**MP2 Report**

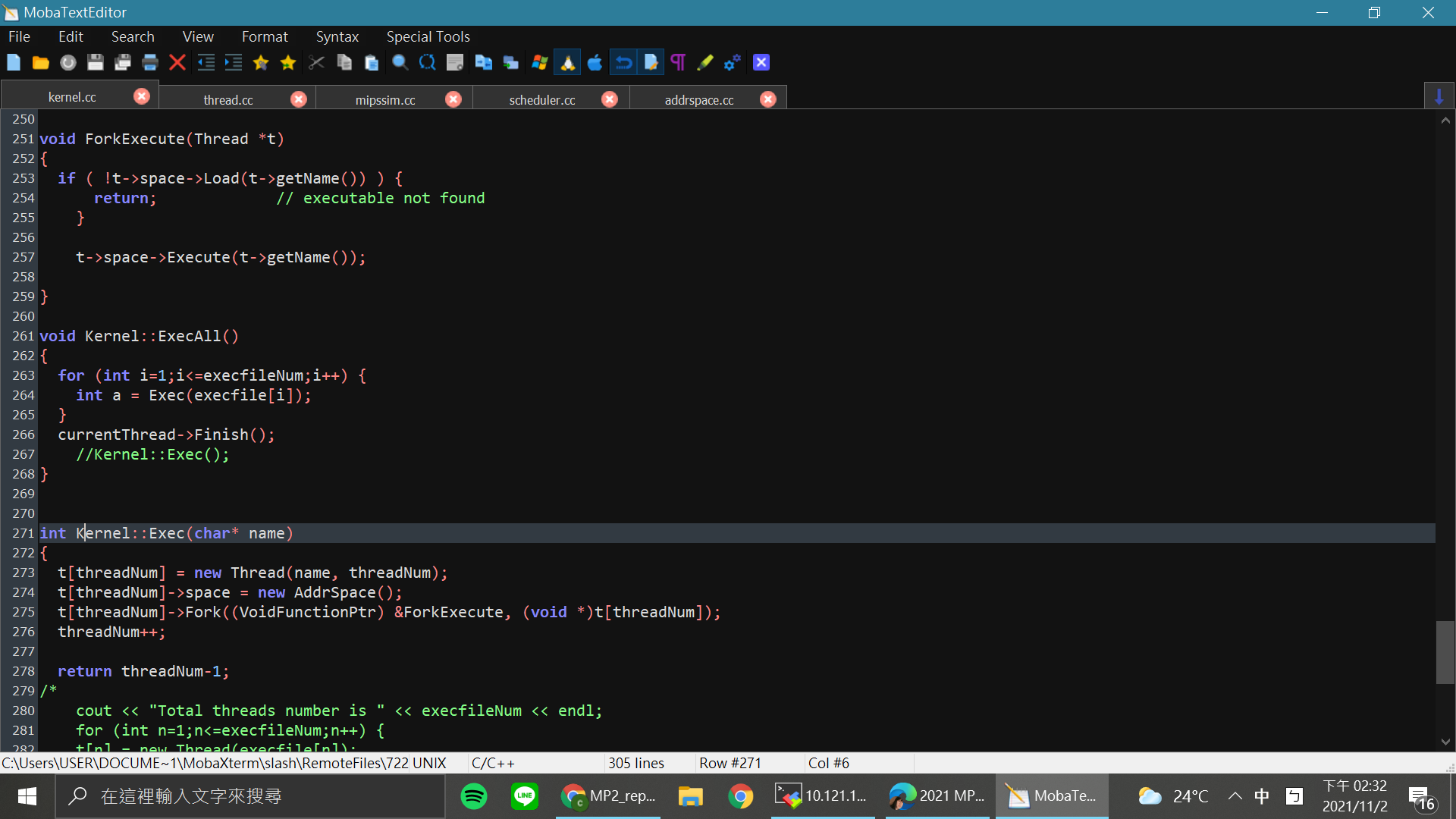
組別:os21team34

成員:江咏宸108030001、蔡懿晨107070035

| Contribution | |
| --- | --- |
| Tracecode與相關report | 蔡懿晨 |
| Implement與相關report | 江咏宸 |

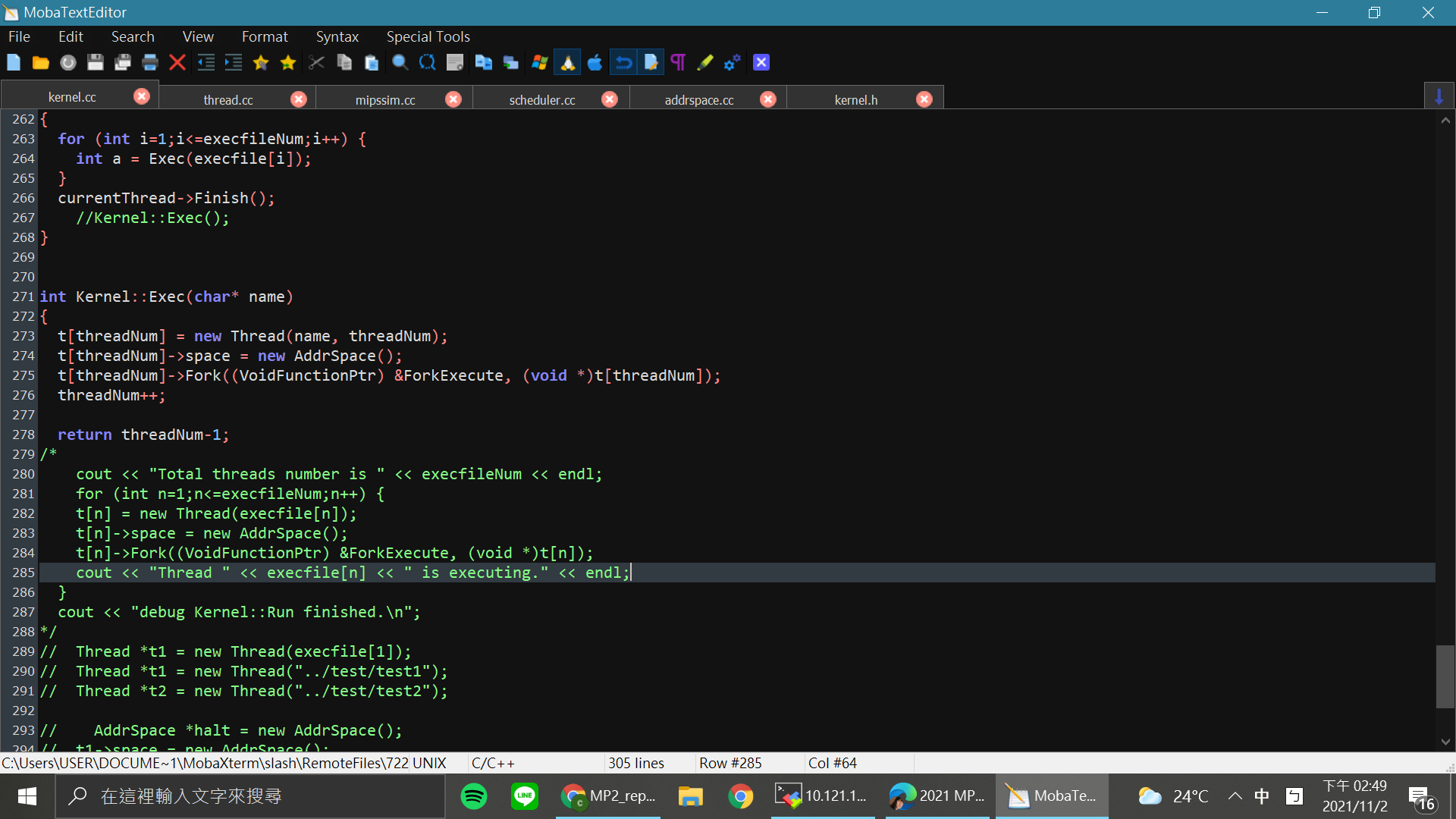
**Part\_1\_Trace Code**

Kernel::ExecALL()

****

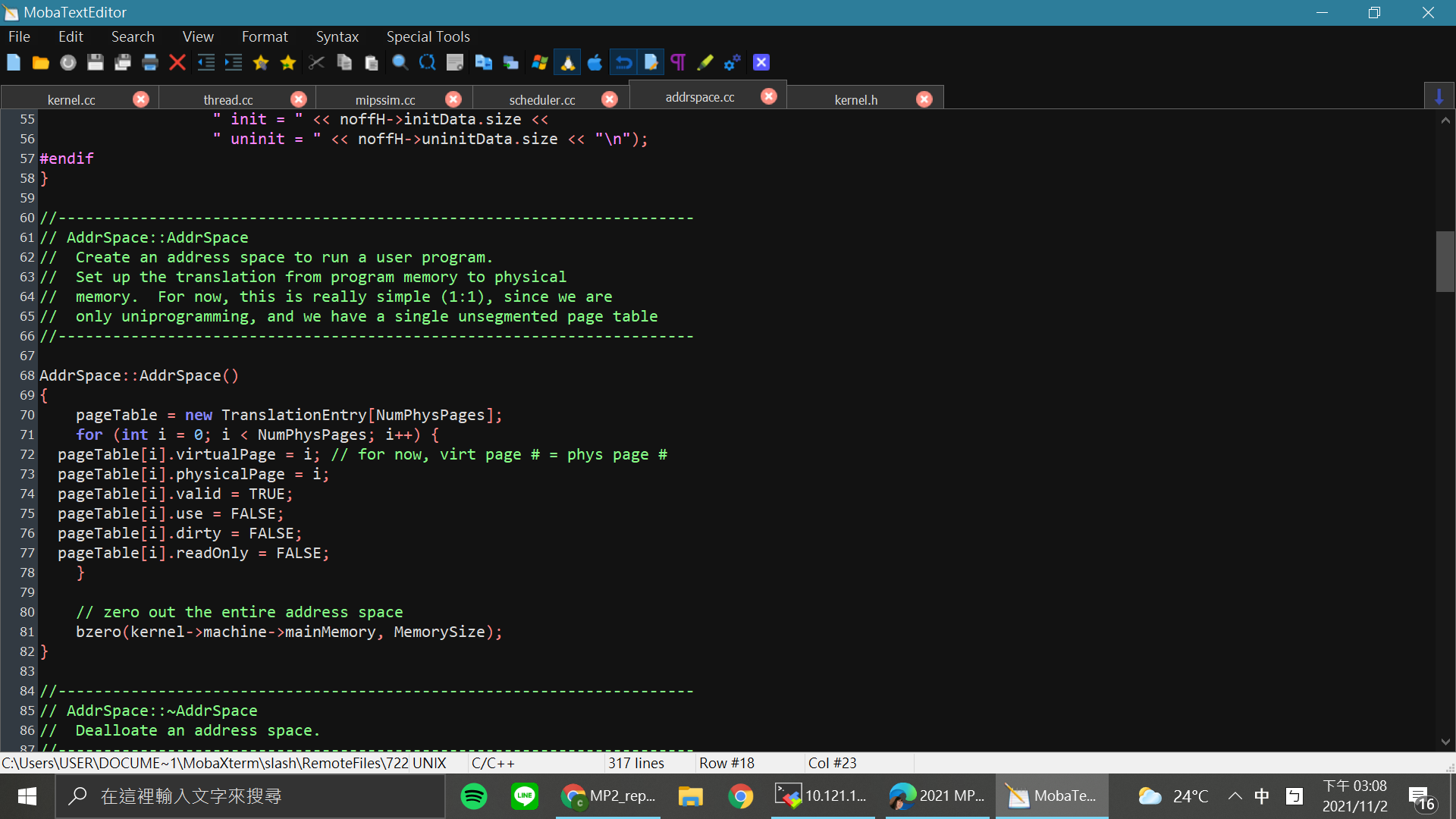
* 目的：分配threads給不同file
* 用迴圈去讓現有正在執行的file分配到thread，呼叫Exec達成此目的，傳遞filename(execfile[i])
* 當此thread執行完畢，CPU暫停使用時，呼叫Finish

