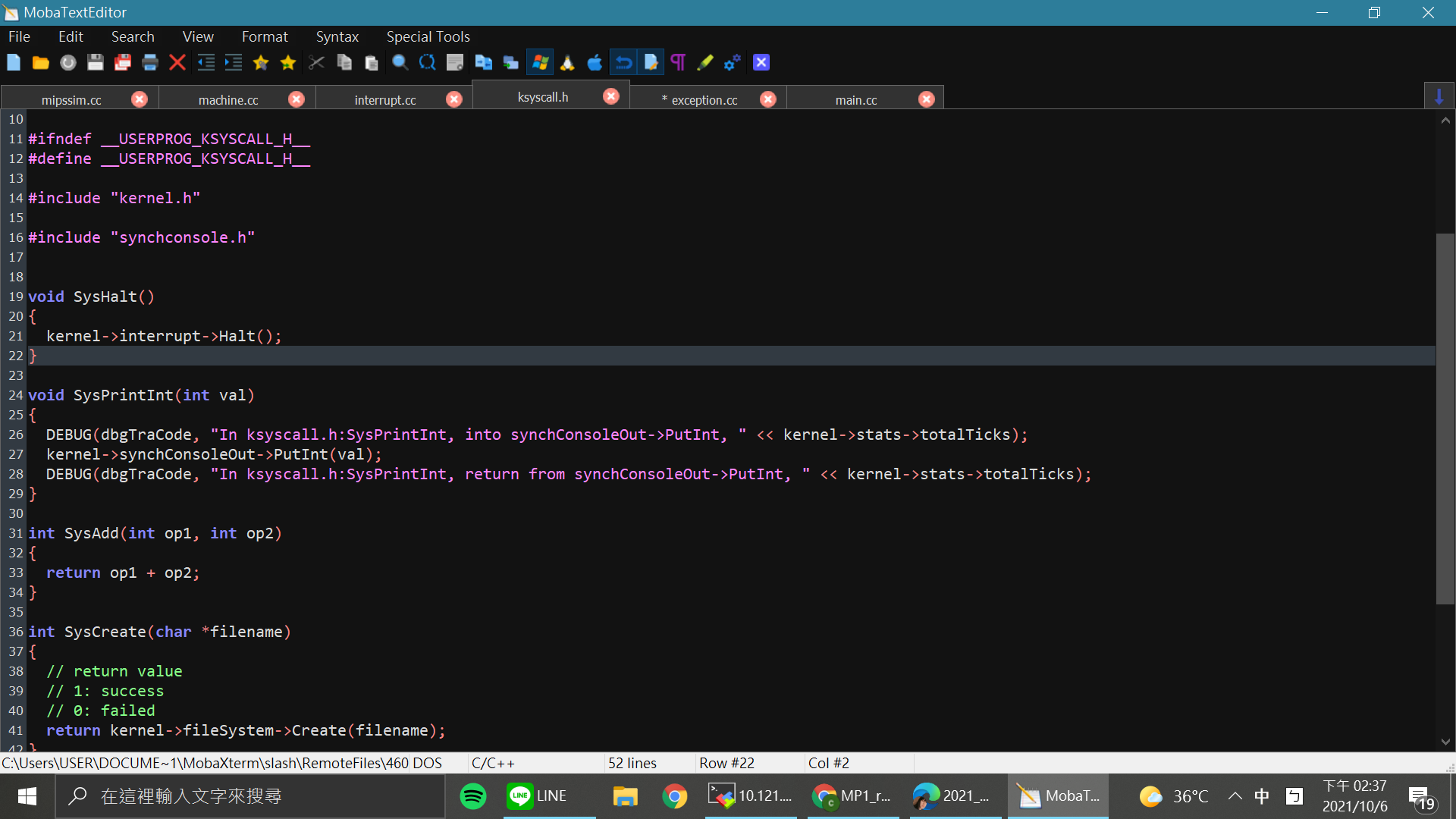


1. 目的：當user program觸發system cal 或發生了exception時，引發一個 interrupt，將控制從user mode切換到kernel mode去處理
2. 引發interrupt，which是interrupt的type，badVAddr是引發此trap的the virtual address
3. 轉為kernal mode(systemmode)
4. DelayLoad是模擬完成該指令的一個delay，
5. 呼叫ExceptionHandler，並傳遞接收到的interrupt type
6. interrupt結束轉回usermode
7. 這裡傳遞參數的方式是pass by value

* ExceptionHandler

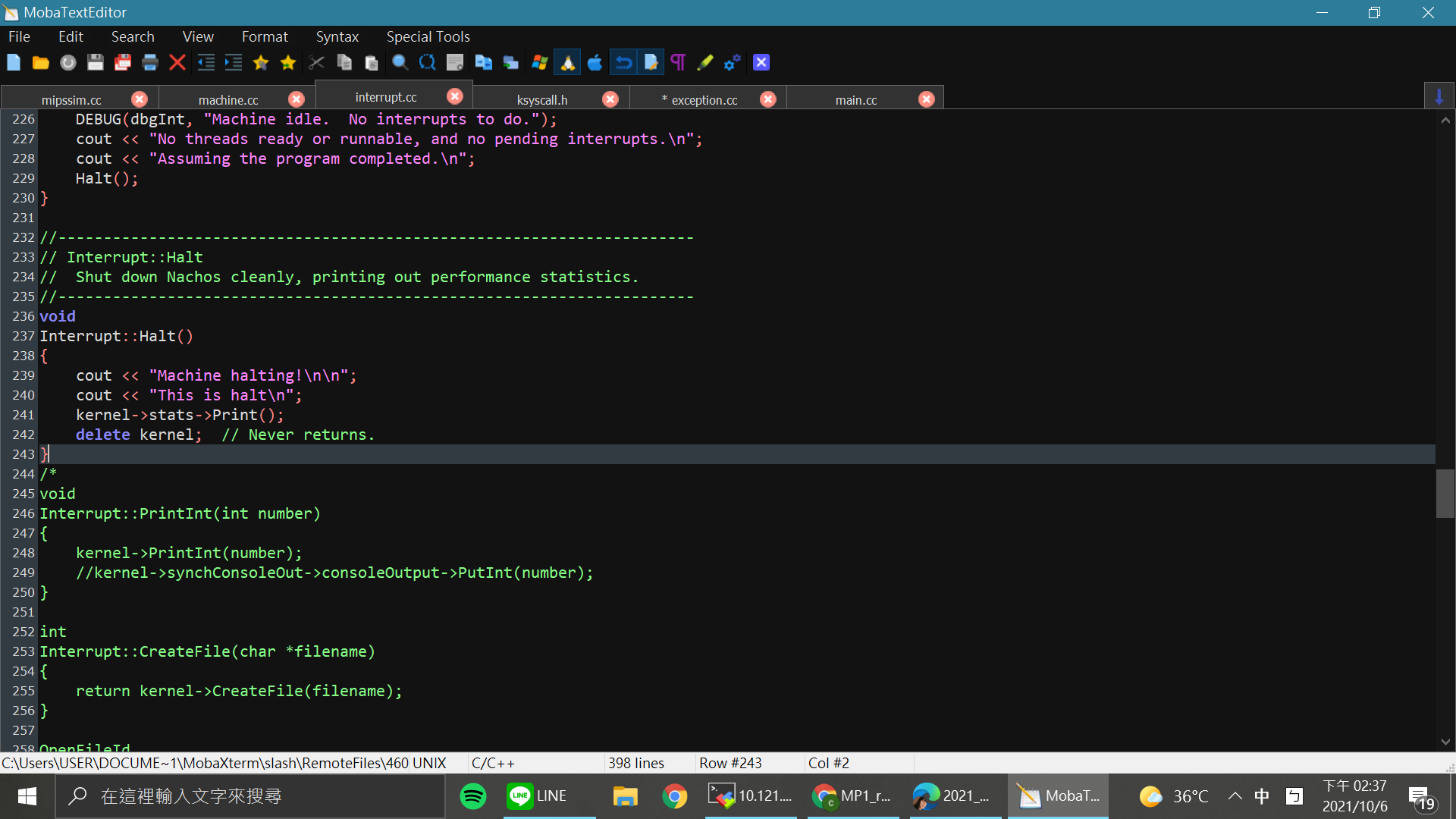
1. 目的：在kernel mode處理呼叫system call引起的interrupt
2. 從2號reg取出type，判斷which是system call 還是 exception
3. 當syscallexception為SC\_Halt時，呼叫sysHalt()
4. 這裡不傳遞參數

