Operating System

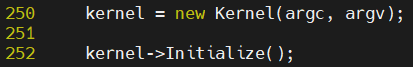
MP2 Report

組別：25

陳凱揚：Trace code、Implement、Report  
簡佩如：Trace code、Implement、Report

1. Trace code
   1. /threads/main.cc 🡪 main()

在執行nachos後，首先會進到main.cc的main()裡，處理command line argument，處理完後會以這些argument建立一個Kernel物件且呼叫Initialize()（圖1-1），並將物件位址存於global variable的kernel中。

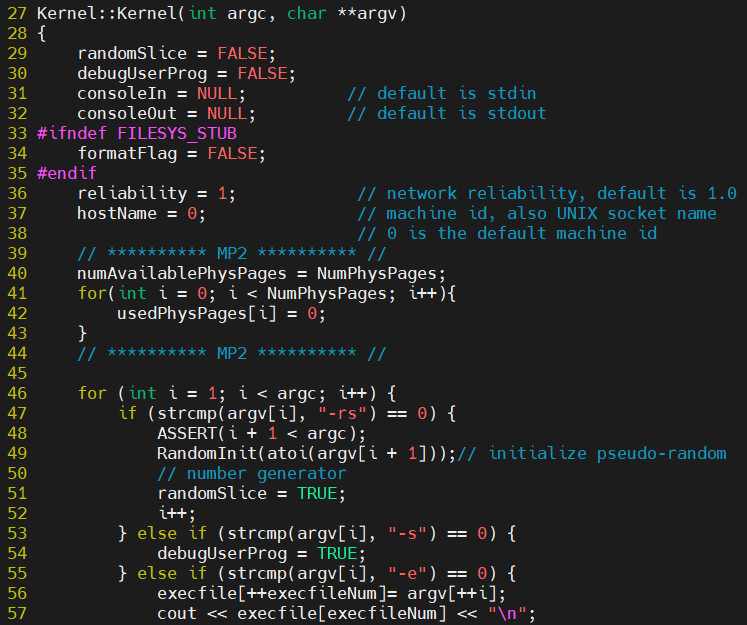


▲ 圖1-1

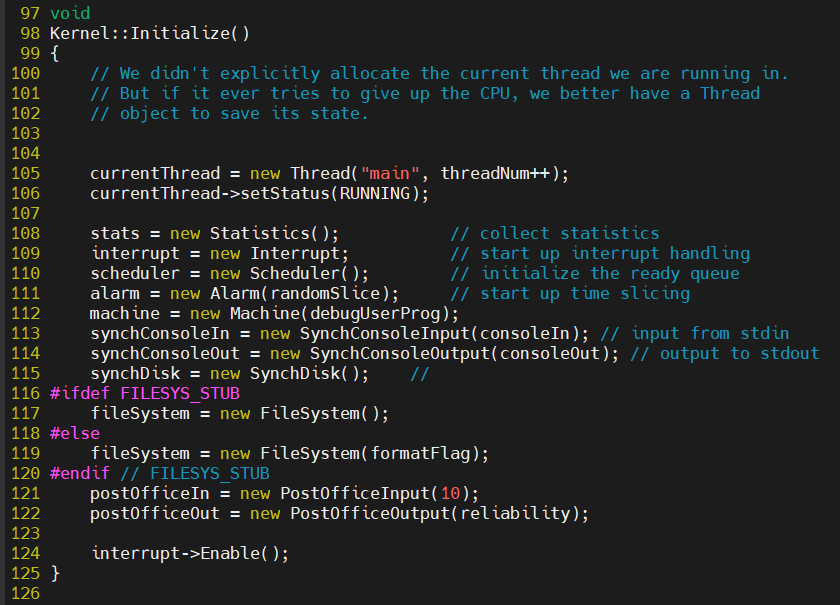
* 1. threads/kernel.cc 🡪 Kernel(), Initialize()

在Kernel的constructor裡（圖1-2），會將一些變數設置初值，並依據傳進的argument有對應的處理，其中特別注意第55行處理的”-e”，會將後面接著的執行檔名稱存入execfile中。

而Initialize()主要是建立Kernel裡所需的其他物件（圖1-3），與constructor分開的原因是因為他需要參考kernel先前設置好的資料。此外，在第105行建立了一個新的thread叫做main，也就是第一個執行的thread。



▲ 圖1-2



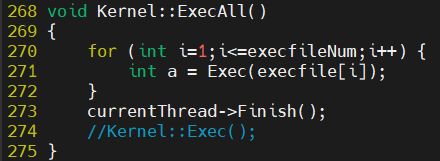
▲ 圖1-3

* 1. /threads/kernel.cc 🡪 ExecAll(), Exec()