**Kernel::Exec()**

****

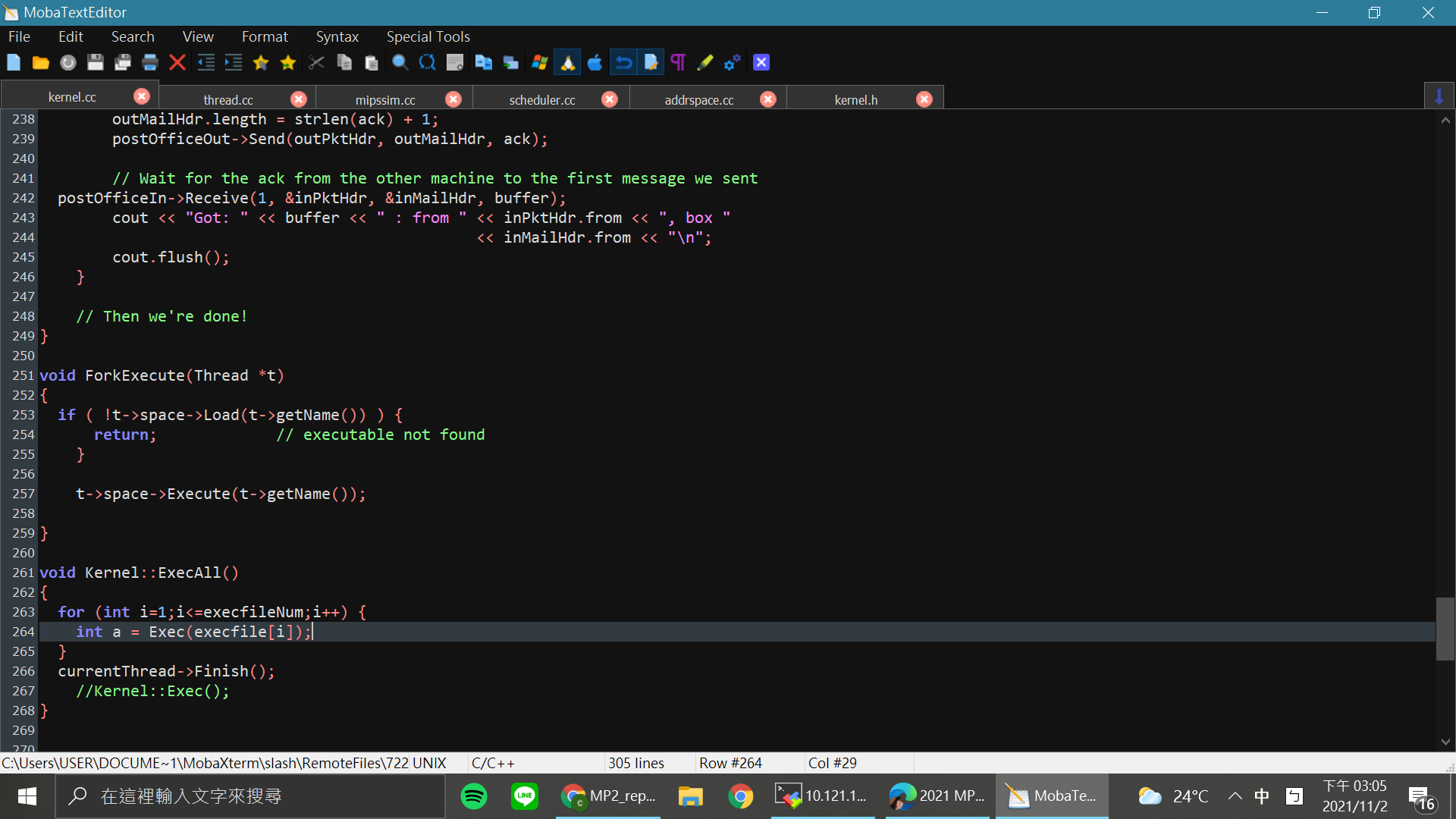
* 目的：創建一個新的thread去處理file
* 創建一個新的Thread並傳入參數(檔名,目前的threadNum)，初始化一個thread的control block
* 呼叫addrspace來分配記憶體空間給space和space裡的page table，然後對page table做初始化，包括page number、frame number、valid...等等。space負責維護page table相關的data
* 呼叫Fork，傳入參數 &ForkExecute是當thread開始執行時的address，也就是現在正在執行的程式，t[threadNum]是須被傳送的參數

**AddrSpace::AddrSpace()**

****

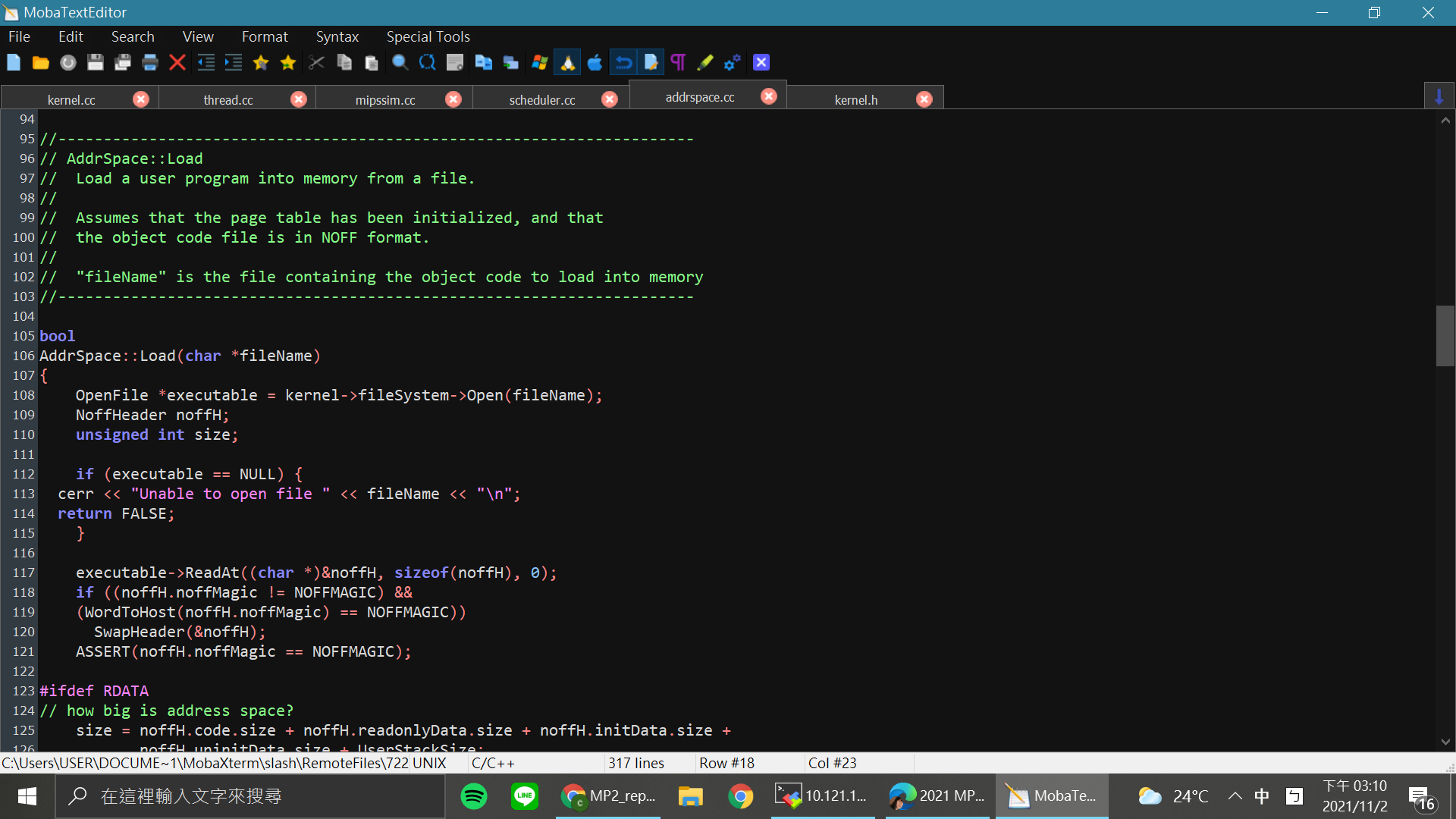
* 目的：建立一個address space來跑user program，創建一個新的page table entry並初始化所有page
* 在這裡page table size設為跟physical pages一樣大，而且在for loop裡面，把virtualPage跟physicalPage都設為一樣的，等同於一個thread擁有所有的記憶體，無法達成multiple thread的概念