* sysHalt



1. 目的：執行Halt()
2. 不傳遞參數

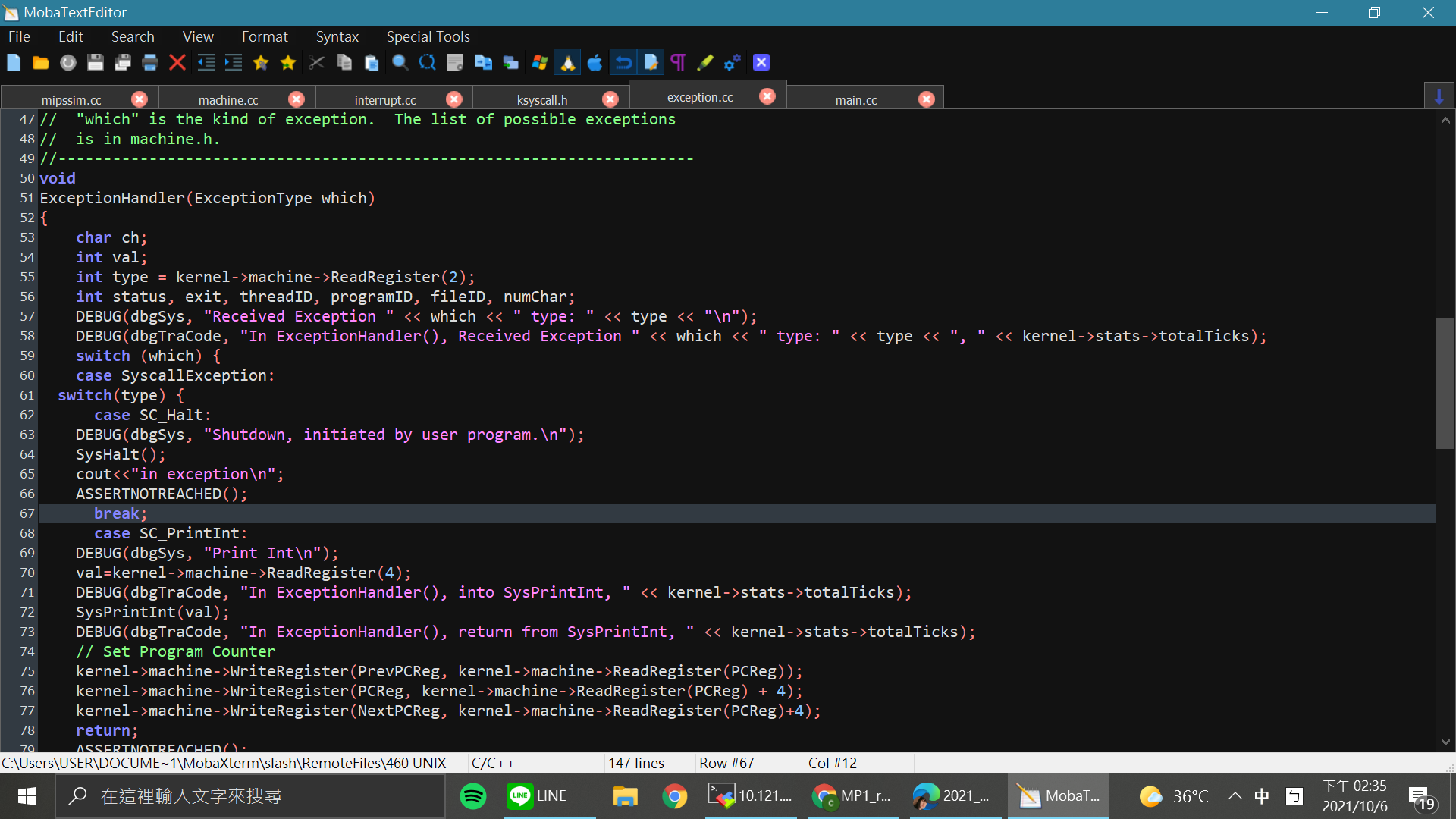
* Halt



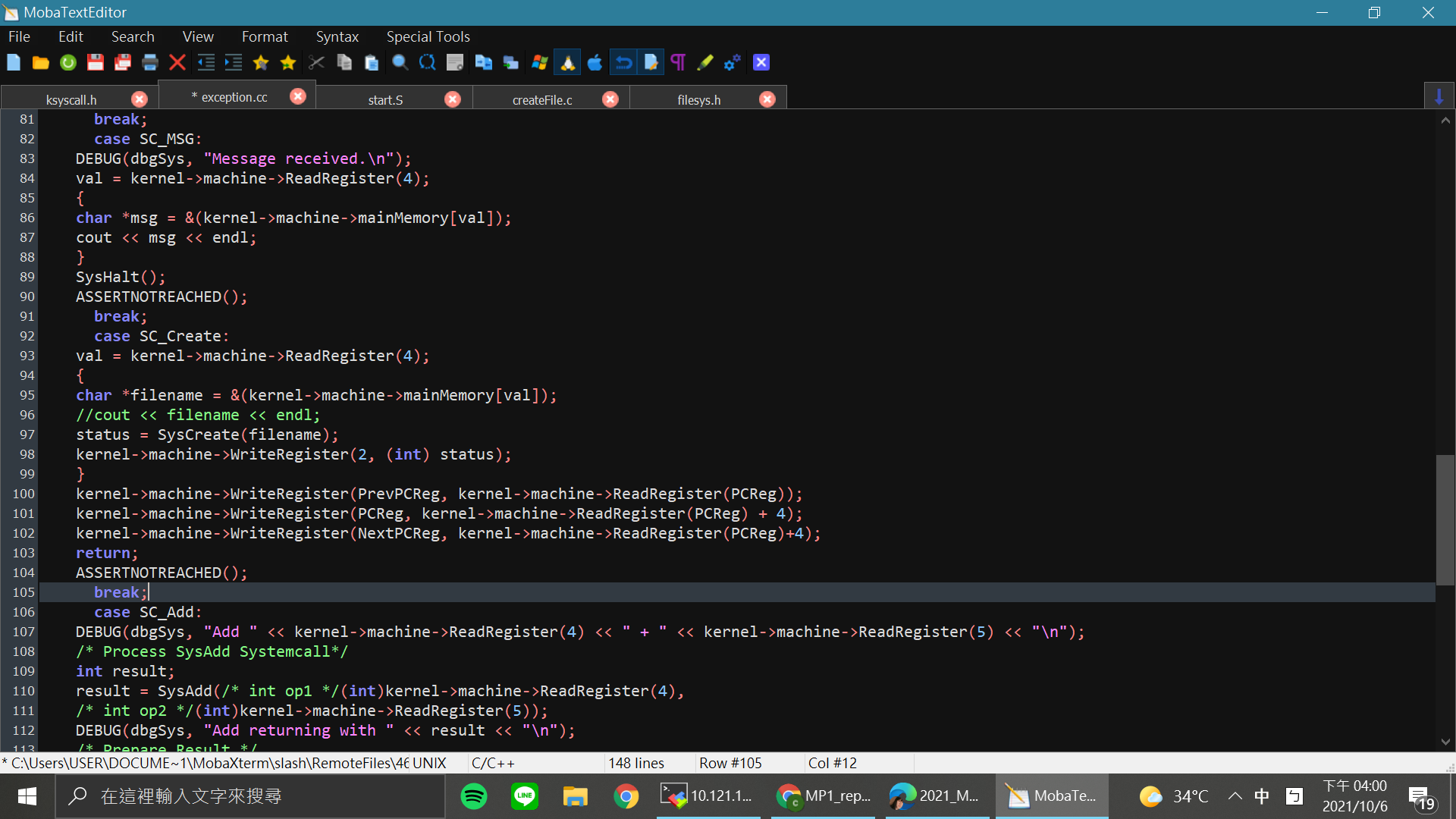
1. 目的：delete kernel，終止程式
2. 執行此function
3. print details of execution time
4. 執行delete kernel，刪除kernel，同時代表程式停止
5. 不傳遞參數

**(b)SC\_Create:**

* ExceptionHandler()

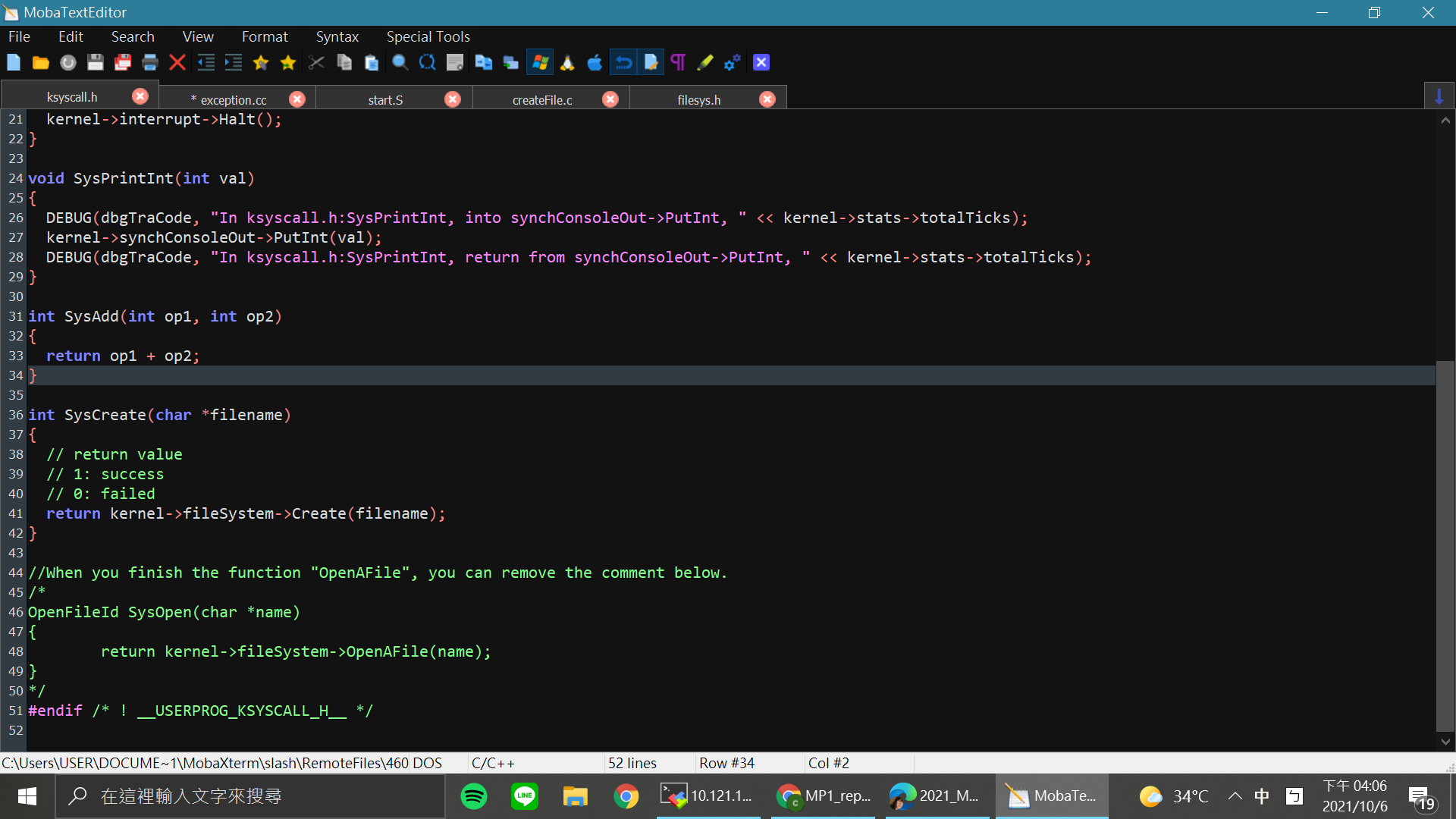


1. 目的：在kernel mode處理呼叫system call引起的interrupt
2. 從2號reg取出type，判斷which是system call 還是 exception



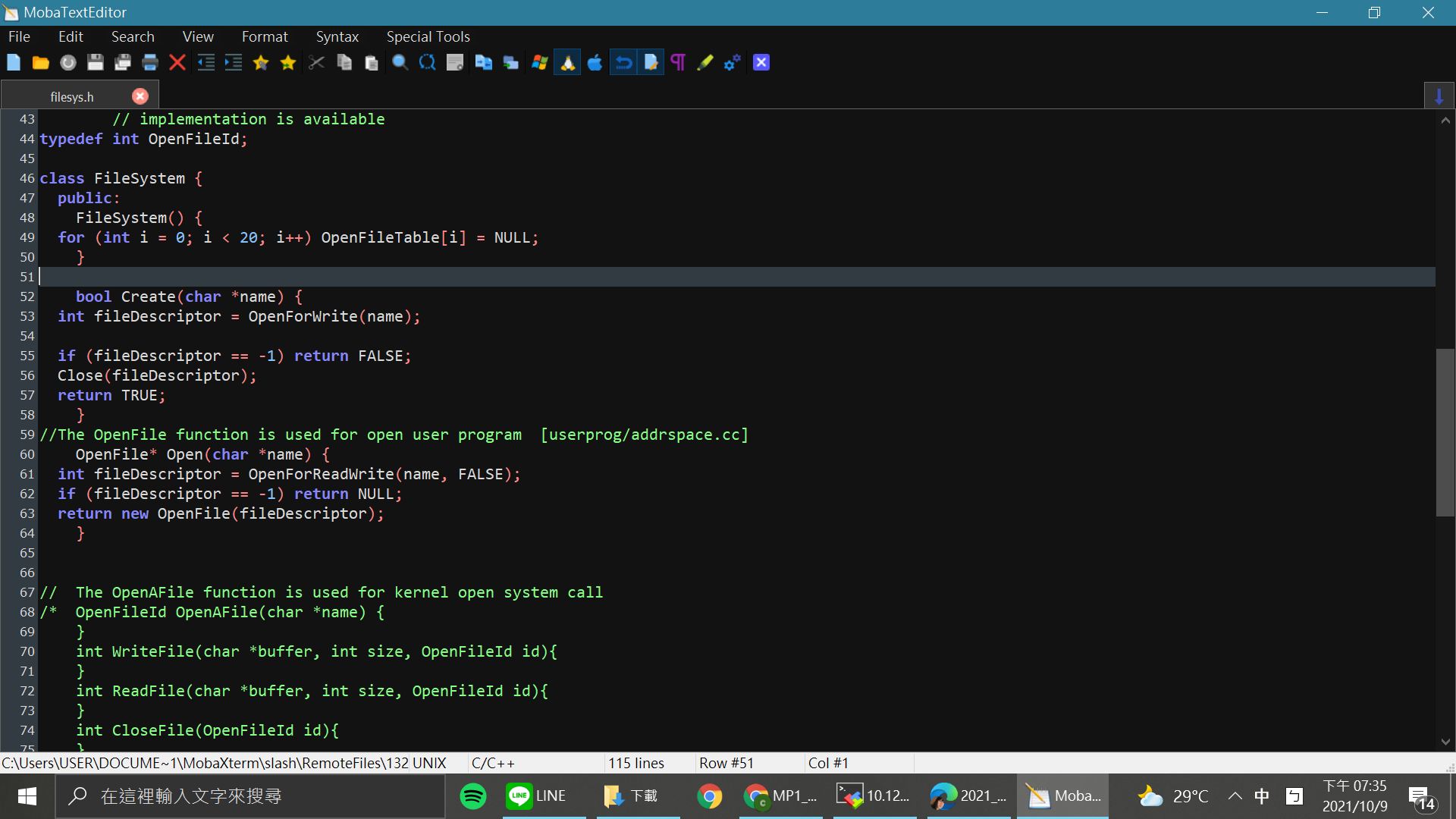
1. 當syscallexception為SC\_Create時，從4號reg取出filename
2. \*filename代表是指標，是存地址，之後還要到mamory取得實際的filename
3. 呼叫sysCreate
4. 把結果存到reg$2
5. 更新存Program Counter的reg的值