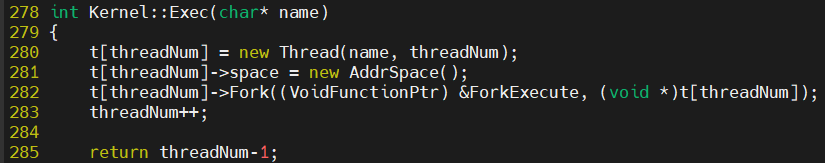
在建好Kernel後，會回到main()繼續向下執行到第288行的ExecAll()（圖1-4），在ExecAll()裡會將剛剛儲存在execfile裡的每個執行檔名稱作為參數，分別執行Exec()，execfileNum指的是有幾個待執行檔（圖1-5）。而Exec()裡的t是一個thread陣列，threadNum會紀錄目前thread數量，接著為每個執行檔建立新的Thread，這個thread會new一個新的Addrspace，並以函數ForkExecute及這個thread本身作為參數呼叫Fork()（圖1-6）。



▲ 圖1-4



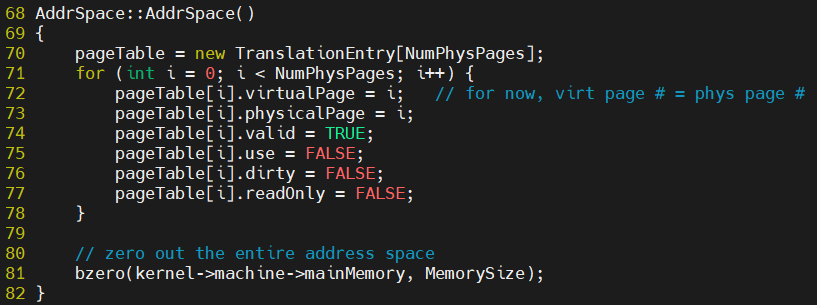
▲ 圖1-5



▲ 圖1-6

* 1. /userprog/addrspace.cc 🡪 AddrSpace()

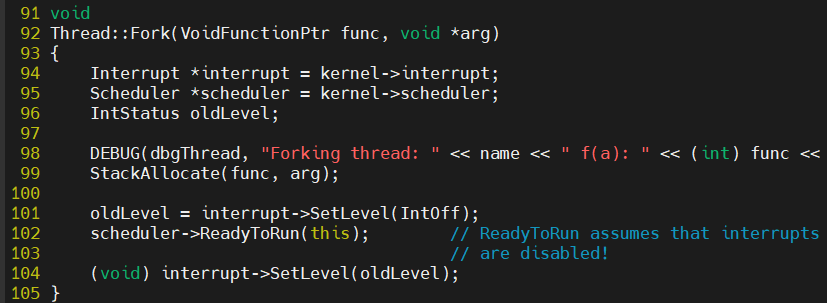
建立一個thread的space也就是建出對應的pageTable，功能為將程式的virtual memory轉換到physical memory，所以在AddrSpace的constructor裡，會建立一個TranslationEntry陣列，並將每個entry中的每個變數都設為預設值（圖1-7），在這裡的physicalPage預設與virtualPage相同，是uni-programming的做法。



▲ 圖1-7

* 1. /threads/thread.cc 🡪 Fork()

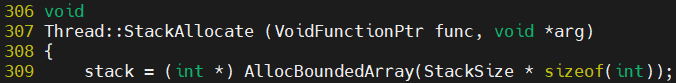
在Fork()裡會先使用StackAllocate()分配一塊stack給現在這個thread，接著儲存interrupt狀態至oldLevel，再設為IntOff（不能被打斷），並使用scheduler的ReadyToRun()將這個thread加進ready queue裡，再將interrupt的狀態回復成oldLevel（圖1-8）。



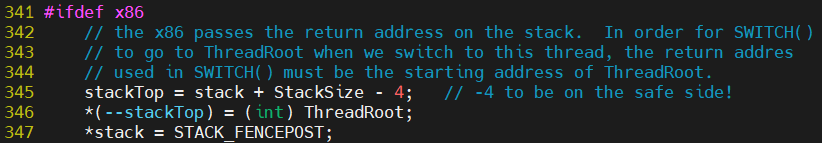
▲ 圖1-8

* 1. /threads/thread.cc 🡪 StackAllocate()

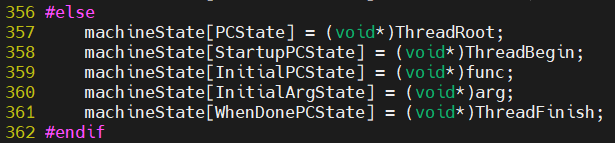
在StackAllocate()裡，會先給這個thread的stack一塊新記憶體（圖1-9），接著會依據不同的CPU架構來執行程式，在這邊是使用x86架構（圖1-10），決定了stackTop的位置、並將ThreadRoot放在stack的第一個frame裡（會被SWITCH()呼叫來開始執行這個thread）、將STACK\_FENCEPOST放在stack的最後一個frame裡（用來判斷是否發生stack overflow），最後會將被fork的func、被傳遞的arg及其他所需function放進machineState中對應的位置（圖1-11）。