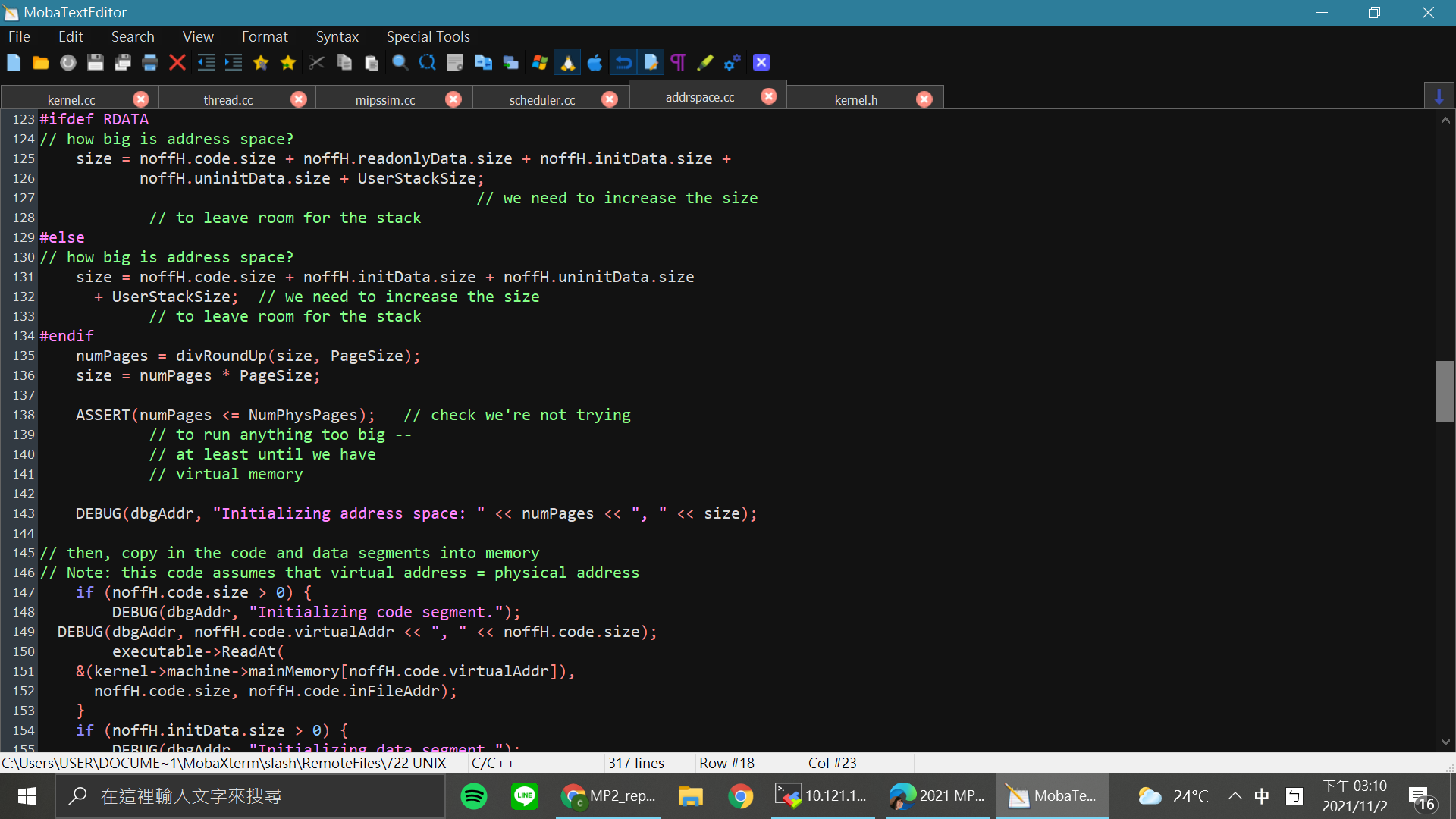
**void ForkExecte()**

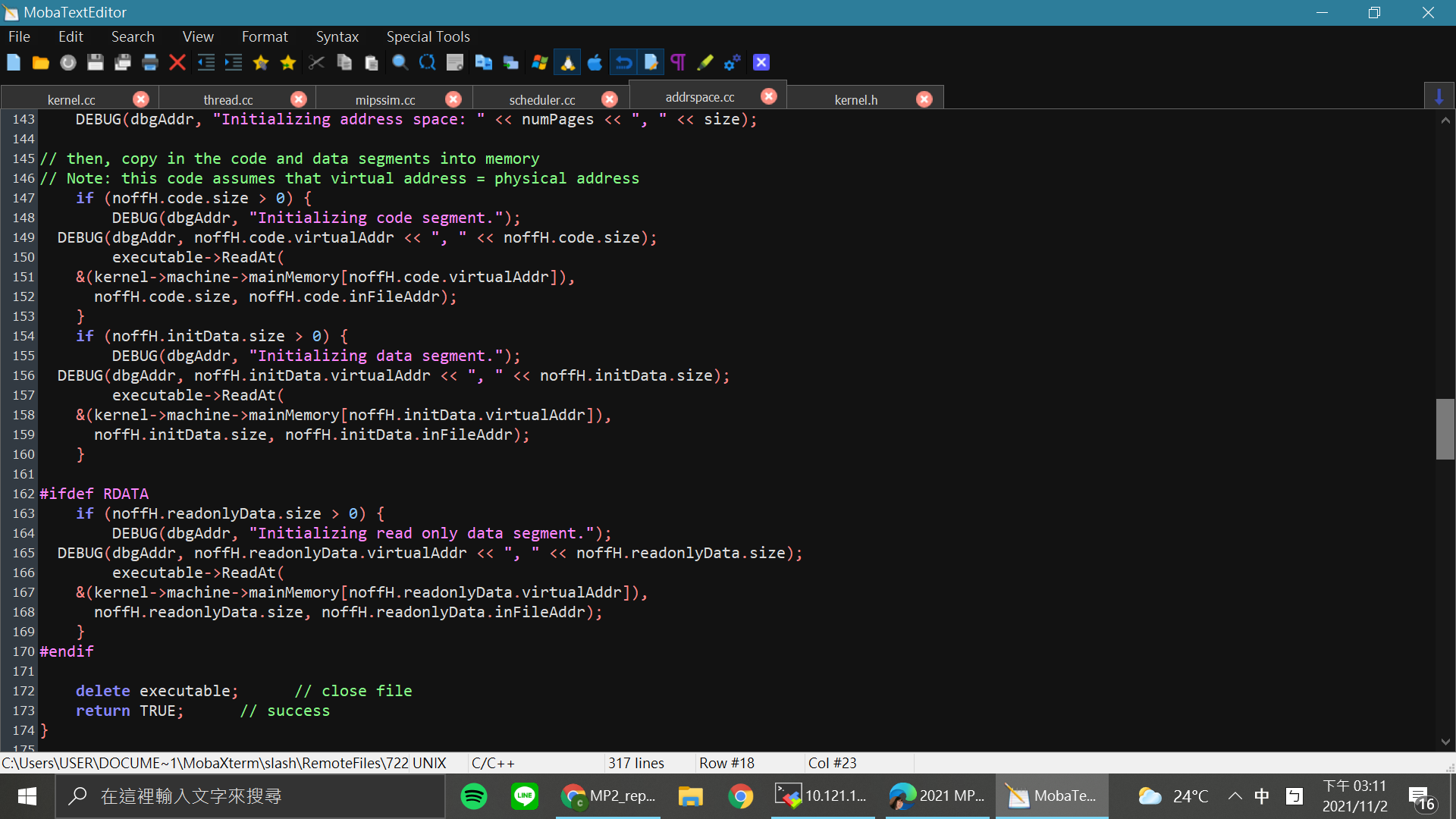
****

* 目的：呼叫load來將user program放入memory，呼叫execute執行thread

**AddrSpace::Load**

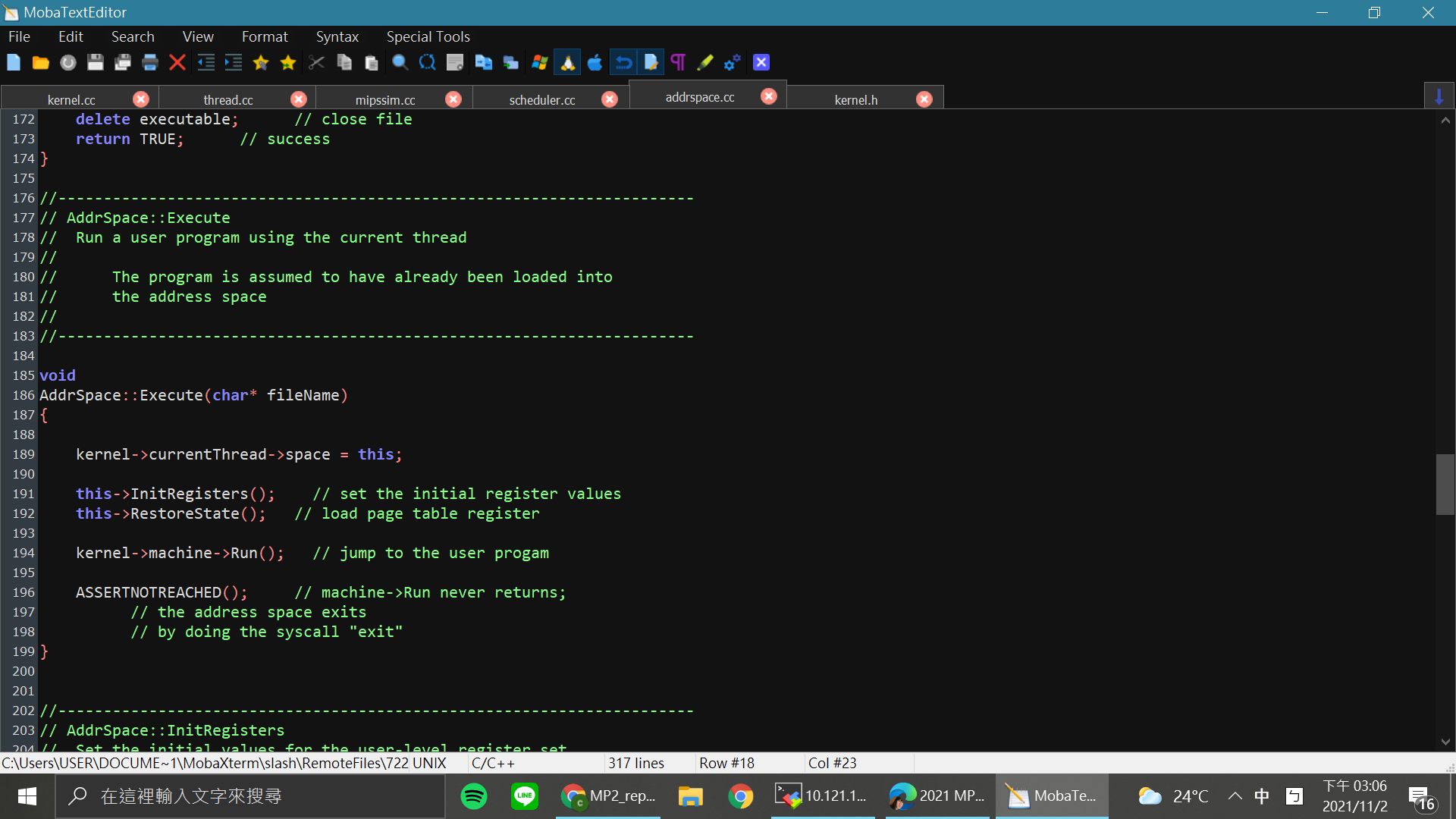
****

****

****

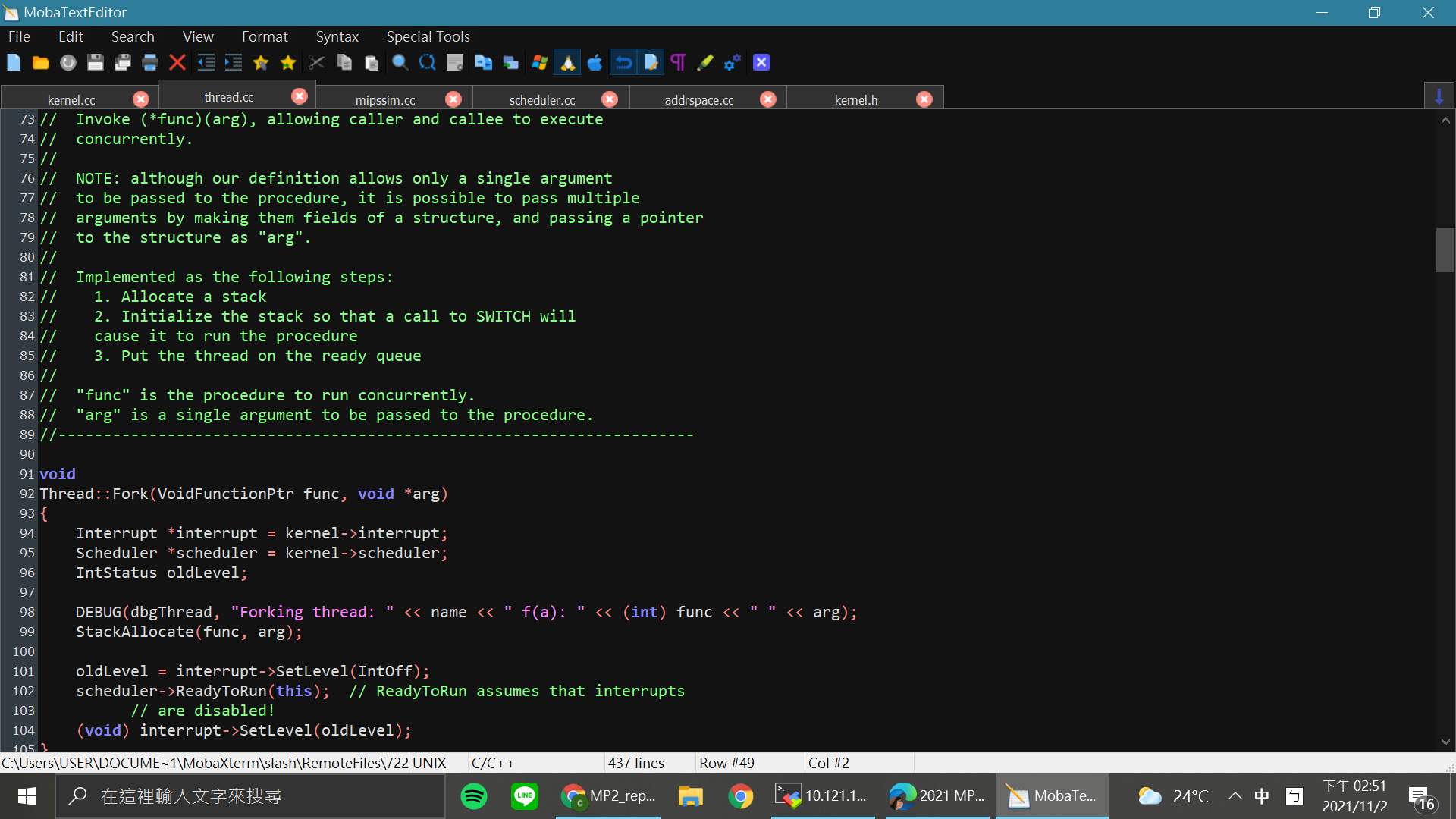
* 目的：把user program 從file load至memory
* 一開始把檔案打開並確認是否能打開
* 所有的 Nachos user program 會轉成 Nachos object file (NoffHeader 型態，變數noffH)