* sysCreate()



1. 目的：呼叫create

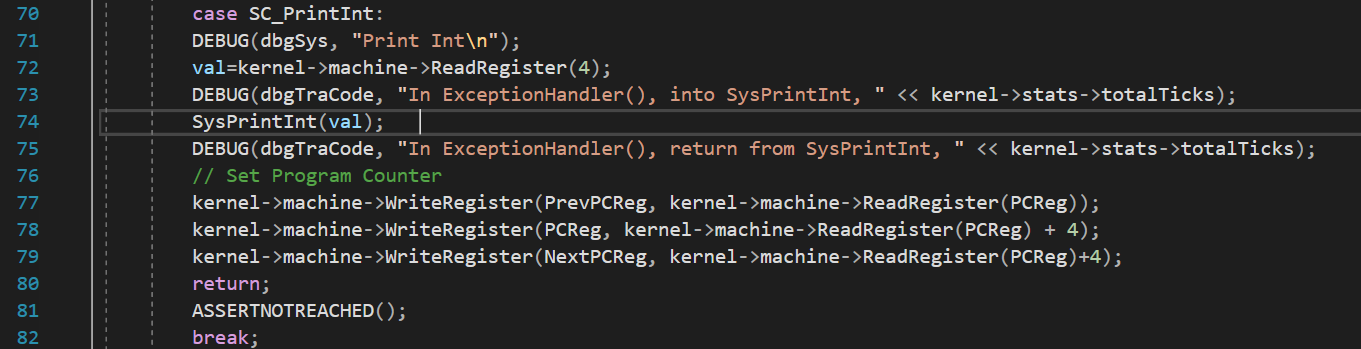
* Filesystem::Create()



1. 目的：建立file並回傳結果
2. 進入Create 呼叫了C++內建的OpenForWrite函式，完成建立檔案，並將name(filename)作為檔案之檔名
3. 當fileDescriptor為-1就代表失敗
4. 建立完成，將file關閉，並且return True

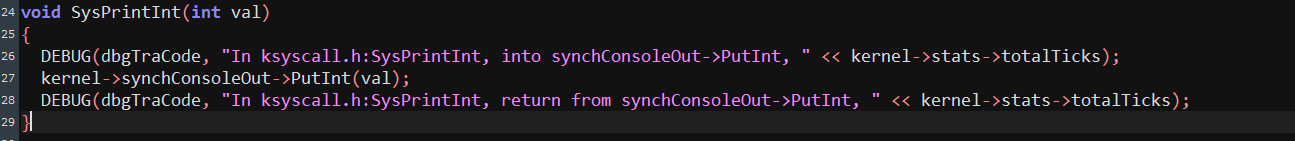
**(c)SC\_PrintInt:**

* ExceptionHandler()

****

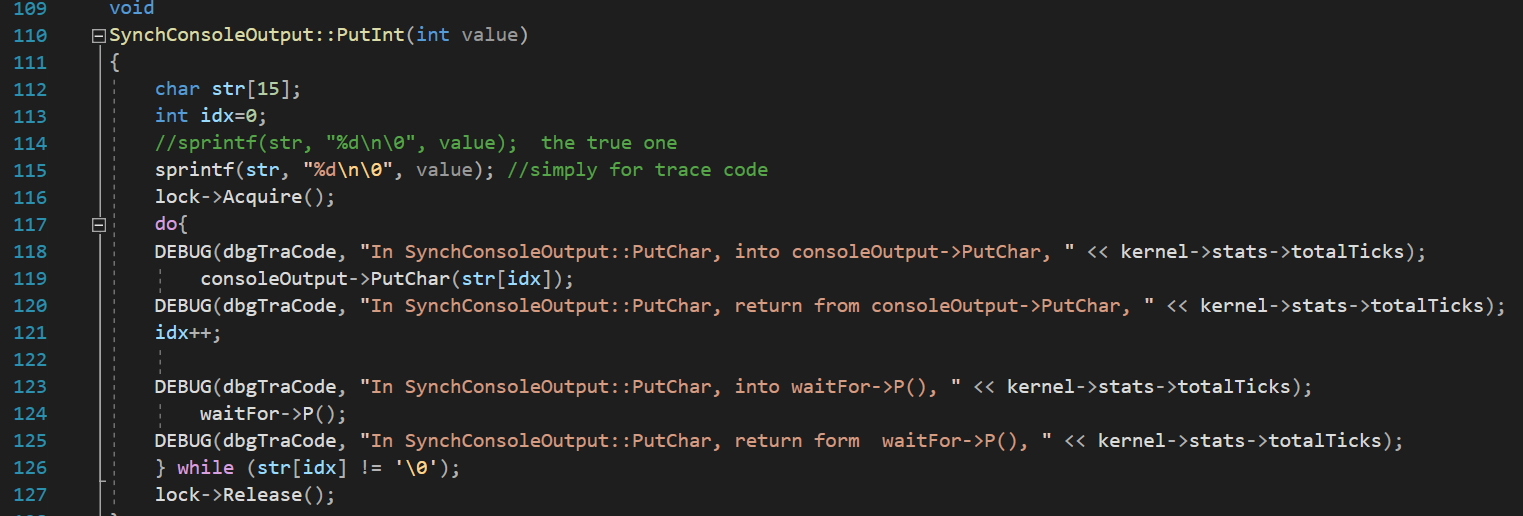
從register(4)讀取值並放到val這個變數並呼叫SysPrintInt(val)。再來用WriteRegister()來更新PrevPCReg,PCReg,NextPCReg，這樣下次才不會重複呼叫同個指令。

* SysPrintInt()



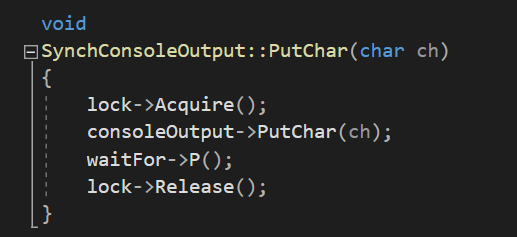
利用kernel呼叫sysConsole(class:SynchConsoleOut)來執行PutInt(val)

* SynchConsoleOutput::PutInt()



利用116和127的lock->Acquire()和lock->Release()來確保執行output時候一次只有一個writer。而在117~126的迴圈中每次把str的值交給consoleOutput (class:ConsoleOutput)來處理並idx++，同時呼叫waitFor(class:Semaphore)->P()。Semaphore有兩個指令P()和V()，P()會等待semaphore的value>0才會減少，V()會增加semaphore value且再有必要時會呼叫等待P()的thread來讓consoleOutput可以處理下個output。在synch.h裡顯示Semaphore用List<Thread \*> \*queue來實作。

* SynchConsoleOutput::PutChar()



PutChar()的實作比較單純，因為一次只output一個char。但同樣有lock和waitFor參與控制以及把執行交給consoleOutput處理。

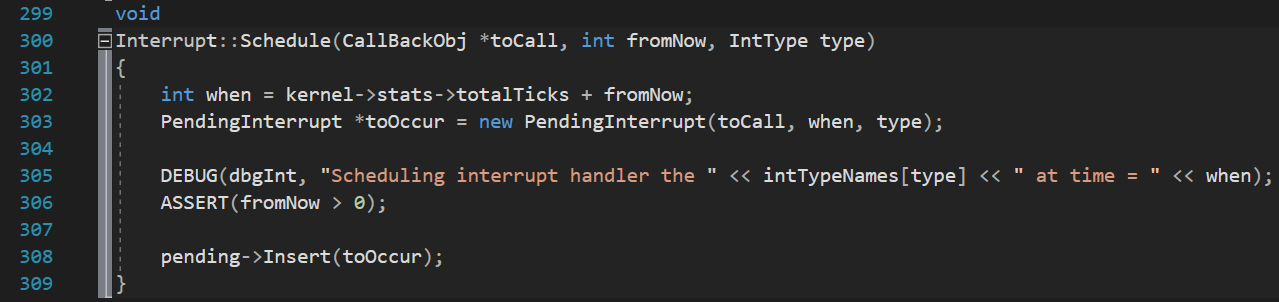
* ConsoleOutput::PutChar()



首先會確保putBussy==FALSE來確保當前沒有其他PutChar()在執行。

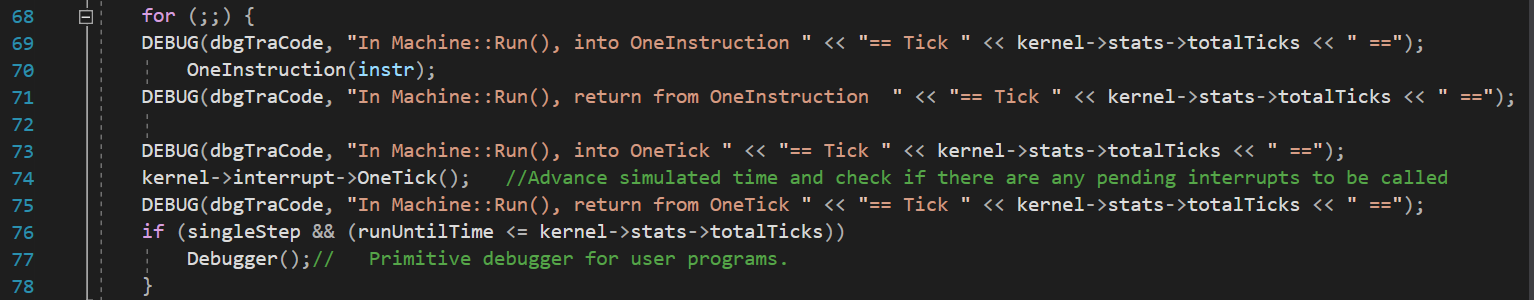
利用sysdep.cc提供的function:WriteFile()來把ch寫入writeFileNo裡面(stub)。再來呼叫Schedule()來register schedule events (interrupt)。

* Interrupt::Schedule()