▲ 圖1-9



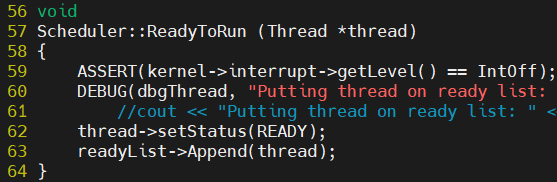
▲ 圖1-10



▲ 圖1-11

* 1. /threads/scheduler.cc 🡪 ReadyToRun()

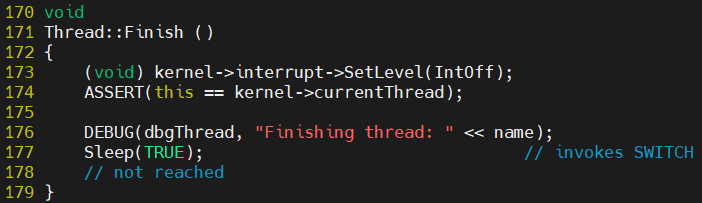
在ReadyToRun()裡，首先會確認interrupt已經被關掉了，再來會將傳進的thread狀態設為ready後，放進readyList的最後面，也就是將這個thread放入ready queue裡等待執行（圖1-12）。



▲ 圖1-12

* 1. /threads/thread.cc 🡪 Finish()

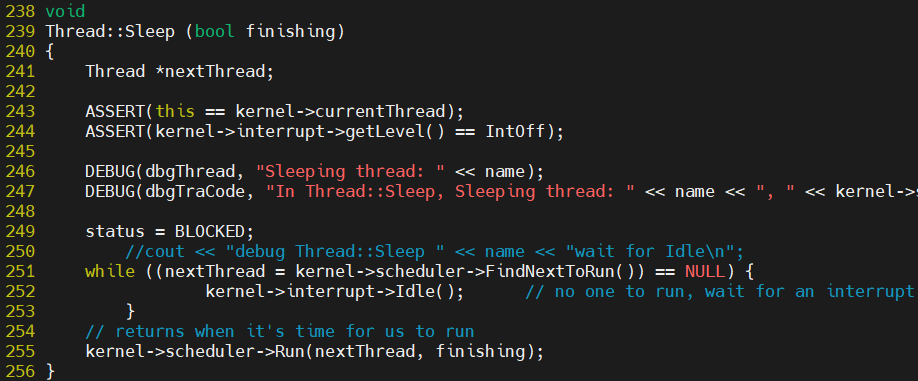
在StackAllocate()、ReadyToRun()、Fork()、Exec()執行完後，會回到ExecAll()裡，執行Finish()，將一開始建立的第一個thread(main)結束掉。而在Finish()裡，首先會將interrupt設定為IntOff（不能被打斷）、檢查是不是currentThread呼叫的，因為只有執行中的thread才能結束自己，接著會呼叫Sleep()，並傳入TRUE作為參數，表示結束這個thread（圖1-13）。



▲ 圖1-13

* 1. /threads/thread.cc 🡪 Sleep()

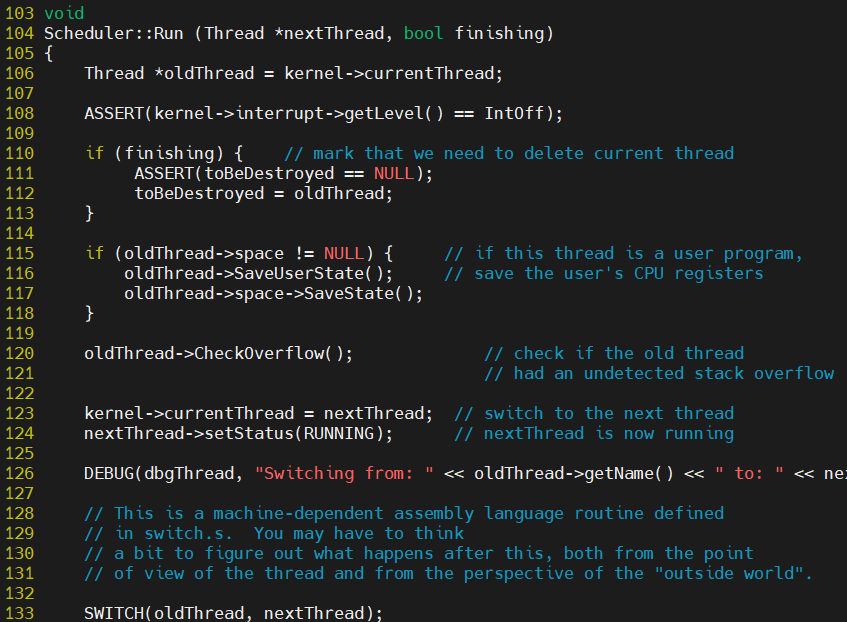
這邊會再次檢查呼叫Sleep的是currentThread，且interrupt為關閉狀態，接著將這個thread狀態設為BLOCKED，並去ready queue找到下一個thread，如果ready queue是空的，就會不斷呼叫Idle()，表示CPU為閒置狀態，並跳到下一個interrupt的時間，直到找到nextThread，就會執行Run()（圖1-14）。