* 設定size、確認size
* ReadAt這個function是在allocate一塊segment大小的記憶體空間並將file匯入

**AddrSpace::Execute()**

****

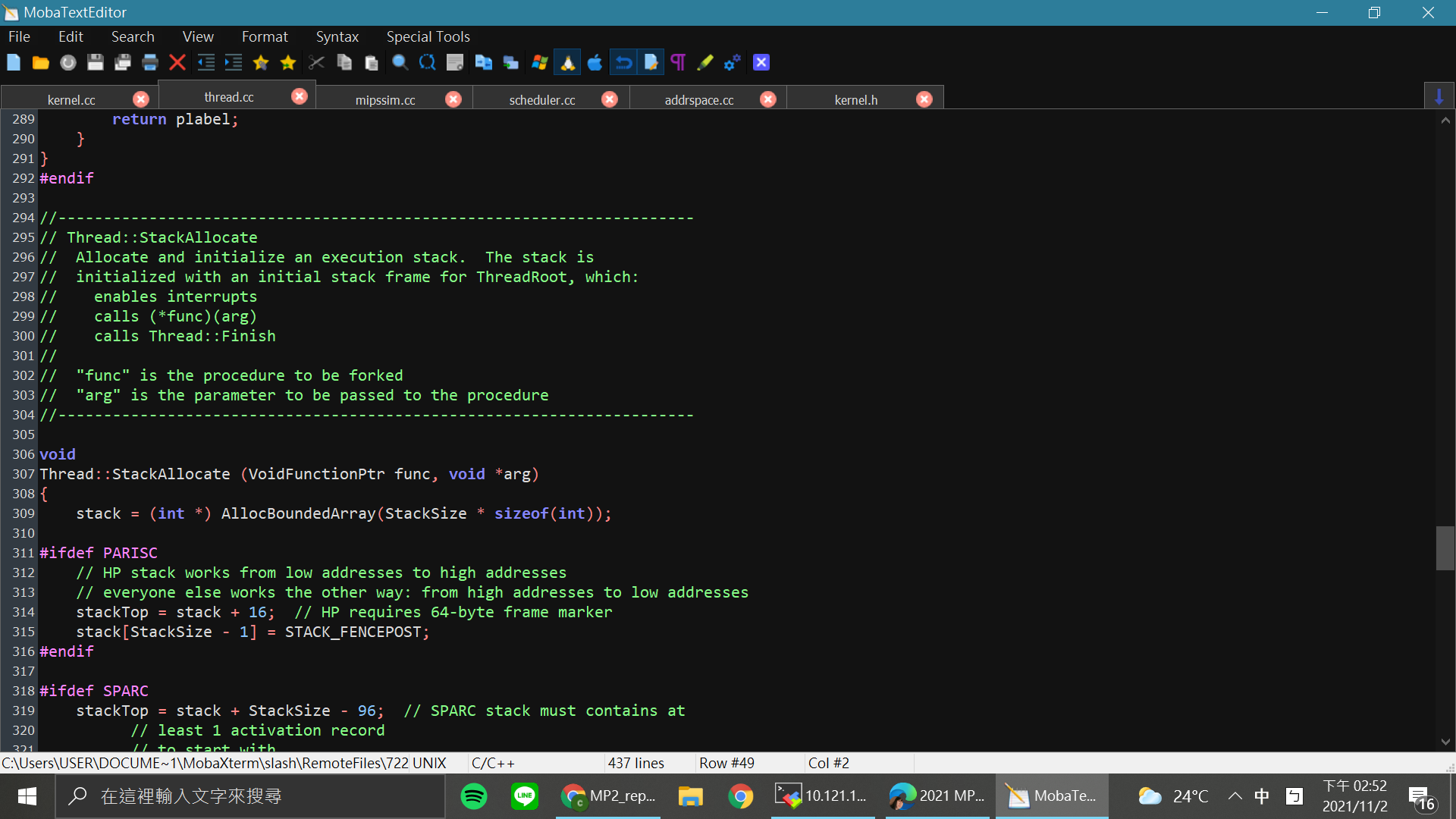
* 目的：調整Register、PageTable，執行程式
* 呼叫InitRegisters()，設定user level的register的initial values
* 呼叫RestoreState()，在context switch時，restore the machine state改成現在的thread的pageTable
* 呼叫run去執行程式，run裡面有oneInstruction