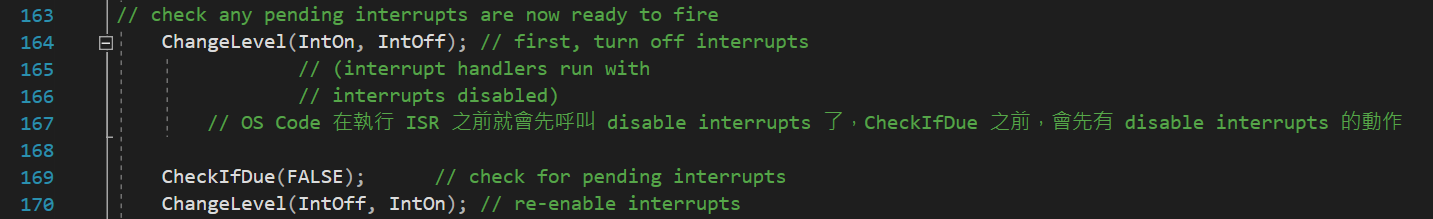
該方程式第一個參數代表註冊event的身分(此為ConsoleOutput)，第二個和第三個為花費時間(ConsoleTime)和類型(ConsoleWriteInt)。302行算出該event發生時間(when)，303和308行則是 initiate 該event 並呼叫Interrupt中SortedList<PendingInterrupt \*> \*pending的Insert()來加入該event。

* Machine()::Run()



74行代表每次執行指令時會去使用OneTick()來模擬時間並檢查是否有pending interrupt要處理。

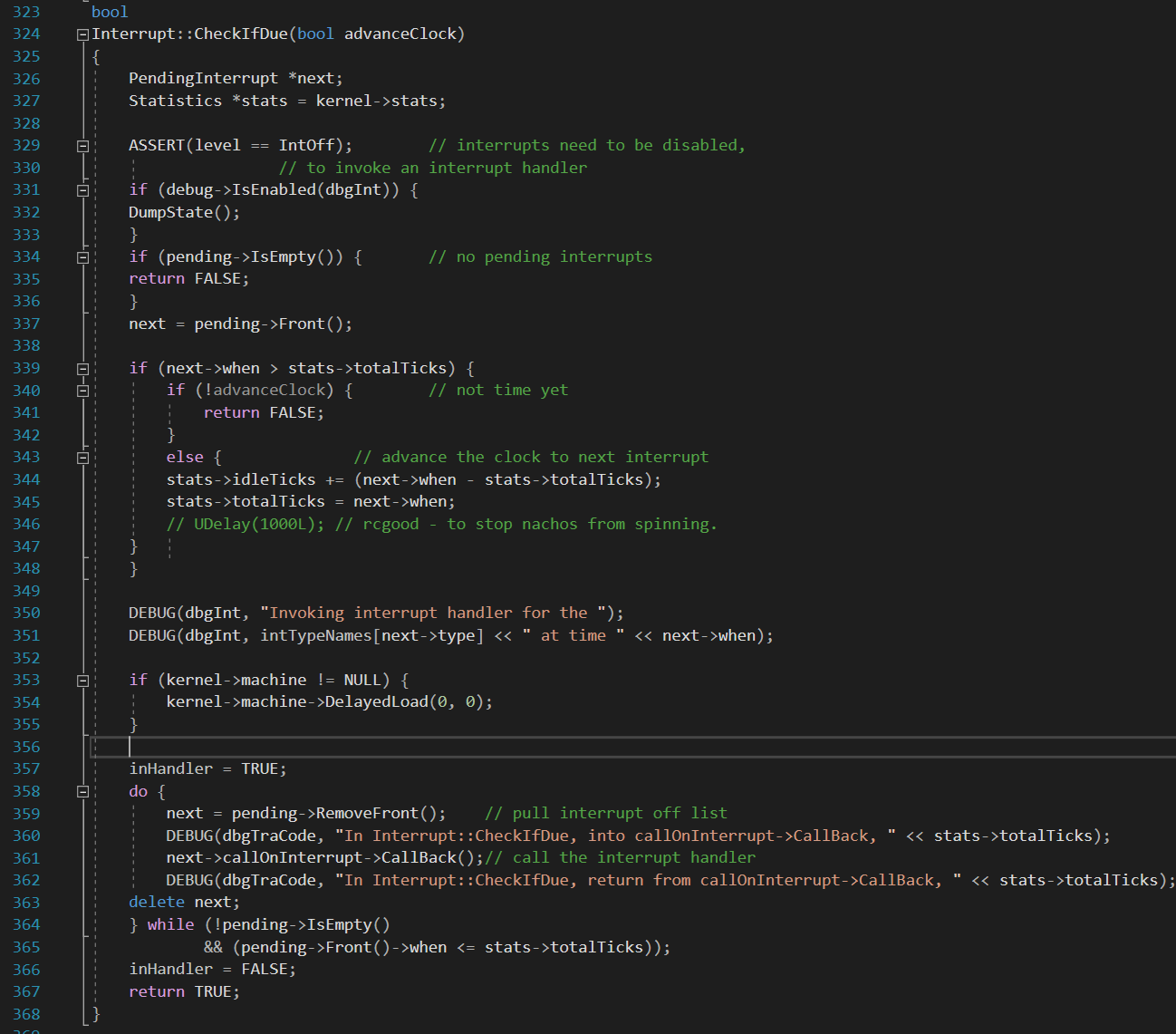
* Machine()::OneTick()



每次在advance time時會去呼叫CheckIfDue()來去檢查是否有Interrupt要執行。

164和170行代表每次檢查時會去disable其他interrupts避免一次執行兩個的interrupts。

* Interrupt::CheckIfDue()



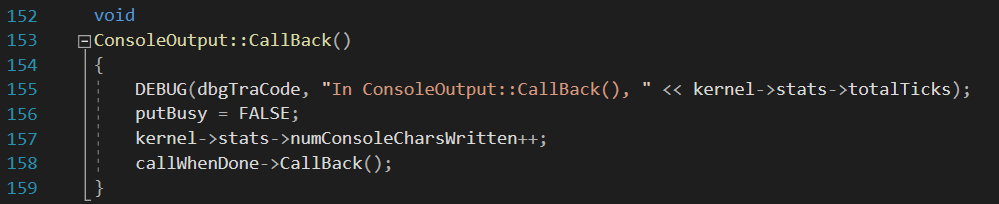
CheckIfDue會return TRUE和 FALSE，代表interrupt被執行與不執行。

329行會先確保在IntOff的狀態(沒有其他interrupts在執行)，334和335行會去檢查當pending是為空時直接return FALSE。326和337會用next(class:Pending Interrupt)去拿列隊裡第一個需要被執行的interrupt。

339~348行會去檢查是否到達該interrupt的執行時間，如果到達(343~347)會把時間advance到這個event，如果沒到達(340~342)return FALSE，339行的條件為next須為未來事件(需要大於當前的totalTicks)。253~355為了模擬delayed load而修改第0個reg為0。

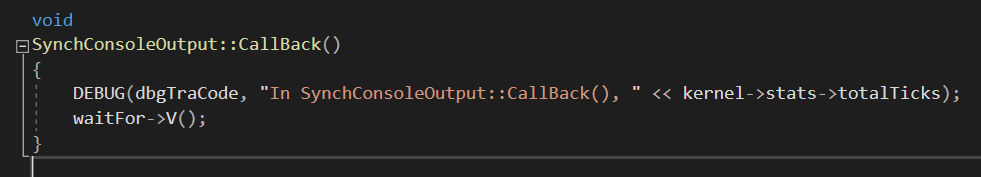
357和366:當inHandler=TRUE帶表正在執行interrupt handler，False代表不執行(處理好了)358~365行為處理interrupt的過程，364和365行代表處理時pending不能空的同時一次只能處理一個interrupt(345行更新totalTicks為next->when)。359和361為取出pending的第一個interrupt並從pending中移除(RemoveFront())，並呼叫next->callOnInterrupt->CallBack()。callOnInterrupt為當初register event的CallBackObj類型(ConsoleOutput)，並其中呼叫CallBack()。363行為執行完後把該interrupt刪掉。最後367行return TRUE為interrupt被正確的執行。

* ConsoleOutput::CallBack()



把putBusy=FALSE代表該ConsoleOutput從忙碌到不忙碌，呼叫stats( Statistics \*stats)->numConsoleCharsWritten++來記錄增加的量。最後呼叫callWhenDone的CallBack()(當初initialize callWhenDone為SynchConsoleOutput )

* SynchConsoleOutput::CallBack()