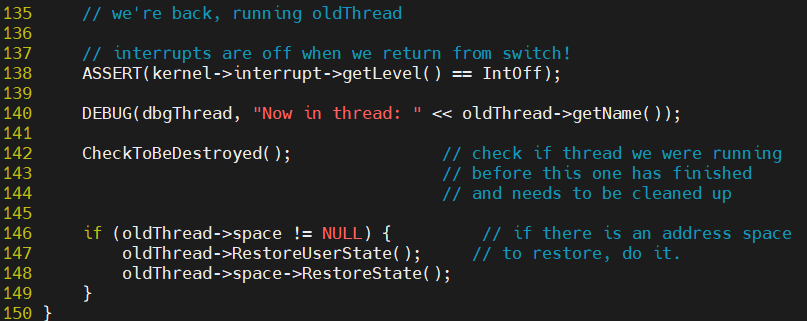
▲ 圖1-14

* 1. /threads/scheduler.cc 🡪 Run()

在Run()裡，功能主要是將目前執行中的thread暫停、並儲存目前狀態，接著使用SWITCH()將傳進來的nextThread轉變成currentThread（圖1-15），而之後回到原本的thread後會檢查是否需要結束這個thread，是的話就刪除掉這個thread，不是的話便回復狀態（圖1-16）。



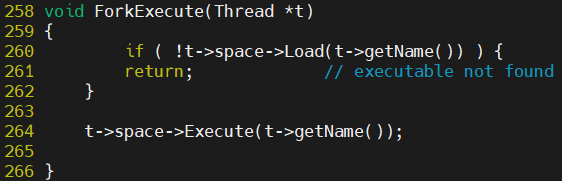
▲ 圖1-15



▲ 圖1-16

* 1. /threads/kernel.cc 🡪 ForkExecute()

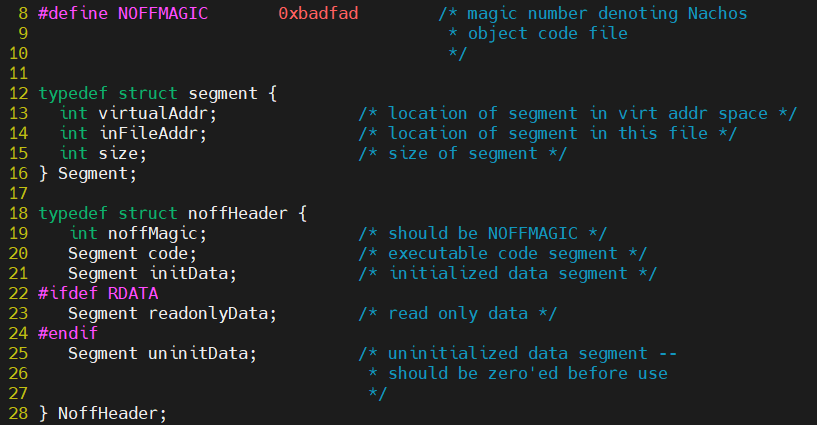
在之前的Exec()呼叫Fork()時，會將ForkExecute當作參數傳入，並且在StackAllocate()時放進machineState[InitialPCState]，而在SWITCH()作thread轉換後，第一個就會執行這個函式，功能是使用Load()取得執行thread所需的記憶體，再呼叫Execute()執行（圖1-17）



