Operating systems

MP2

Team 67

108062128林育丞

108062135呂佳恩

分工表

|  |  |
| --- | --- |
| 林育丞 | Trace Code, Question回答整理, Report整理 |
| 呂佳恩 | Trace Code, Implementation, Report整理 |

1. Trace Code

Threads/thread.cc

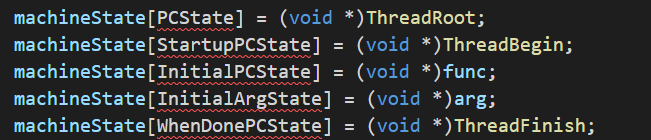
1. Thread::Sleep( bool finishing )

在兩個狀況下，Sleep() 會被呼叫 :

1. Current thread is finished, the argument passed in will be “TRUE”
2. Current thread blocked waiting on a synchronization variable, the argument passed in will be “False”

Sleep 會進入一個判斷式，若有新的Thread 在 Scheduler 當中，會Run, 若沒有，FindNextToRun()會回傳NULL，此時進入While迴圈，讓kernel 進入 Idle

1. Thread::StackAllocate( VoidFuntionPtr func, void \*arg )
   1. Func是指 procedure to be forked, arg 是將要被 passed through的arguments
   2. Call AllocBoundedArray( ) allocate 所需要的空間
   3. #ifdef directive **allows for conditional compilation**
   4. 利用下面的code 設定correct Machine state



* 1. 將PC 設定丞指向rounte Threadroot, Thread的執行並非在user provided 的routine 而是在 routine ThreadRoot執行

1. Thread::Finish ( )
   1. 當一個 Thread 完丞executing Fork時，由ThreadRoot 呼叫
   2. 將Interrupt 設定為 Off, 因為 Sleep 當中，在OS將thread 從Ready List 到 switch至他開始執行當中，不能有interrupt 使兩者當中出現time slice.
   3. 呼叫Sleep 並傳入 finishing 為TRUE
2. Thread::Fork ( VoidFuntionPtr func, void \*arg )
   1. Func 是指procedure在thread 要開始執行後的procedure address，而
   2. 呼叫 StackAllocate( ) 讓其將空間以及初始畫作完
   3. Set interrupt to off
   4. Put current thread into scheduler for run **(ReadyToRun)**
   5. Set interrupt back to the original value
3. ThreadRoot
   1. 以Assembly Language寫成
   2. 會呼叫startup function (即Thread::Begin)和讀取初始參數(在StackAllocate時候被記錄，即ForkExecute和指向該thread的指標)，以及讀取最後該呼叫的Thread::Finish。  
      (示意圖，實際執行情形不一定會是這個 #ifdef 的 #else)一張含有 文字 的圖片

      自動產生的描述
   3. 在main thread SWITCH之後，會呼叫 thread::Finish(), 將thread terminate, 再以此函式為開端，讓OS繼續執行其他thread。(若沒有這個動作，則會讓main thread一路return，從Sleep -> Finish -> ExceAll -> main -> ASSERTNOTREACHED，而中止程式。
4. Switch
   1. 以Assembly Language寫成
   2. 移動Stack pointer，讓他從舊的thread指向新的 (t1 -> t2)
5. Thread::CheckOverflow( )
   1. 如果Stack overflowed，則程式中斷(assertion failed)

Userprog/addrspace.cc

1. AddrSpace::AddrSpace( )

原版未修改的code 當中，virtual memory 被 1:1對應到physical memory當中

1. Create Entry for a new thread in PageTable
2. Set the initial values for the Thread
3. Zero out the entire address space (bzero : set N bytes to zero)
4. AddrSpace::Execute( Char \*filename )
   1. Assumes the program is already loaded into the address space
   2. Set register by calling Init Registers and RestoreState
   3. Call Machine::Run to jump to the user program to run the program.
5. AddrSpace::Load( Char \*filename )
   1. Use the FileSystem to Open the File
   2. Use NoffHeader去判斷大小，以及其他資訊，再讀進memory

Threads/kernel.cc

1. Kernel::Kernel( int argc, char \*\*argv )
   1. 由 main.cc去呼叫
   2. Use strcmp to compare different input values and do the according actions
   3. Execfile[ ++execfilenum ]會記錄 -e 參數的後面一個參數的名稱
2. Kernel::ExecAll( )
   1. Call Exec for each file that needs execution
   2. Call finish when the loop ends (Finishing = TRUE)

(讓當前的Thread (即main thread) 進入睡眠，以觸發**S**W**I**T**C**H)

1. Kernel::Exec( char \*name )
   1. Create new thread by the 檔案名稱與執行緒編號
   2. Allocate the space for phy to virtual memory translation
   3. Fork
2. Kernel::ForkExecute( Thread \*t )
   1. Load information for space
   2. Call Execute if the executable is found