****

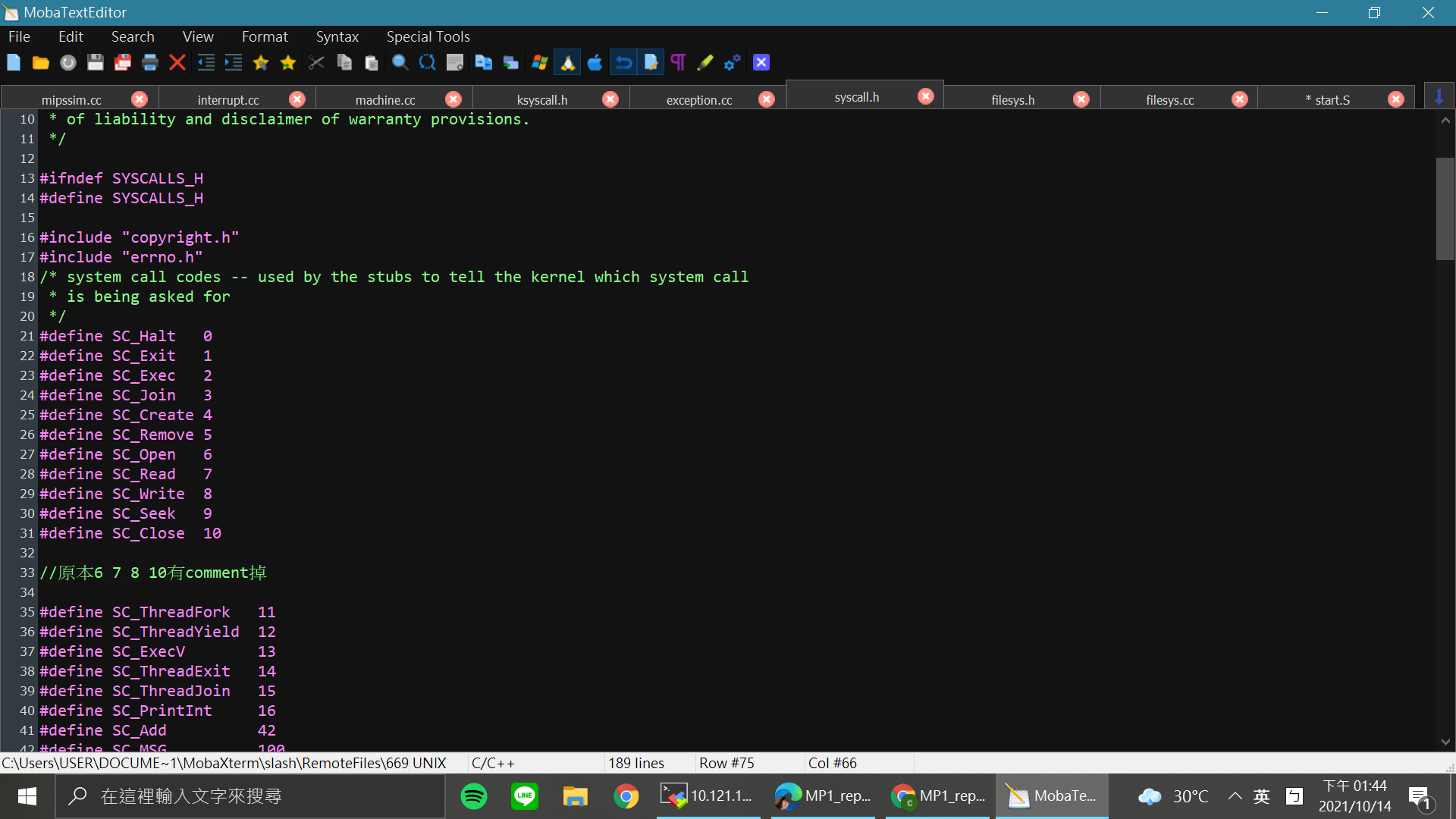
**waitFor(class:Semaphore)->V():**

V()會增加semaphore value，再有必要時會呼叫等待P()的thread，也就是下次output

**part2\_implement**

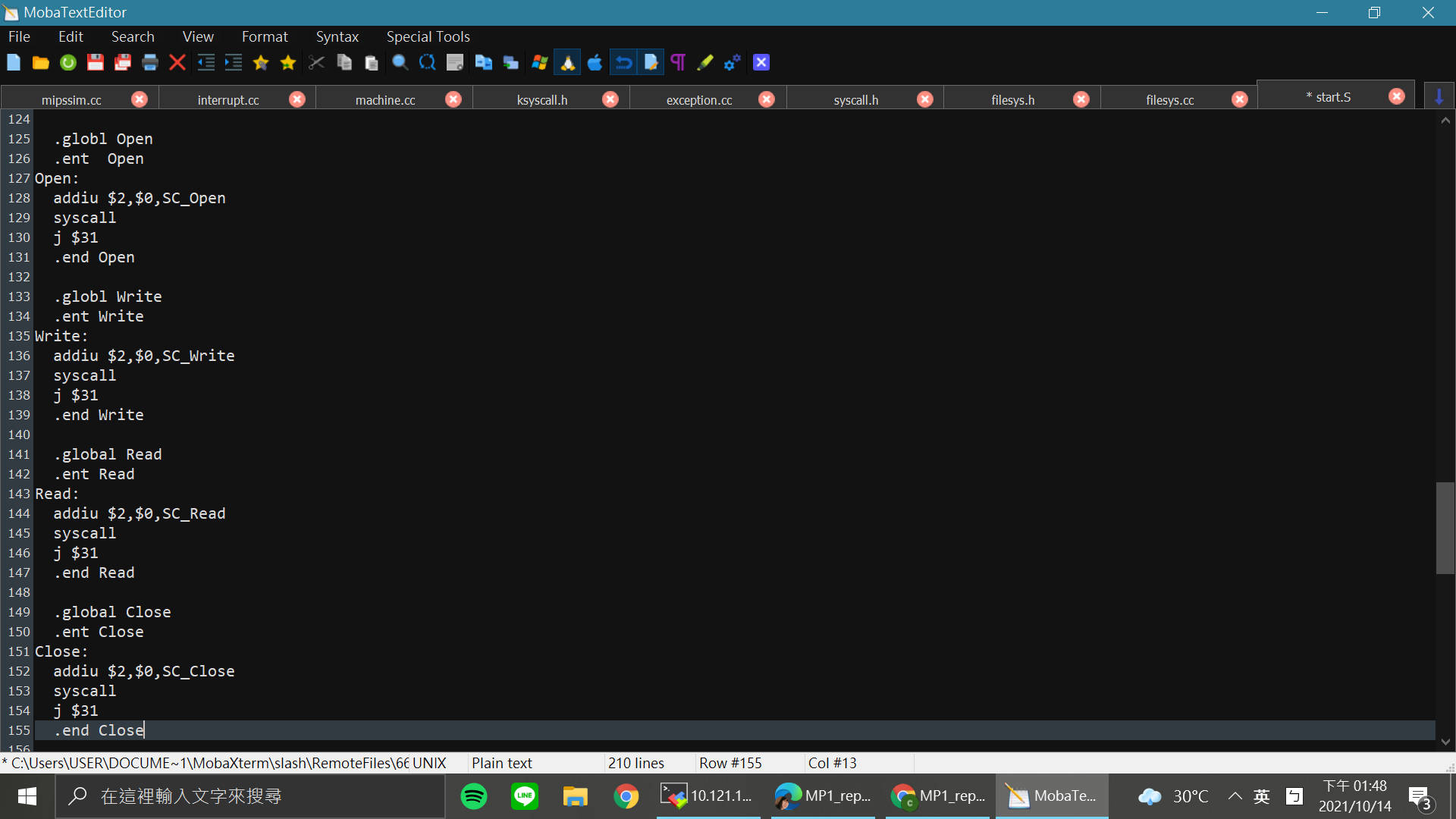
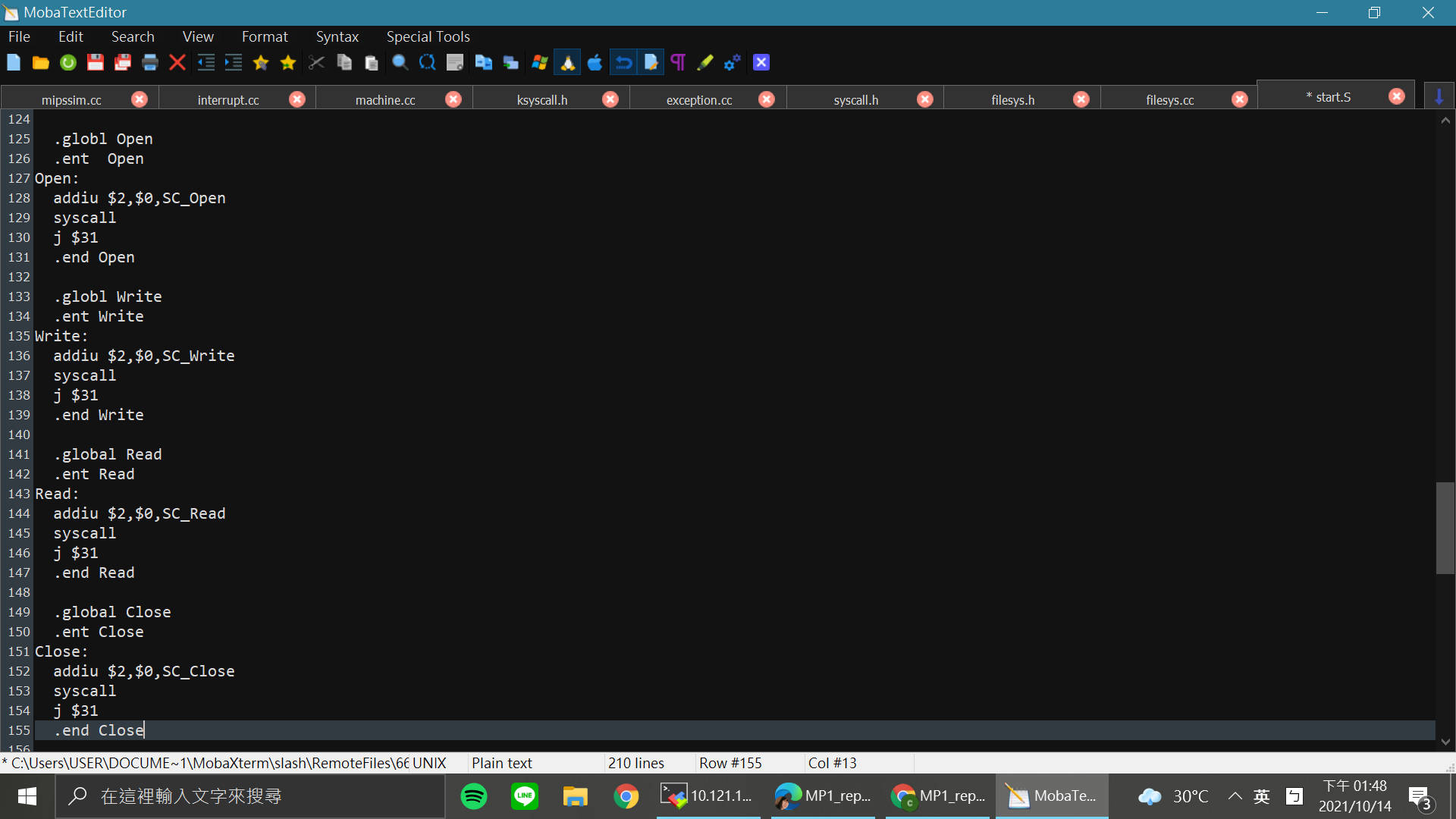
先改了三個syscall會用到的檔案

**syscall.h**



* 原本 6 7 8 10四行有註解，將comment拿掉

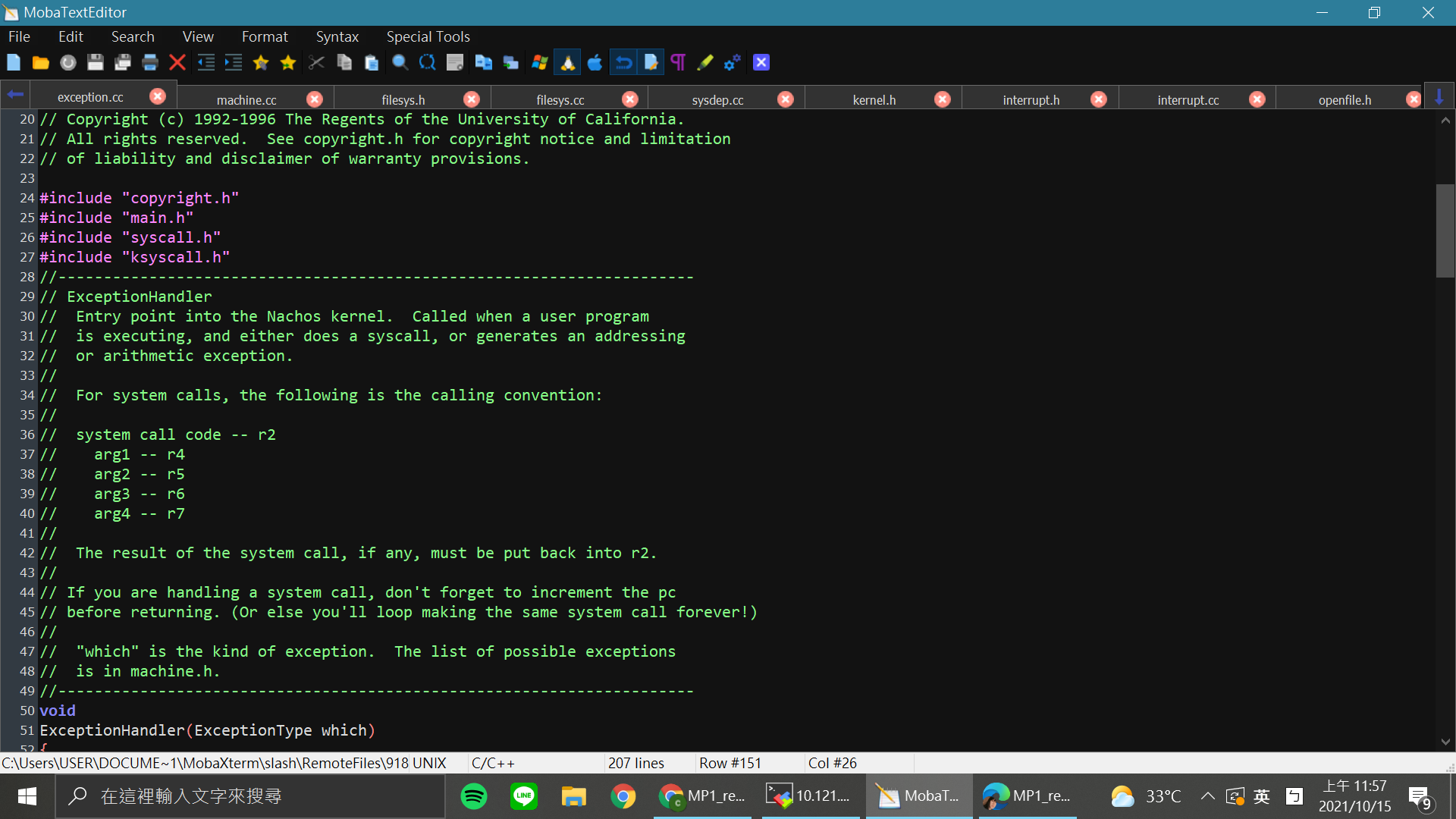
**start.S**



* 以MIPS加入了四個system call的，格式是依照其他已經寫好的system call來改
* 以Open舉例
  + addiu $2,$0,SC\_Open 是把SC\_open放入$2
  + j $31是return address