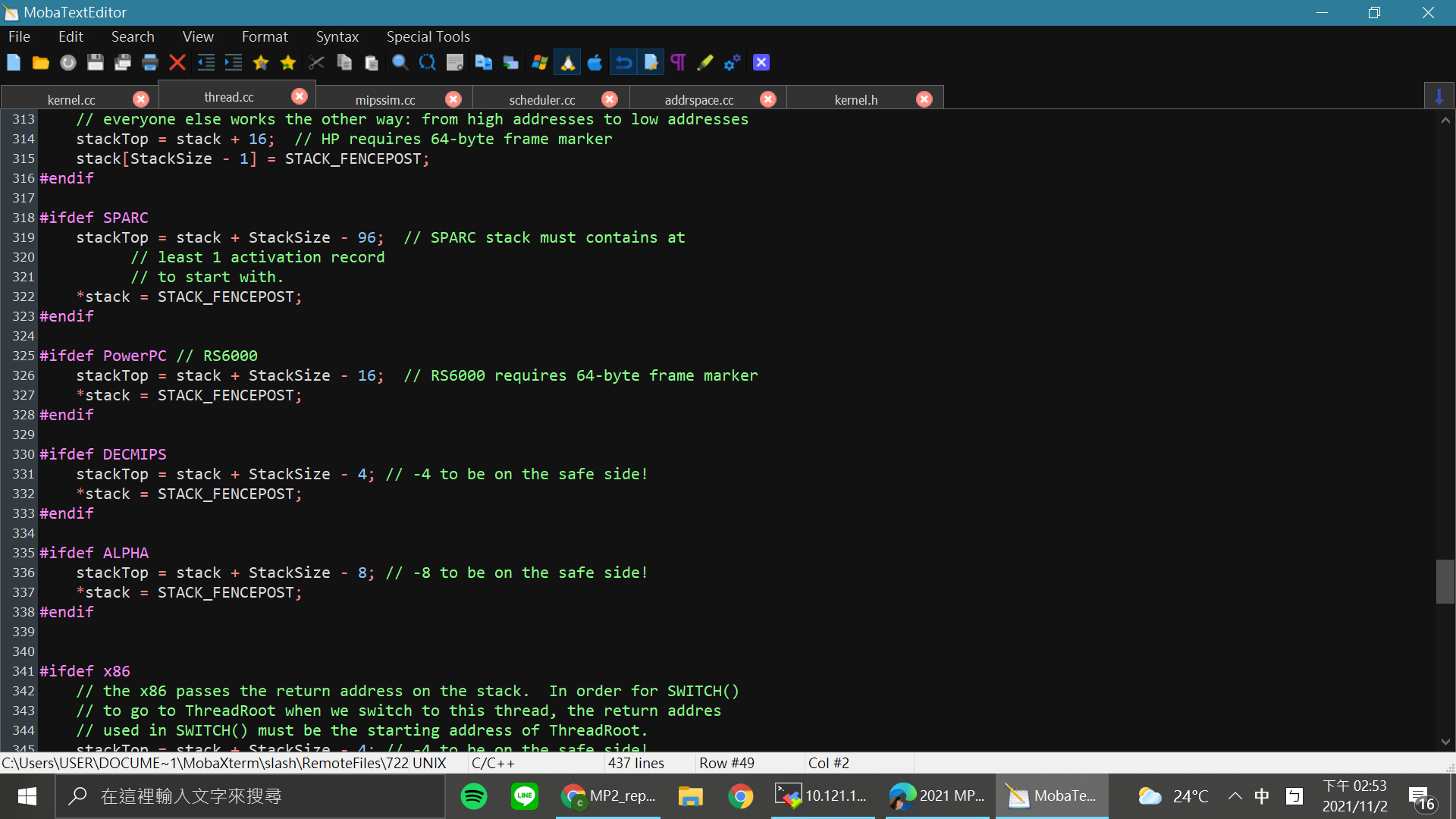
**Thread::Fork()**

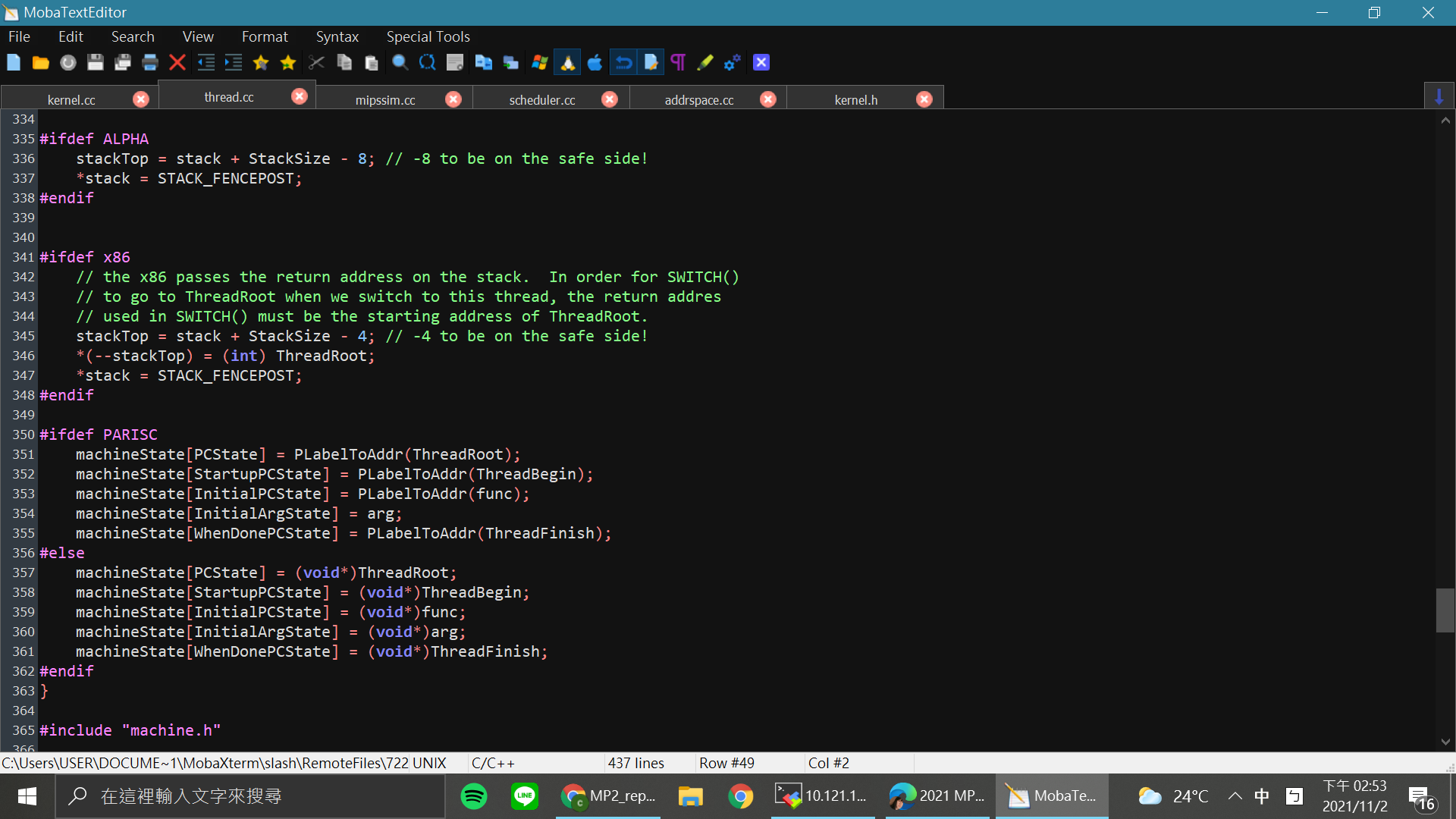
****

* 目的：Allocates stack space for the new thread, initializes the registers, put thread into the ready queue(new->ready).
* 呼叫stackAllocate，分配stack給thread，並通過設定 machineState[] 做execution stack的初始化
* 將interrupt disable
* 呼叫ReadyToRun，將一支經過上述初始化的thread放進ready（尚未running）
* 恢復interrupt

**Thread::StackAllocate()**

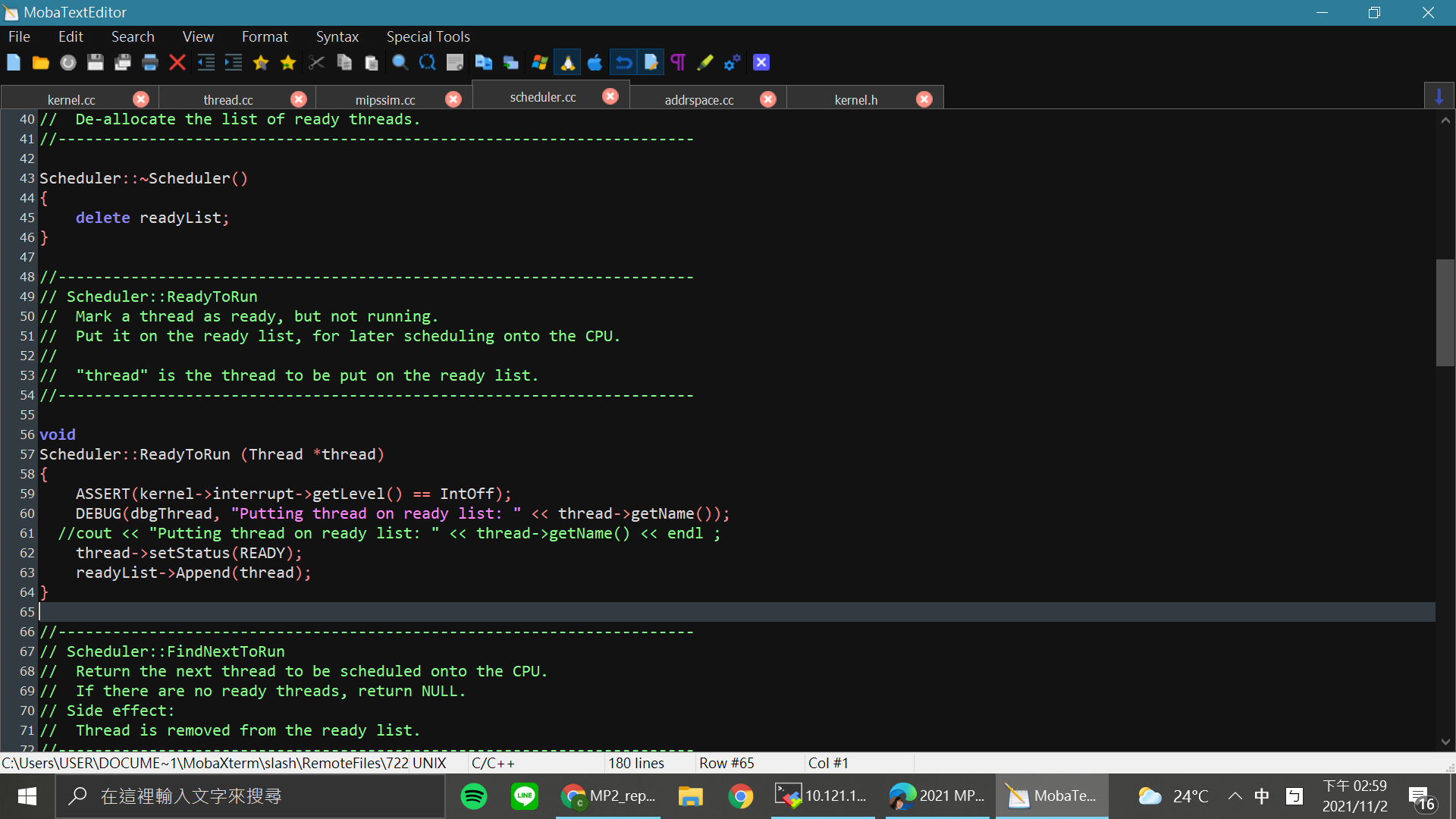
****

****

****

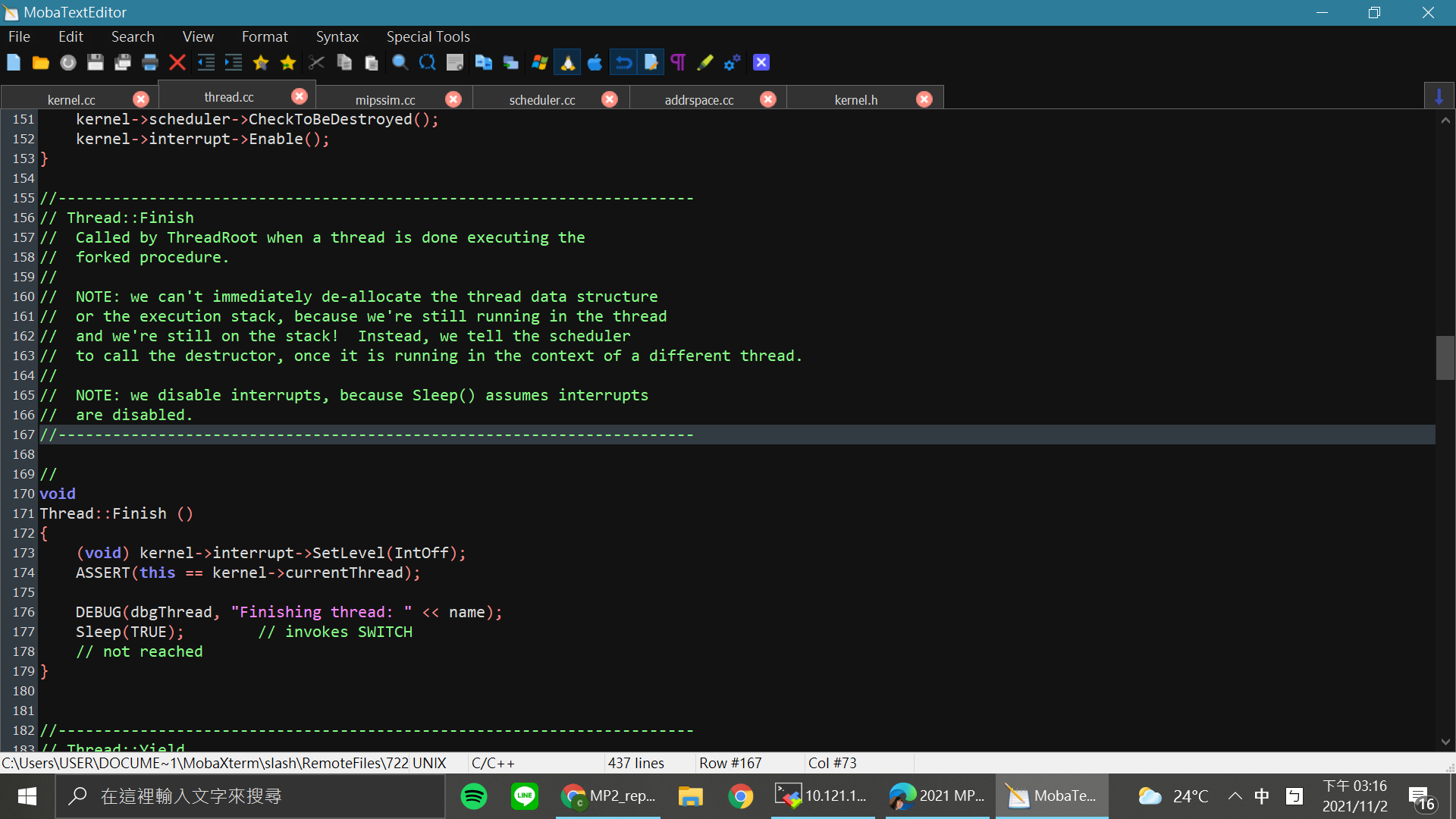
* 目的：根據不同環境來分配、初始化stack
* 檢查thread的size
* 透過設定 machineState[] 做execution stack的初始化