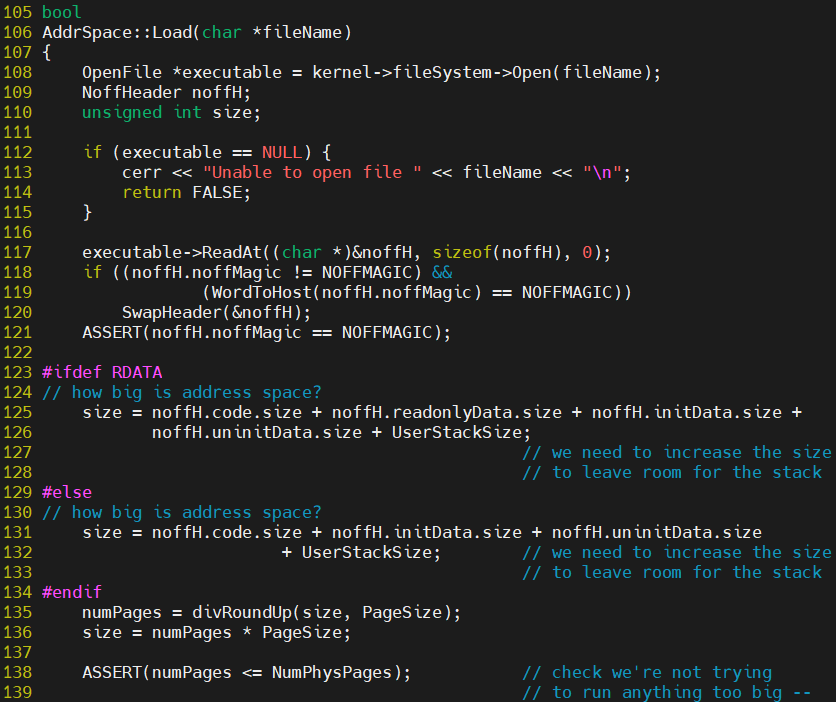
▲ 圖1-17

* 1. /userprog/addrspace.cc 🡪 Load()

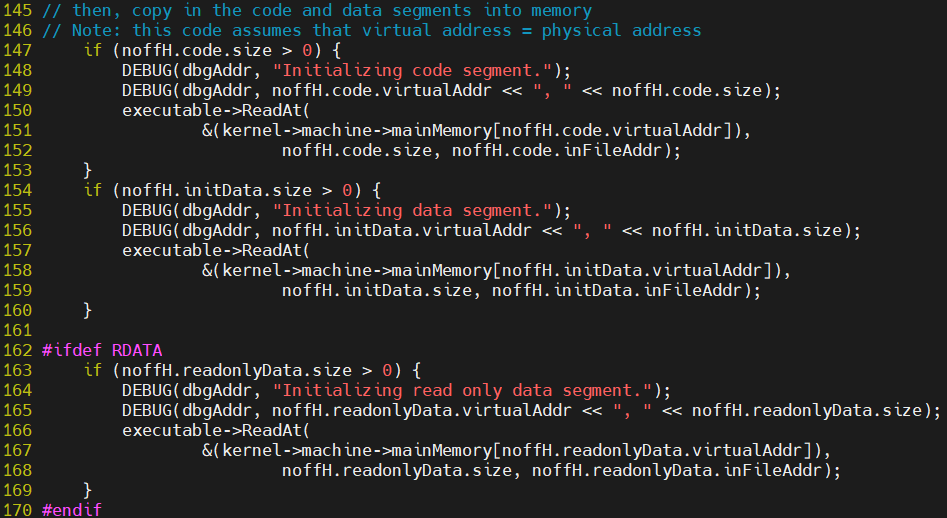
在Load()裡，首先會讀取執行檔最前面的header，並存進NoffHeader資料結構中（圖1-18），包含了code、initData、uninitData的size和address等資訊，再來就能使用這些資訊算出thread所需的size和numPages（圖1-19）。最後將file裡的資料寫進mainMemory中，這邊預設virtualAddr即為physicalAddr（圖1-20），為uni-programming的作法，load成功後會回傳TRUE表示成功。



▲ 圖1-18



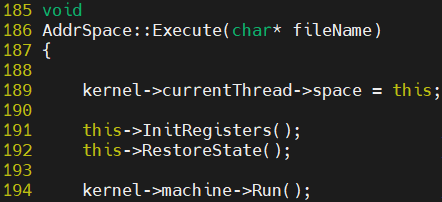
▲ 圖1-19



▲ 圖1-20

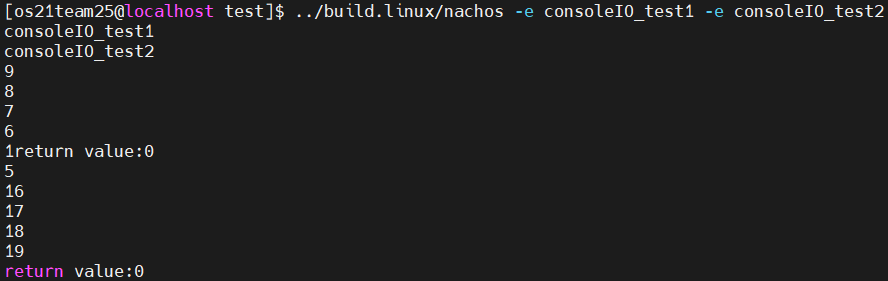
* 1. /userprog/addrspace.cc 🡪 Execute()