**exception.cc**

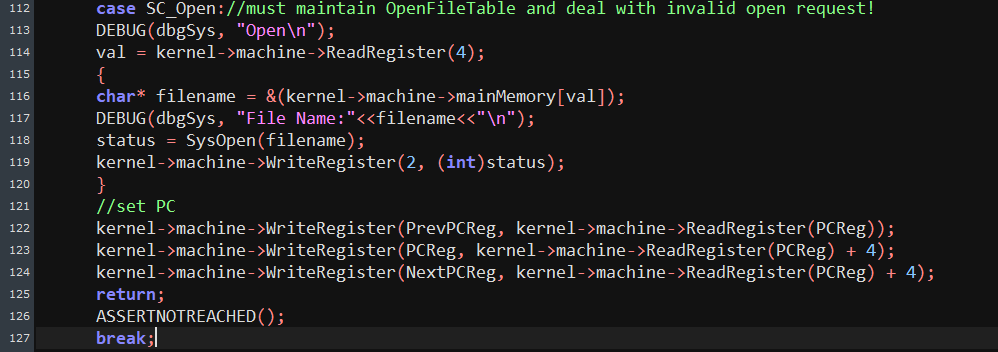
* 由註解可知，參數都是存在register 4 5 6 7裡面



* 新增四個system call指令SC\_Open、SC\_Close, SC\_Write, SC\_Read

以下開始以Function來做介紹

**(a)SC\_Open**

****

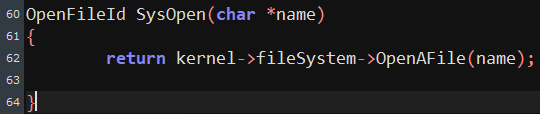
**exception.cc**

A.114行:從register[4]中讀取int val，接著用val把對應在mainMemory中的filename的char pointer讀取出來

B.118行:呼叫system call:SysOpen()(定義在ksyscall.h)，並把結果存在status(int)，並把status寫在register[2]

C.122~124行:更新program counter:PrevPCReg(過去),PCReg(現在),NextPCReg(下個)

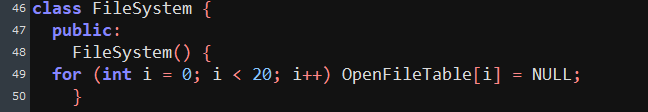
**ksycall.h**

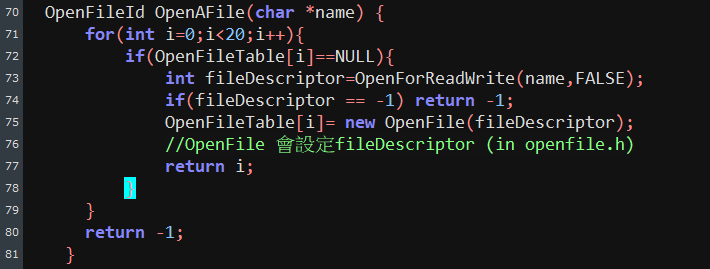


A.在ksycall.h裡面定義SysOpen()，**其實這個function就是OpenFileId Open(char \*name)的實做只是名字不同**，並呼叫fileSystem裡面的OpenFile()，回傳值由status接收(exception.cc)

**filesys.h**

fileSystem為FileSystem這個class產生出來的物件，該class寫在filesys.h裡面

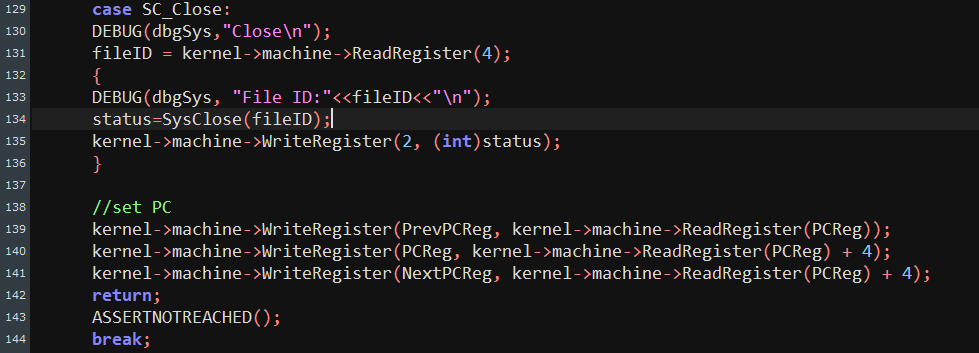






1. 在FileSystem裡面用OpenFile(定義在openfile.h)這個class產生的object pointer array去維護OpenFileTable，初始值為null
2. 在FileSystem裡面實作OpenFileId OpenAFile(char\*name)，return OpenFileId為該新開起的檔案在OpenFileTable的第幾個位置，故71，72行就在找是否有值為null的OpenFileTable，如果找不到就return -1(80行)代表table已滿無法新增被要求開啟的檔案。如果找到就用sysdep.h/sysdep.cc寫好的OpenForReadWrite()來實作stub file的檔案開啟，回傳值給fileDescriptor為該檔案的編號(73行)，由於在這裡是使用OpenFileTable來管理的，故需要建一個OpenFile這樣的object來放置fileDescriptor並當作檔案存取使用(75行)。最後return在OpenFileTable的位置i(77行)，如果73行無法正確開啟該檔案會回傳-1給fileDescriptor，而此時會return -1代表檔案開啟失敗(74行)，可能是invalid or non-exist file。

**(b)SC\_Close**

****

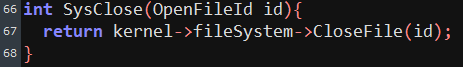
**excecption.cc**

A.131行:從register[4]裡讀取fileID，此值就是OpenFileId的類型

B.134行:呼叫SysClose()(寫在ksyscall.h)，並把執行成功與否的結果(1/-1)傳給status

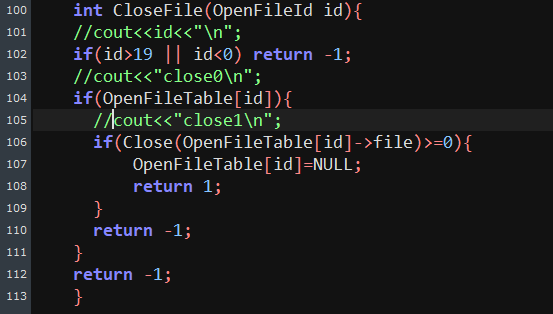
C.135行:把status寫在register[2]

D.139~141行:更新program counter:PrevPCReg(過去),PCReg(現在),NextPCReg(下個)



**ksyscall.h**

A.為了配合格式與方便管理，我在此新增SysClose()來呼叫fileSystem的內建function CloseFile(id)，id為要關掉OpenFileTable的第幾個位置，return 執行結果，**其實這個funciton就是int Close(OpenFileId id)的實做，只是名字不同。**



**filesys.h**

A.102行:排除在OpenFileTable不合法的位置存取，並return -1

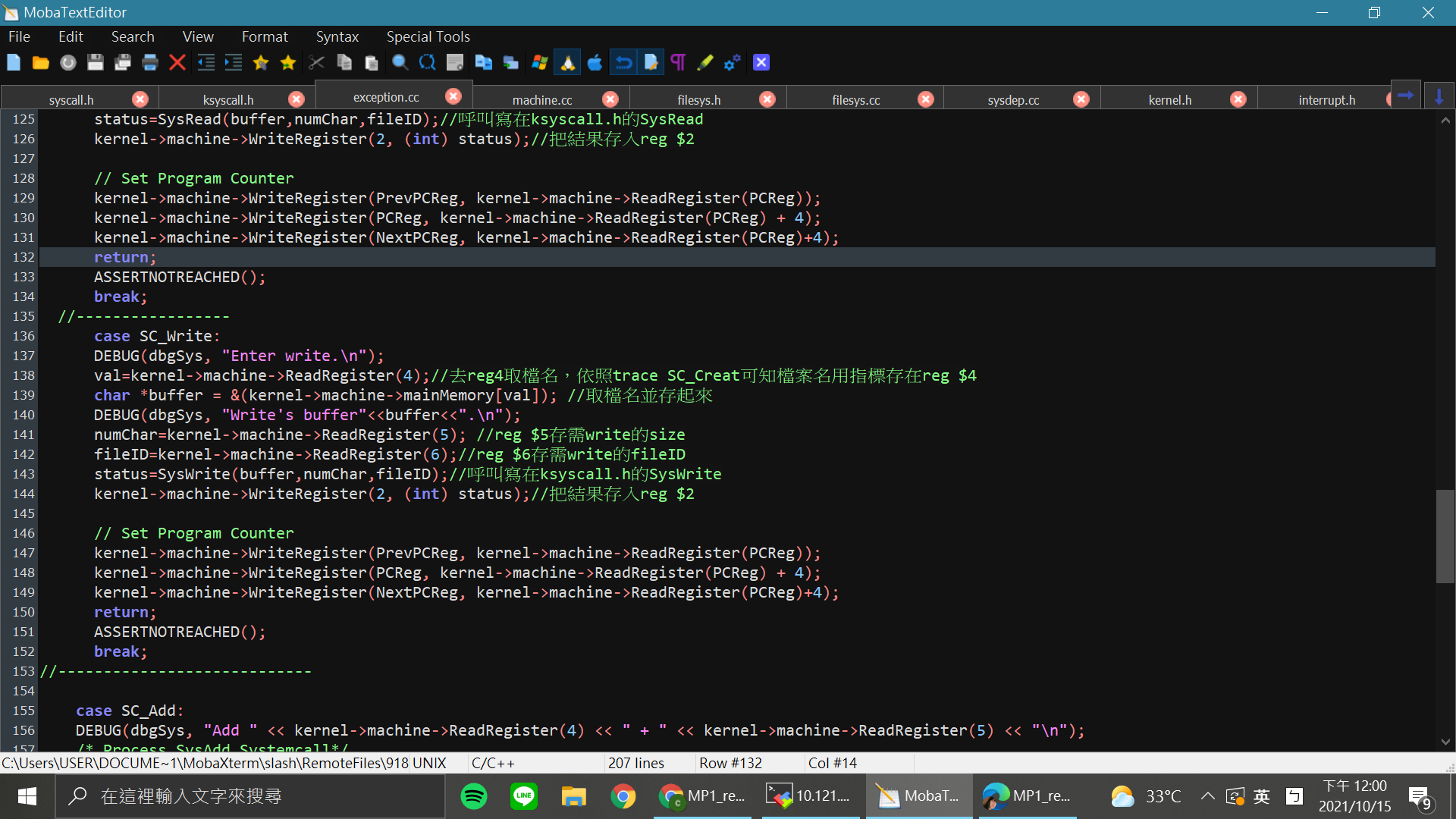
B.104行:確認該位置不等於NULL，否則return -1

C.106行:利用OpenFile[id]->file存取當初存放的fileDescripter並用其當參數呼叫在sysdep.h/.cc 的Close()來執行file stub的關掉檔案，並確保close回傳>=0來確保成功關掉檔案，否則return -1。