**Scheduler::ReadyToRun()**

****

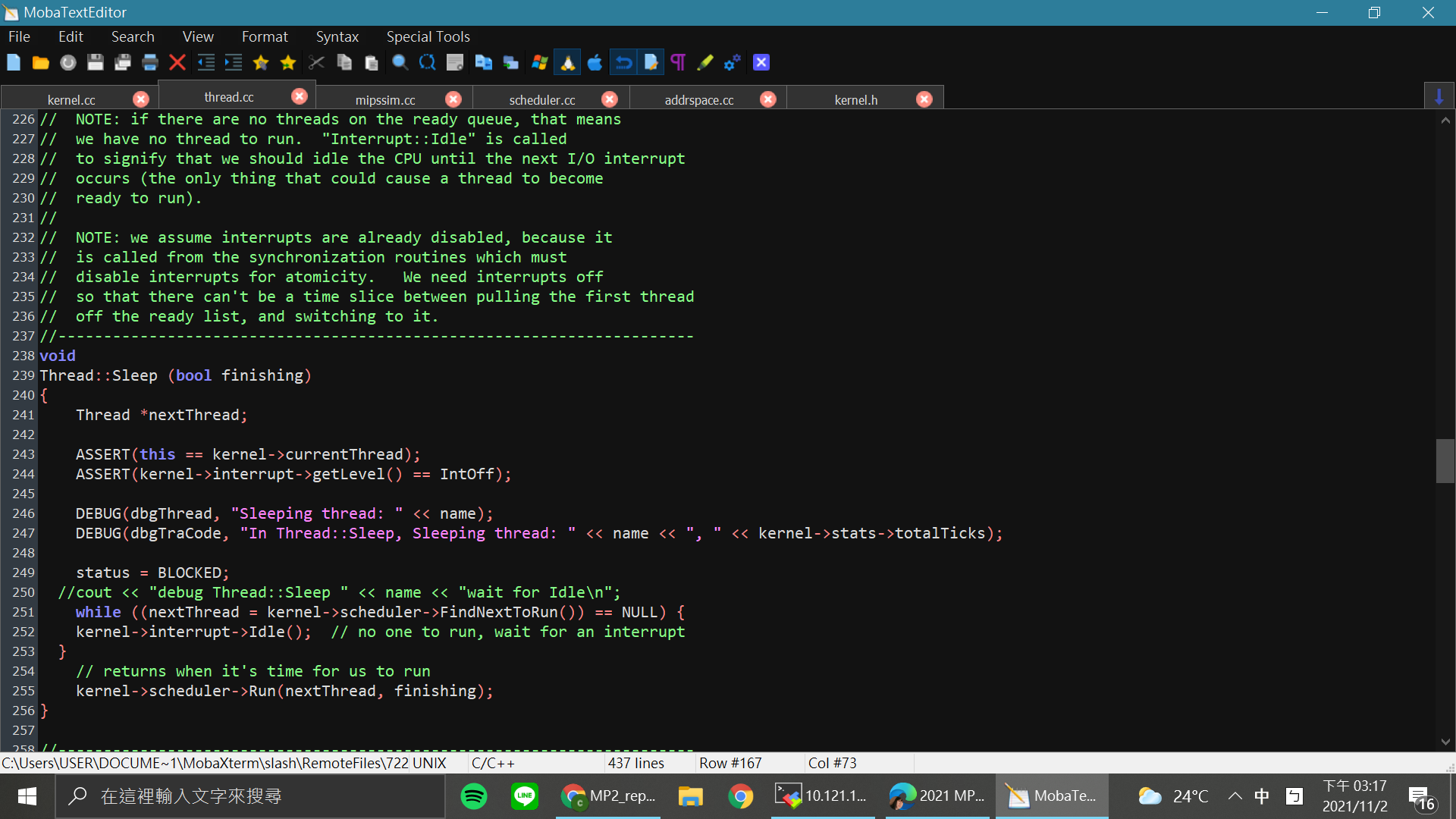
* 目的：已經被放在queue的thread，已經ready但尚未running，正等待被挑選進CPU
* 將Status設為Ready，放入ready queue

**Thread::Finish()**

****

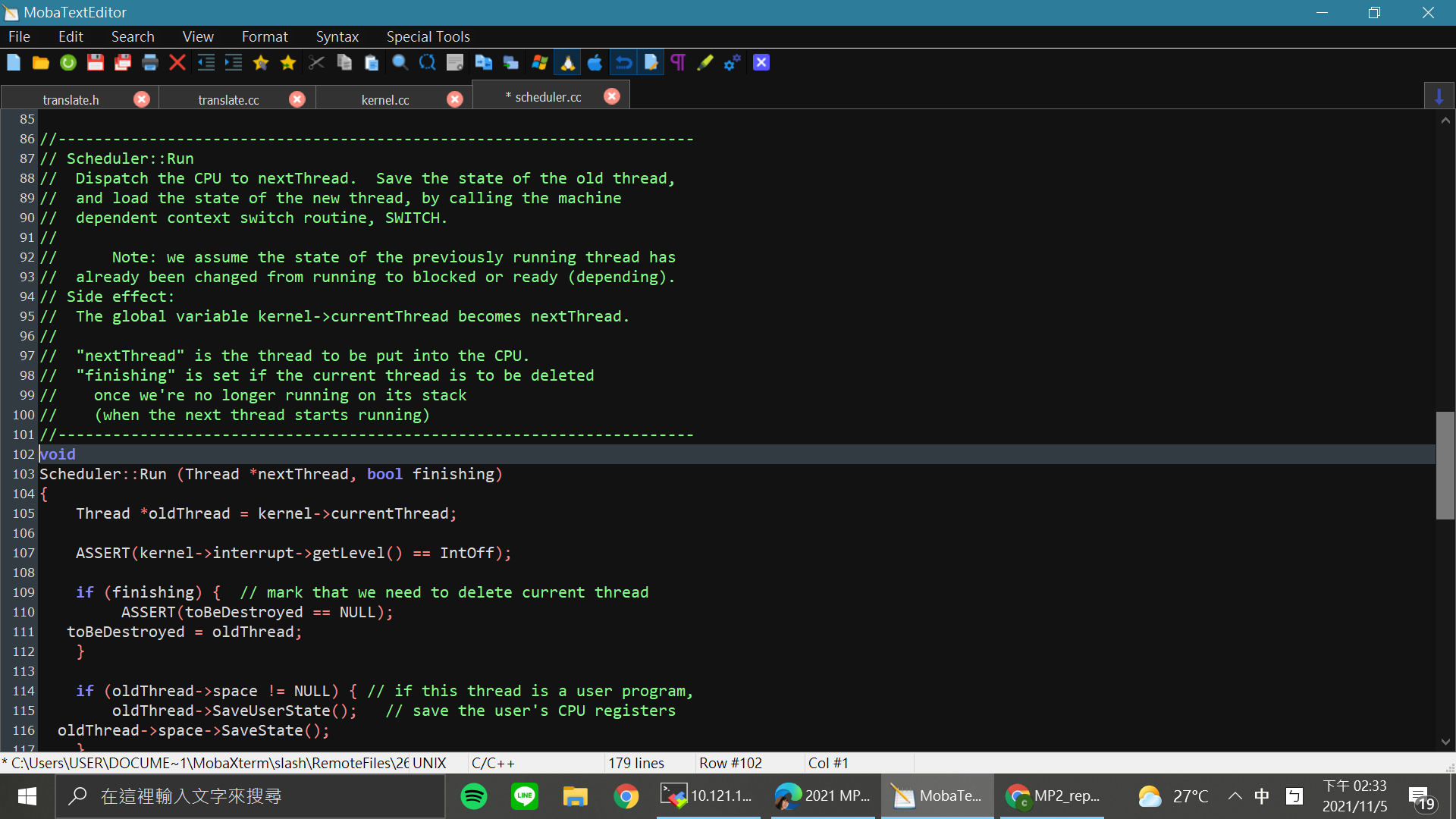
* 目的：當program被執行完成，被呼叫刪掉目前thread
* disable inteerupt並呼叫sleep