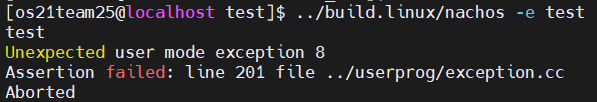
Execute()會將space、machine的register、pageTable調整為這個thread所需的狀態後，就會呼叫machine的Run()，開始fetch instruction（圖1-21）。



▲ 圖1-21

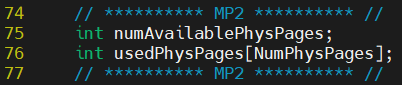
1. Implement





* 1. /threads/kernel.h 🡪 class Kernel{}

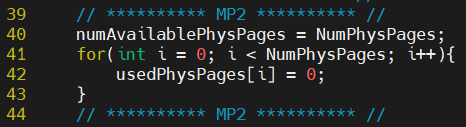
在class Kernel中，新增了兩個成員變數（圖2-1），numAvailablePhysPages記錄目前還有多少可以使用的physical page數量，而陣列usedPhysPages則是記錄每一個physical page有沒有被正在使用，1代表使用中，0代表未使用。



▲ 圖2-1

* 1. /threads/kernel.cc 🡪 Kernel()

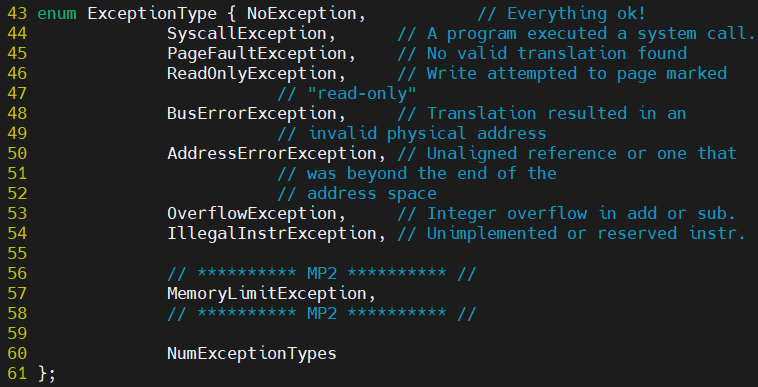
在Kernel的constructor中，numAvailablePhysPages設為NumPhysPages，而所有usedPhysPages都設為0，因為所有physical page都未被使用（圖2-2）。



▲ 圖2-2

* 1. /machine/machine.h 🡪 enum ExcaptionType{}, class Machine{}

在ExceptionType裡會新增一個MemoryLimitException，表示記憶體不足（圖2-3），而在load執行檔進記憶體時，如果發生了記憶體不足或是translate失敗的情況，就會需要呼叫RaiseException，因此會在class Machine裡新增friend class AddrSpace（圖2-4）。



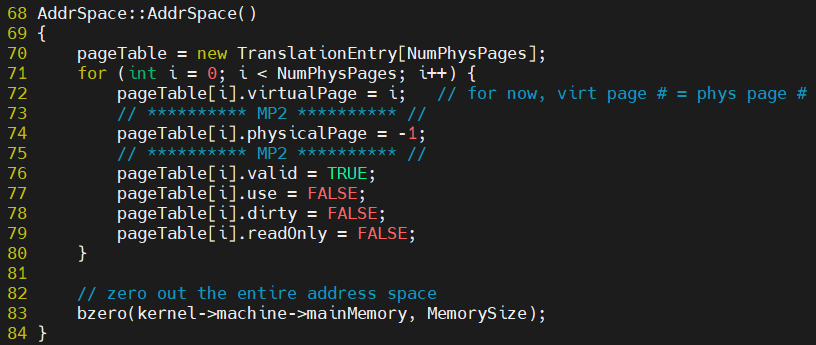
▲ 圖2-3



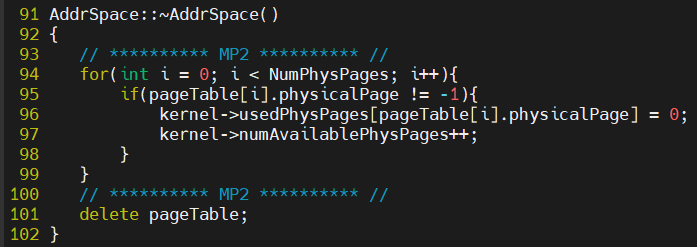
▲ 圖2-4

* 1. /userprog/addrspace.cc 🡪 AddrSpace(), ~AddrSpace()

在AddrSpace的constructor裡，會將physicalPage都設為-1，表示目前還沒有load進memory，所以沒有對應的physicalPage（圖2-5）。而在destructor，則會將所有pageTable中對應到的physicalPage釋出，也就是將usedPhysPages設為0，且增加numAvailablePhysPages（圖2-6）。