D.107~108行:將該位置的pointer指向NULL來重複使用，並return 1代表執行成功

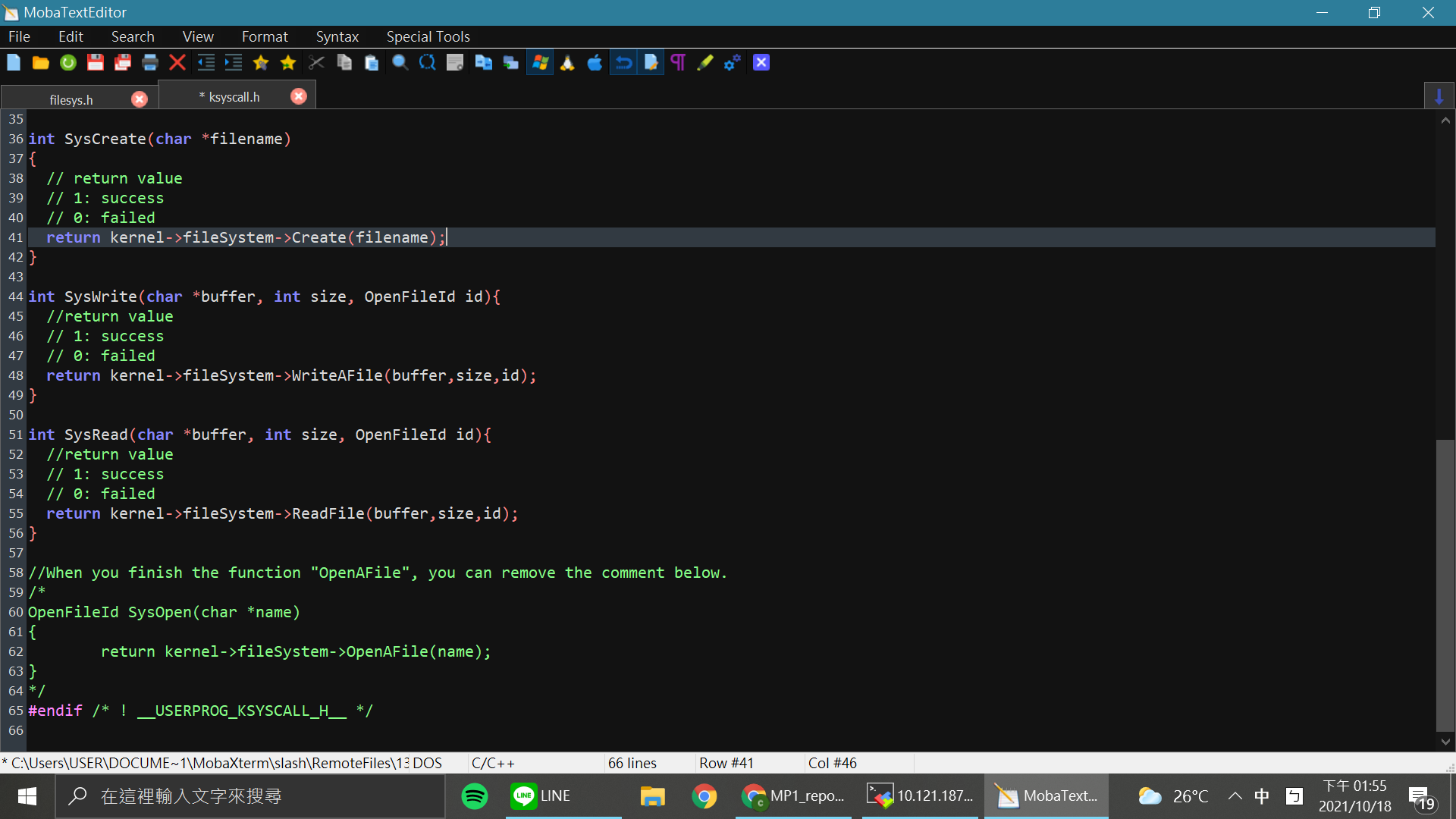
**(c)SC\_Write**

**exception.cc**



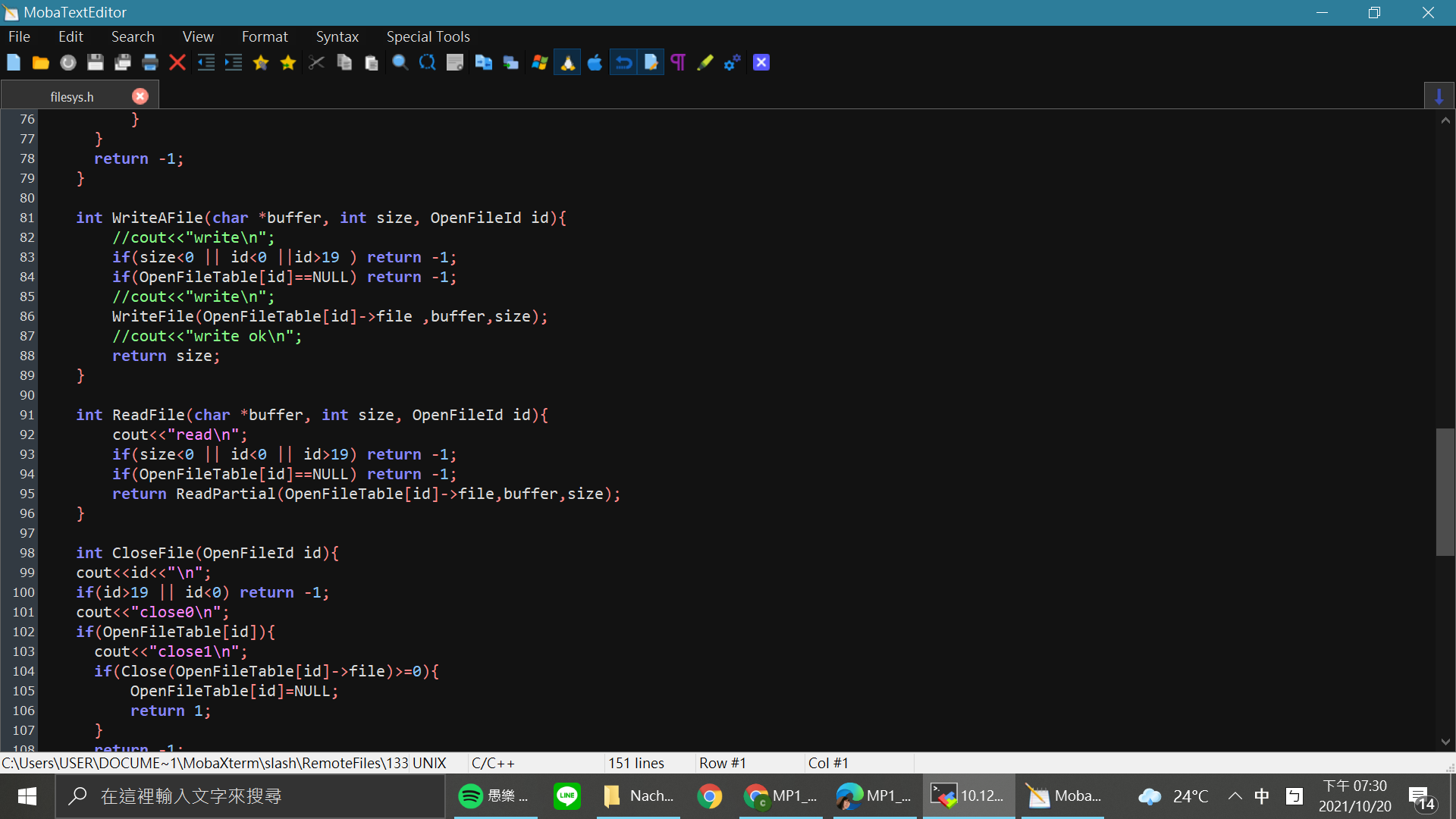
1. reg $4取出 val檔案名稱的指標，並從memory找到address存入buffer
2. 從reg $5取出size，存入numCharsize
3. 從reg $6取出fileID，存入fileID
4. 呼叫Syswrite並將結果存在status(1:success、0:failed)，把status存入reg $2
5. 最後set Program counter