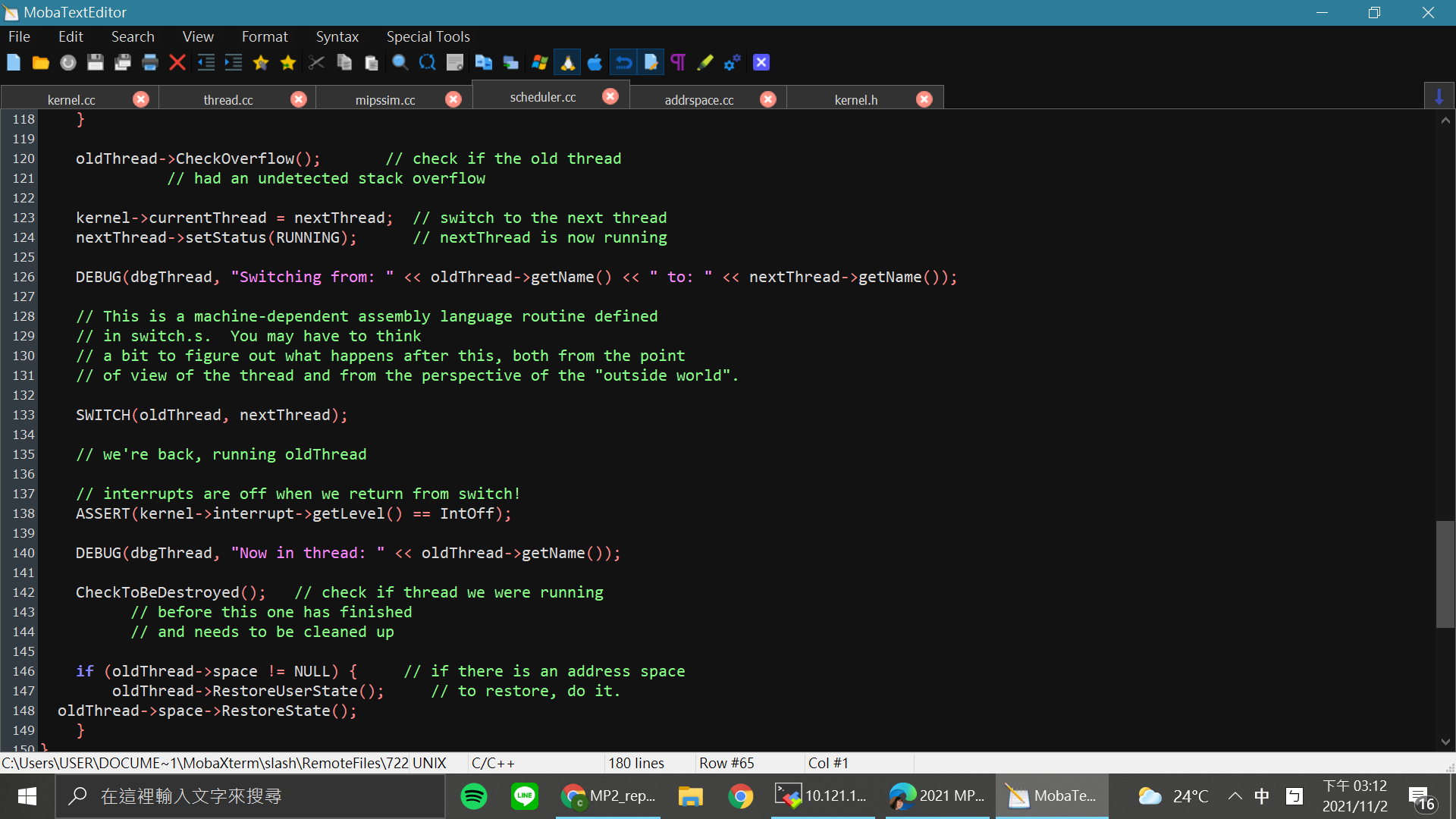
**Thread::Sleep()**

****

* 目的：block thread
* 將status設為Blocked
* 用while來模擬只要沒有新的thread要run，就持續idle
* 如果有thread要跑就跳出while，執行下一個thread，開始回到run做content switch

**Scheduler::Run()**



****

* 目的：實作content switch
* 如果old結束就刪掉，而如果old還存在就保存舊的thread的狀態
* 將kernel switch到新的thread，load新的thread的狀態
* 呼叫Switch，讓oldThread和nextThread交換，在此階段CPU是idle的
* CheckToBeDestroyed()，檢查thread的Program是否已經做完了

**Question**

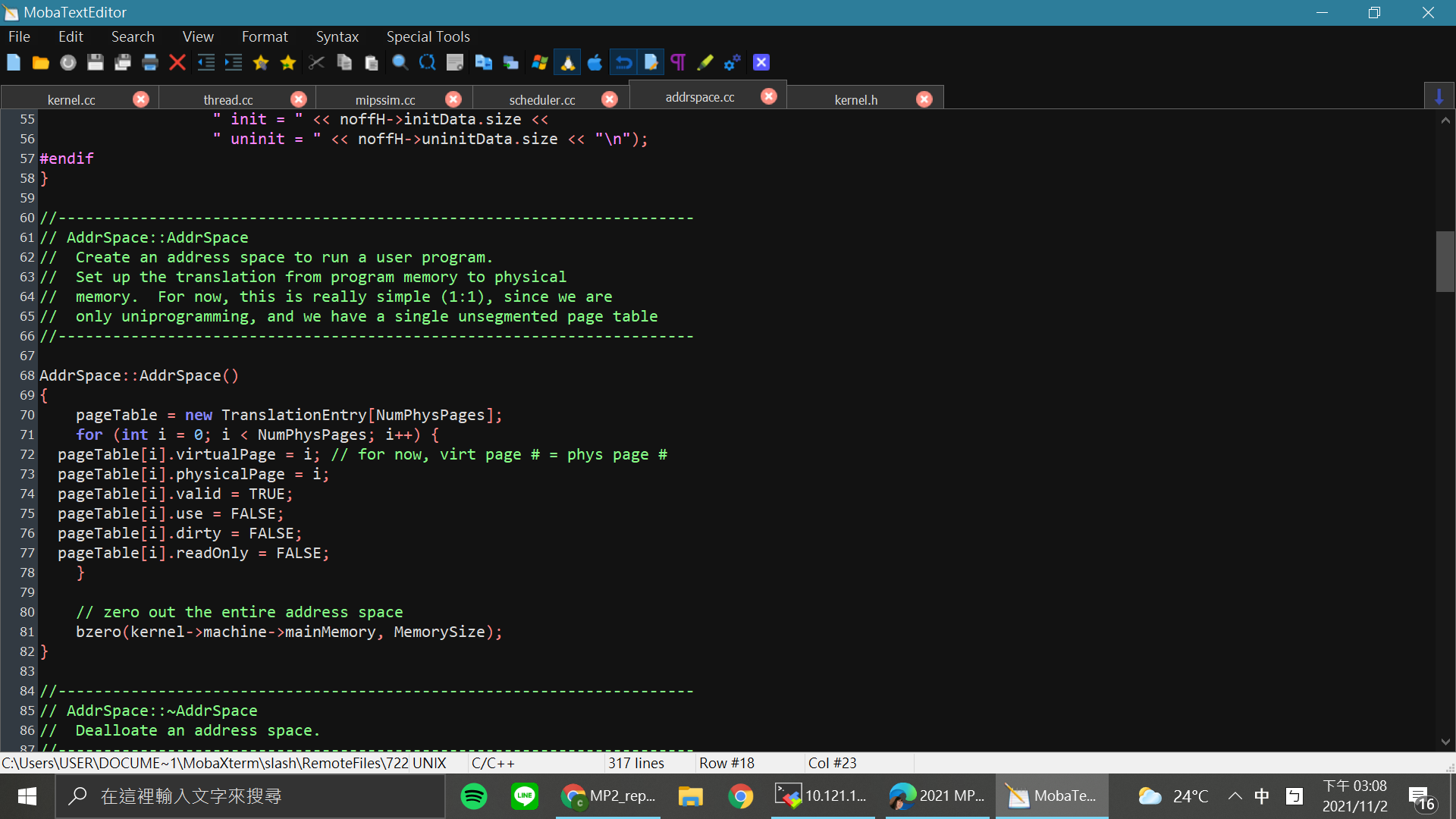
1. **How Nachos allocates the memory space for new thread(process)?**

**kernel::Exec()**

t[threadNum] = new Thread(name, threadNum);

t[threadNum]->space = new AddrSpace();

Thread()會初始化一個新的thread（但不會把它放入ready queue）



原本在AddrSpace裡面去做PageTable，將VirtualPage和PhysicalPage設為一樣位置，經過實作之後我們改為在Load的時候做PageTable，bzero也改在Load時清空記憶體，根據分配之空間清除記憶體，而不是所有記憶體

1. **How Nachos initializes the memory content of a thread(process), including loading the user binary code in the memory?**

初始化的部分原本在addrspace做，實作後轉為Load。Exec()也會呼叫ForkExecute()，ForkExecute()裡面有t->space->Load(t->getName())會把code load至memory在Load裡面。

1. **How Nachos creates and manages the page table?**

原本是在AddrSpace()裡面去做創立和管理，但這樣當兩個programm同時執行時可能會共有到相同的physical page，所以在實作時我們移至load去做管理

1. **How Nachos translates address?**

會在Load裡面去做translation，因為原本的NachOS不支援multi-programming，是把virtual page==physical page，計算address是pagesize\*page\_number + offset

但現在我們實作multi-programming，計算方式就改為pageTable[noffH.readonlyData.virtualAddr/PageSize].physicalPage \* PageSize + noffH.readonlyData.virtualAddr%PageSize]

（virtualAddr /PageSize，求出在virtual的第幾個page，再用pageTable去找對應的 physicalPage，找到後乘上PageSize算出physicalPage在memory裡的位置再加上 offset(virtualAddr%PageSize)，即找出對應的memory位置。）

1. **How Nachos initializes the machine status (registers, etc) before running a thread(process) ?**

Address::Execute()

呼叫InitRegisters()，設定user level的register的initial values

呼叫RestoreState()，在context switch時，restore the machine state改成現在的thread的pageTable，把現在正在執行的program的pageTable和pageTableSize (numPages)load 進 memory裡面

1. **Which object in Nachos acts the role of process control block?**

**class Thread**

是Thread object，從Run()可看到context switch是在做save、reload thread的過程，所以在NachOS就是把thread 當成 PCB

1. **When and how does a thread get added into the ReadyToRun queue of Nachos CPU scheduler?**

Thread::Fork()呼叫ReadyToRun，在ReadyToRun裡面的

thread->setStatus(READY);

readyList->Append(thread);

這兩行把thread狀態設為在ready，並放入ReadyToRun queue

**Part\_2\_Implementation**

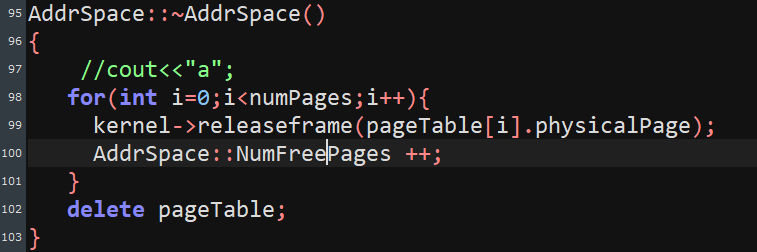
AddrSpace::AddrSpace()

把當初的code全部刪掉，讓pageTable的建立與bzero()丟到AddrSpace::Load()時在做。

static int NumFreePages

在addrspace.h新增，去紀錄還剩下多少frame可以被使用。初始值=NumPhysPages

AddrSpace::~AddrSpace()