▲ 圖2-5

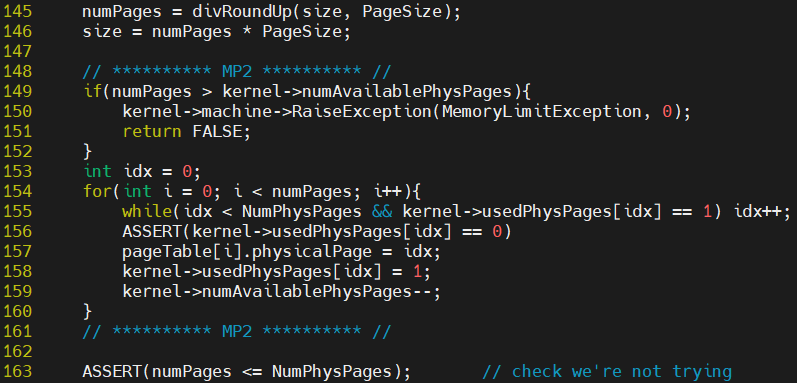


▲ 圖2-6

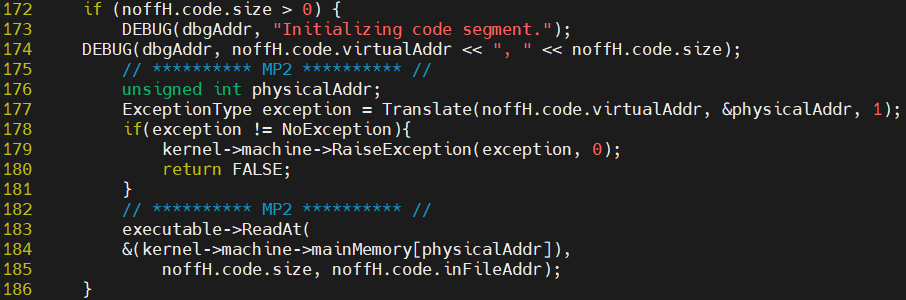
* 1. /userprog/addrspace.cc 🡪 Load()

首先會檢查所需的numPages是否超過可使用的numAvailablePhysPages，如果是的話便會呼叫RaiseException()，傳進MemoryLimitException作為參數，表示記憶體不足，並回傳FALSE，表示load失敗；如果沒有超過，就會開始一個個尋找可用的physPage，總共需要找到numPage個，每找到一個就需要將usedPhysPages設為1，表示使用中，並減少numAvailablePhysPages（圖2-7）。

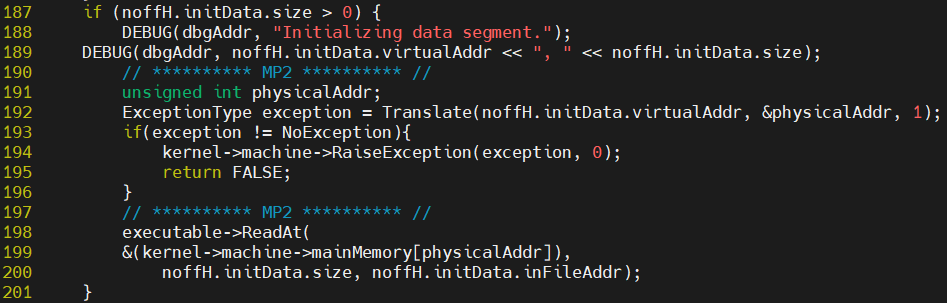
最後分別改寫code（圖2-8）、initData（圖2-9）、readonlyData（圖2-10）在寫入時使用的virtualAddr，可以使用Translate()將virtualAddr透過pageTable轉換成physicalAddr，而如果Translate()過程出現問題會回傳exception的type，此時也需要RaiseException()，並回傳FALSE，表示load失敗。