**ksycall.h**



1. 在這裡加上SysWrite等指令，並在裡面呼叫WriteAfile function

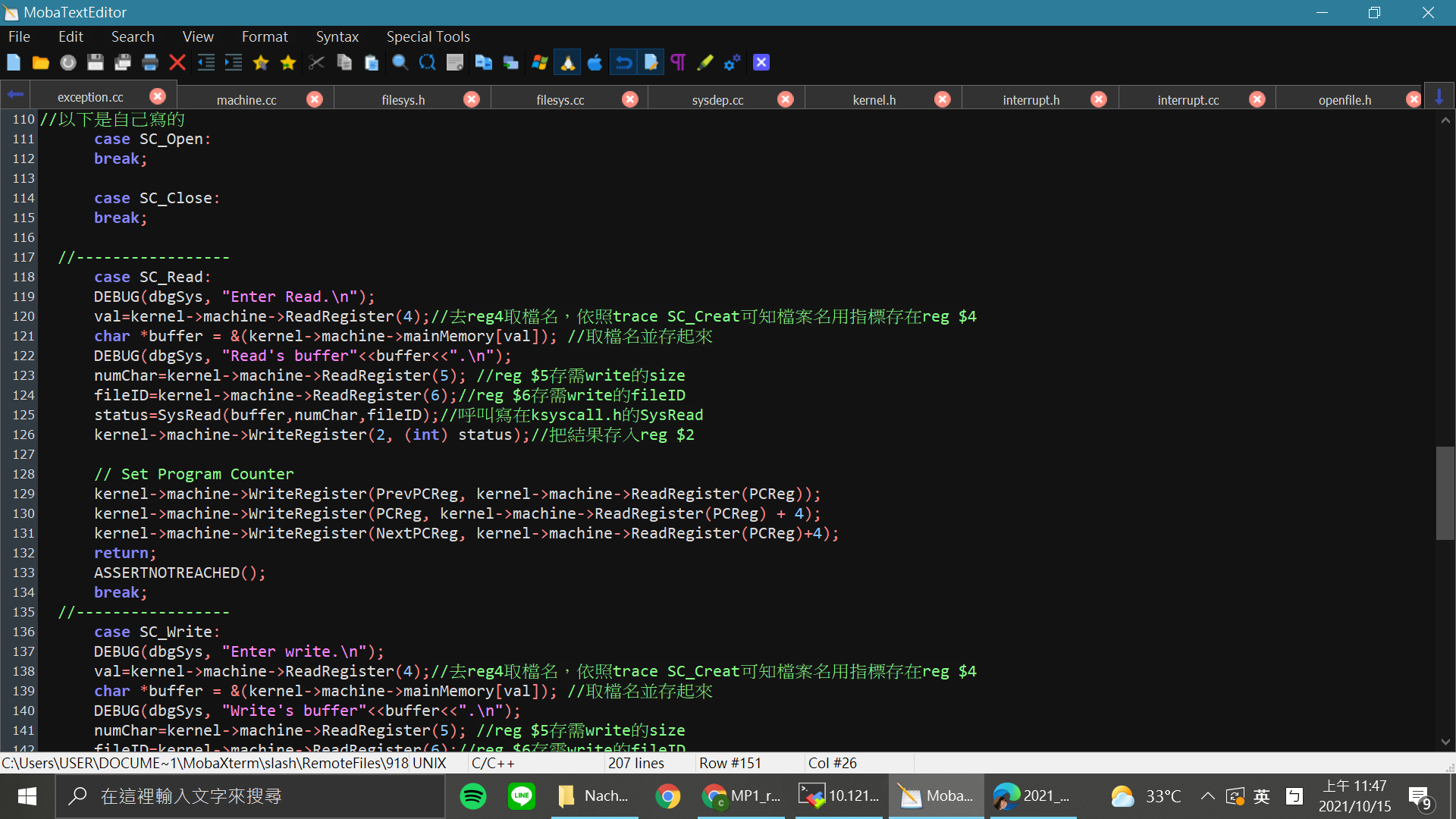
**filesys.h**



1. 當size<0 或id不在範圍內(0~19)時就return -1
2. 當OpenFileTable[[id]為NULL，也就是檔案不處於開啟狀態或是不存在，也return -1
3. 下一行呼叫WriteFile，傳入參數，這裡的參數OpenFileTable[id]->file是代表fileDescriptor(fid)，在open時new OpenFile時 fileDescriptor 存到int file ，所以要用 到fileDescriptor就必須要透過OpenFileTable[id]->file 的方式取得(需事先在openfile.h檔案裡面把file設成 public)
4. 最後依題目要求return size

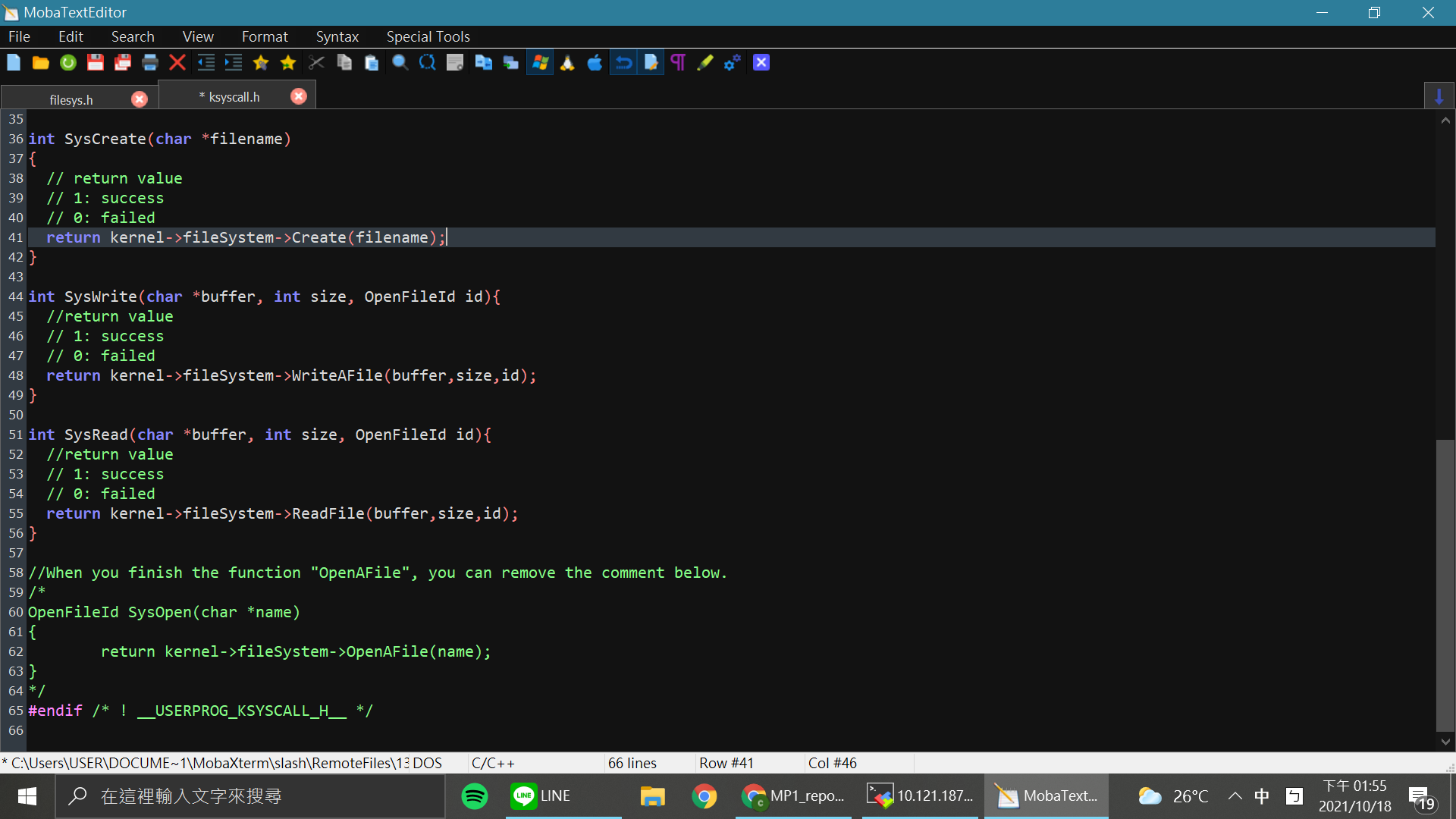
**(d)SC\_Read**

**exception.cc**



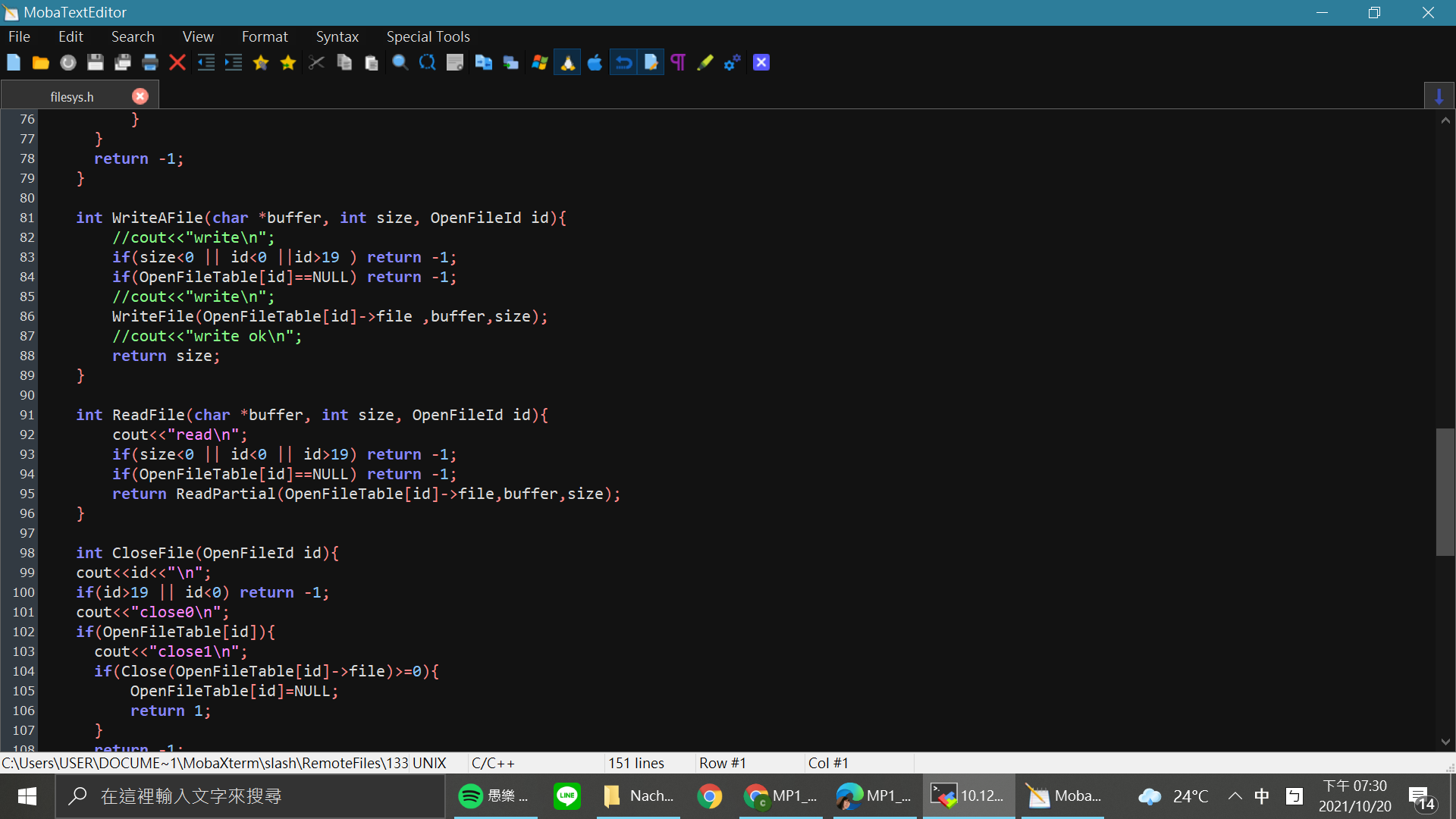
1. reg $4取出 val檔案名稱的指標，並從memory找到address存入buffer
2. 從reg $5取出size，存入numCharsize
3. 從reg $6取出fileID，存入fileID
4. 呼叫SysRead並將結果存在status(1:success、0:failed)，把status存入reg $2
5. 最後set Program counter

**ksyscall.h**



1. 在這裡加上SysRead等指令，並在裡面呼叫ReadFile function

**filesys.h**



1. 當size<0 或id不在範圍內(0~19)時就return -1
2. 當OpenFileTable[[id]為NULL，也就是檔案不處於開啟狀態或是不存在，也return -1
3. 下一行呼叫ReadPartial，傳入參數
4. ReadPartial會呼叫C++內建函數read，從對應 fid 檔案裡讀取長度size的字串，並寫入buffer裡面，最後回傳實際讀取的資料長度

**Part3\_What difficulties did you encounter when implementing this assignment?**

蔡懿晨：

這次作業我先做trace(a)(b)，一開始找到一些呼叫的關係就以為自己已經完成了，後來發現其實跟implement息息相關，在trace時應該要詳細到在每一行都仔細研究關係。在一開始trace時每個檔案都要找才知道function是在哪邊，後來找到了Linux的grep指令，用起來真的很方便！在實作時，遇到OpenFileTable的問題，一開始看不懂是什麼，上網查加上去trace其他檔案才了解。也有一些小細節要注意，比如說檔案限制的數量等等

江咏宸:

這次的困難點就是第一次接觸這種類型的code，因此trace code花了不少時間來了解系統架構。並花了許多時間trace SC\_PrintInt，因為它的function call真的很多。再來implement就比較有概念了，第一次嘗試寫的時候有點搞錯openfieldid和openfiletable的概念了，幸好隊友很罩幫我修code才讓這次作業提早結束。