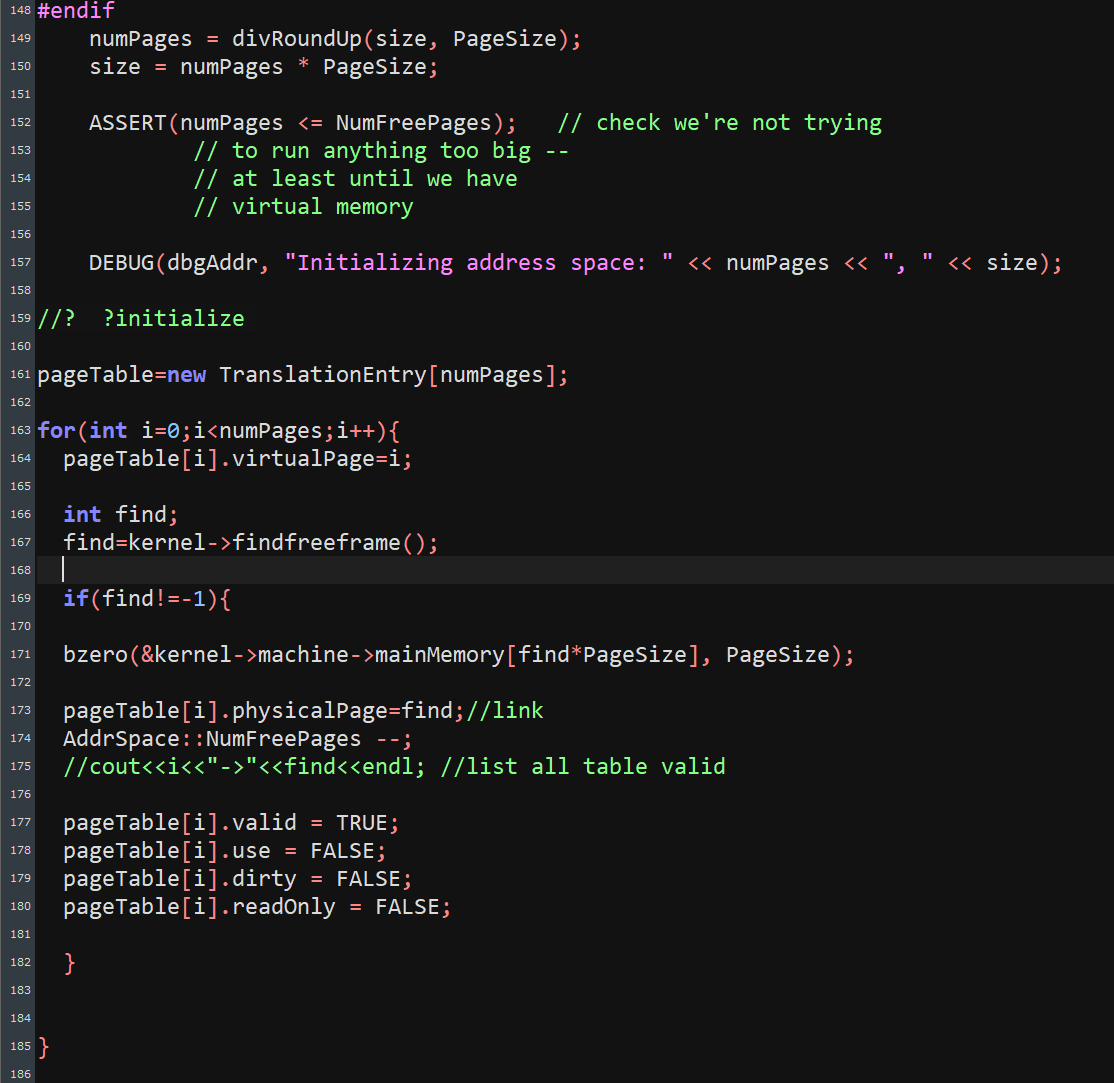
****

1.呼叫kernel的releaseframe()把當初使用到的physicalPage restore成初始沒被使用過的狀態。

2.NumFreePages 回復成初始狀態

3.delete pageTable，因為程式執行完畢了不再需要用到了。

bool AddrSpace::Load(char \*fileName)

****

149~150行:

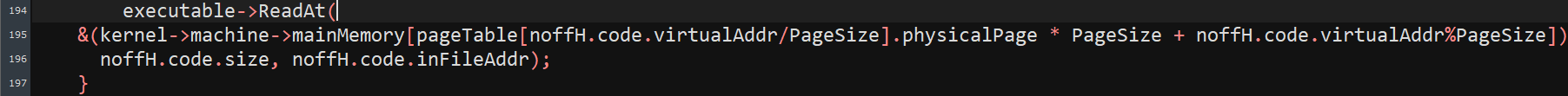
計算程式需要的numPages(size:code size/PageSize)，並取上限是因為確保程式都有被load進memory裡面，同時利用numPages更新size大小。

161行:

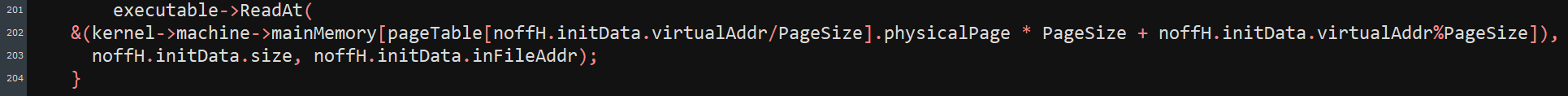
利用numPages的大小建出pageTable

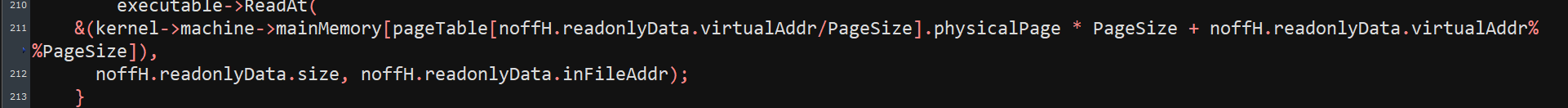
163~180行:

每個pageTable的virtualPage設為i，呼叫kernel->findfreeframe()找出空的frame並把physical page number設給pageTable[i].physicalPage。同時更新NumFreePages與設置 valid,use,dirty,readOnly的值。171行code是確保每次把code load到memory的特定位置時該frame的memory是乾淨的。



**ㄎㄟ**

****

****

上面改的部分是針對noffH的code,initData,readonlyData進行memory轉換的更改。

int ReadAt(char \*into, int numBytes, int position):將position的numBytes寫入into裡面。這裡的into就是把code,initData,readonlyData分別要寫入在mainMemory的地方。mainMemory實際上的位置需要pageTable做轉換:

mainMemory[pageTable[noffH.code.virtualAddr/PageSize].physicalPage \* PageSize + noffH.code.virtualAddr%PageSize]

code的virtualAddr/Pagesize得到pageTable的index，從pageTable取出來該index的physicalPage\*PageSize得到frame的base address。再加上virtualAddr%PageSize(offset)就可以得到mainMemory要放入code的physical address的起始位置。第二個參數是noffH.code.size，代表要寫入mainMemory的code大小。第三個參數noffH.code.inFileAddr代表要寫入的檔案位置。

剩下的noffH.initData和noffH.readonlyData也是同樣的轉換方法。

kernel.h/kernel.cc

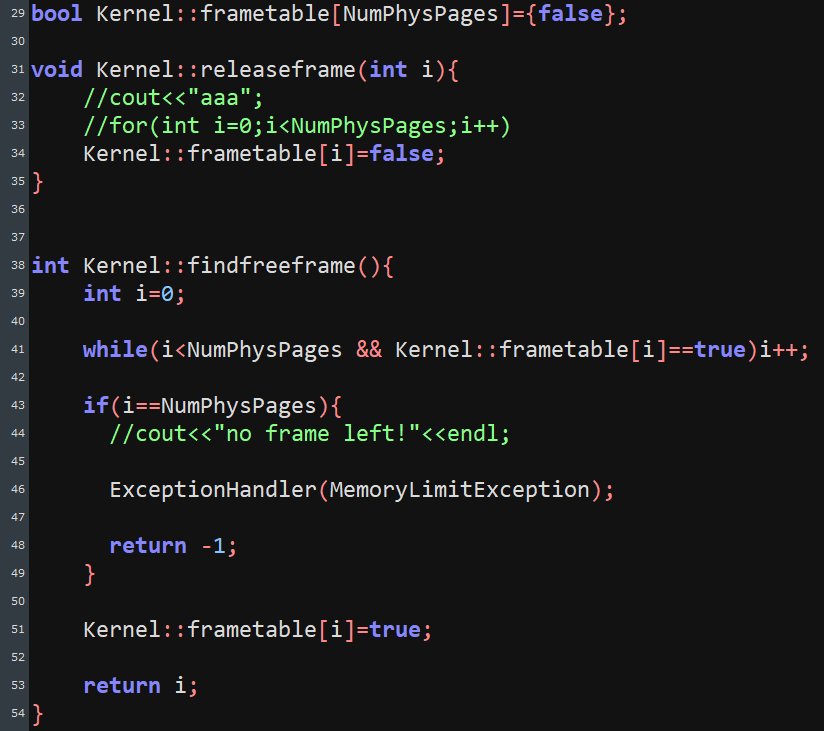
新增:

int findfreeframe()

void releaseframe()

static bool frametable[NumPhysPages]

利用以上function和array讓kernel維護frame的使用情形。



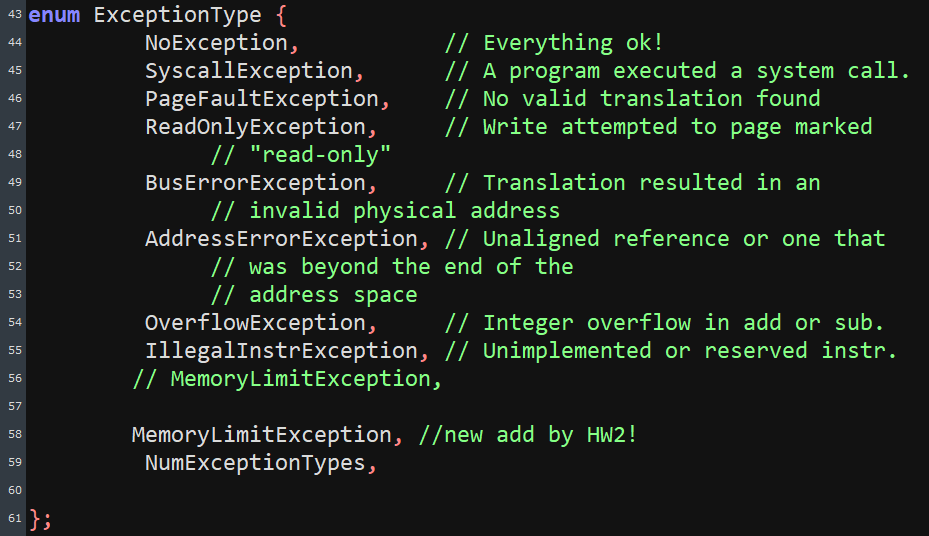
1.frametable初始設為false代表都沒有被使用到

2.releaseframe(int i):把frametable[i]設為false，讓該frame可以繼續使用。

3.findfreeframe():讓 i 去找沒被使用的frame，如果成功找到就把frametable[i]設為true代表該frame被使用了，最後return i 代表第幾個frame被assign給pageTable。

如果沒有找到沒被使用的frame代表超出memory的使用範圍了，這時候呼叫ExceptionHandler(MemoryLimitException)來讓螢幕顯示錯誤訊息並終止程式。

machine.h



在指定的位置新增ExceptionType的類別

在exception.cc裡面不需要做MemoryLimitException的case處理，下面default會顯示錯誤的訊息就足夠了