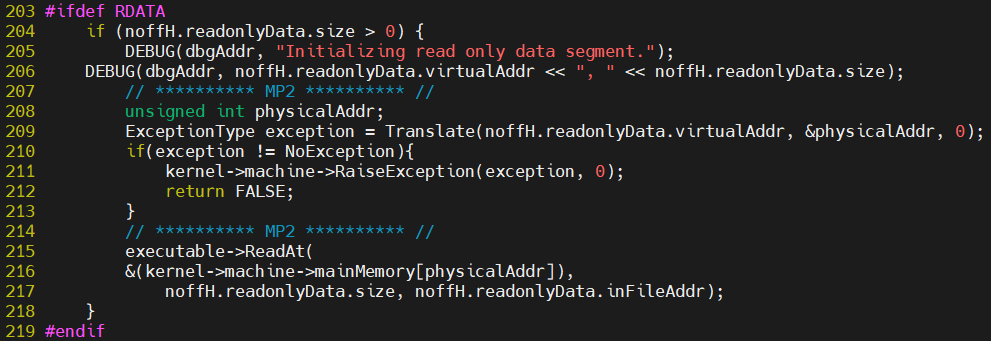
▲ 圖2-7



▲ 圖2-8



▲ 圖2-9



▲ 圖2-10

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

thread.cc裡的Fork()會使用StackAllocate()分配並初始化記憶體空間給新的thread。

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

Nachos在建立新的thread的時候，這個thread會new一個AddrSpace來管理他的pageTable，並設定初始值。在之後Load()時，會去打開我們輸入的binary檔，且先讀取binary檔最前面固定格式的header進NoffHeader資料結構中，就可以知道這個binary檔裡面的code、initData、readonlyData的size、virtualAddr、inFileAddr，最後就可以分別將這些資料從file裡寫進mainMemory中。

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

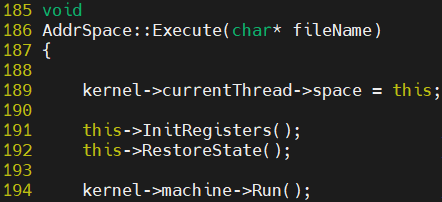
每個thread在建立時，都會new一個AddrSpace來管理pageTable，pageTable是一個TranslationEntry陣列，儲存了每個virtualPage對應到哪一個physicalPage，其中也會儲存一些狀態，如valid、dirty、readonly等，Nachos可以透過查pageTable，將loagical address轉換成physical address。

* 1. How Nachos translates address?

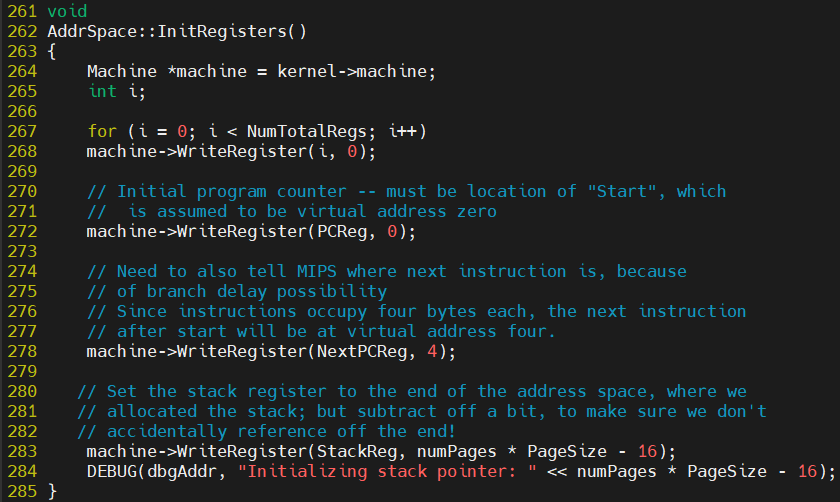
translate.cc裡的Machine::Translate()會使用pageTable或TLB把virtual address轉換成physical address。過程中如果有錯誤的話會回傳錯誤類型，確認沒有錯誤後，會將entry的use設為TRUE，並將算出的physical address存進physAddr裡。

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

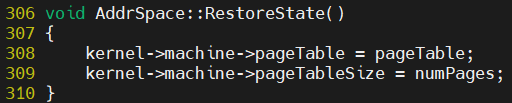
在Execute()裡，最後執行Machine::Run()前，會先將currentThread的space指向現在這個Address（圖3-1）；並且呼叫InitRegisters()，將program counter和stack的位置寫進machine的register，其他register則設為0（圖3-2）；再來呼叫RestoreState()，將machine的pageTable和pageTableSize設為現在這個AddrSpace的pageTable和numPages（圖3-3）。



▲ 圖3-1



▲ 圖3-2



▲ 圖3-3

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

Thread，裡面包含了machine state、stack、state、userRegisters等資料，還有Fork()、Sleep()等功能。

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

Scheduler會管理threads的ready list，當某個thread可以使用CPU的時候就會呼叫ReadyToRun()，把thread的status設成READY，並加進list的最後面，像是在Fork()、Yield()都會呼叫到ReadyToRun()。

1. Difficulties and Feedback

經過了MP1之後對NachOS有比較熟悉了，所以這次作業花了比較少的時間在理解架構。但是有時候trace到組語就停下來，導致沒有辦法trace到函式的盡頭，也因此沒辦法完全掌握它的奧義，比如trace到 SWITCH(oldThread, nextThread)時，就像註解說的，我們只能view of the thread and from the perspective of the "outside world"，有點可惜，也算是在寫作中遇到最大的困難。上課的時候花了很多時間在介紹memory的管理，雖然聽課的時候就覺得很神奇了，但是真正實作的感覺就像是看著說明書組樂高一樣，感覺更理解這個章節了。