Threads/scheduler.cc

1. Scheduler::ReadyToRun( Thread \*thread )
   * + 1. Set this thread to Ready state
       2. Append it on the ready list ( Fork的時候被呼叫)
2. Scheduler::Run( Thread \*nextThread, bool finishing)
   * 1. Check if finishing, if true, it destroys the current thread
     2. Check if thread is user program, if so, save CPU regs

(main thread的space是null，所以不是user program，不需要保存資料)

* + 1. CheckOverflow()
    2. Change to nextThread and set state to Running
    3. Call SWITCH to stop current thread and start new thread  
       SWITCH在switch.S中被Implement  
       主執行緒進入SWITCH最後一句話執行時CPU的返回值指向ThreadRoot函式，子執行緒進入SWITCH最後一句話執行時CPU的返回值指向Scheduler::Run函式中的SWITCH函式語句。
    4. Call CheckToBeDestroyed if there is any toBeDestroyed thread, if so, delete IT  
       並沒有在第一時間就刪除，是為了完成SWITCH的程序。
    5. Try to restore available address spaces.

(II). Questions

Q1 : How Nachos allocates the memory space for new thread(process)?

A : 在Fork的時候，會new一個AddrSpace給thread->space。

Q2 : How Nachos initializes the memory content of a thread(process), including

loading the user binary code in the memory?

A : 使用AddrSpace::Load()，將讀取的file切進記憶體。

Q3 : How Nachos creates and manages the page table?

A :

1. 實作之前是直接在建構式宣告一個NumPhysPages大小的page table。
2. 實作之後則是把建立的這件事，等到Load的時候，再依據讀取進來的程式大小，宣告適當尺寸的page table。
3. 管理的部分，則是Load在宣告完page table之後，會接續進行。

Q4 : How Nachos translates address?

A : 1. translate.cc中的machine::Translate

\*physAddr = pageFrame \* PageSize + offset;

2. AddrSpace.cc中 (在實作後)

Virtual page number = virtualAddr / PageSize

Page內的Offset = virtualAddr%PageSize

Q5 : How Nachos initializes the machine status (registers, etc) before running a

thread(process)

A : 在Thread::StackAllocate() 的時候，會把自己的ThreadRoot() 與相關資訊存進Stack裡面。相關資訊包含：ThreadBegin、ThreadFinish、該thread的ForkExecute函式指標，與指向自己的指標 (Thread \*)。

Q6 : Which object in Nachos acts the role of process control block

A : Thread (這一個class)。因為它包含了page table、register等資訊。

Q7 : When and how does a thread get added into the ReadyToRun queue of Nachos CPU scheduler?

A : 在Thread::Fork() 裡面，會呼叫Scheduler::ReadyToRun( )，並在此時把這個thread 放進readyList裡面。

(III) Implementation

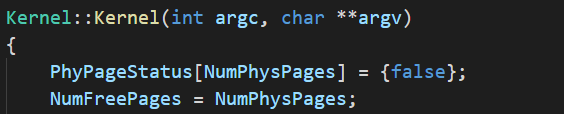
Kernel.h

Add two members in the Kernel class,

PhyPageStatus records whether if this page is in use or not.

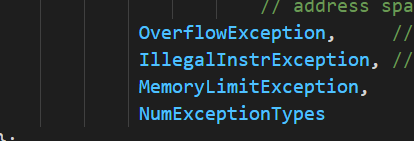
NumFreePages records the number of pages that are free.

Kernel.cc

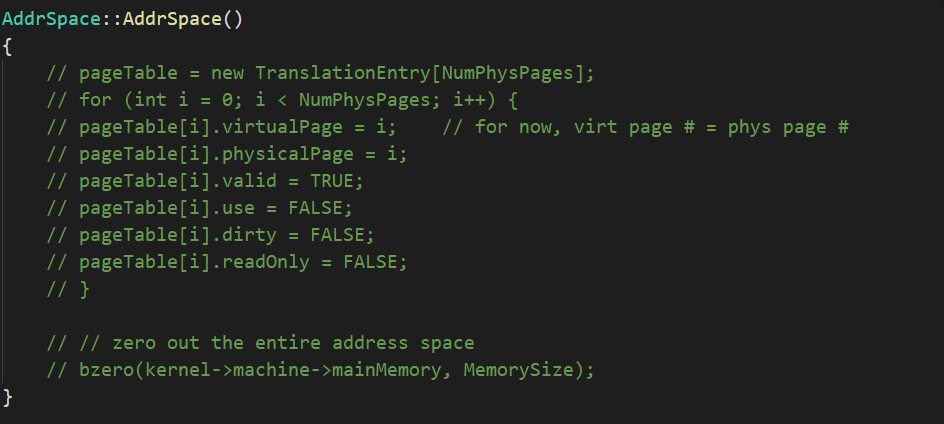


Add the initial values of the two members in the constructor of Kernel.

Set all off PhyPageStatus to false since none are in use when initialization, and the value of NumFreePages to the value of NumPhysPages since all are free among initialization.

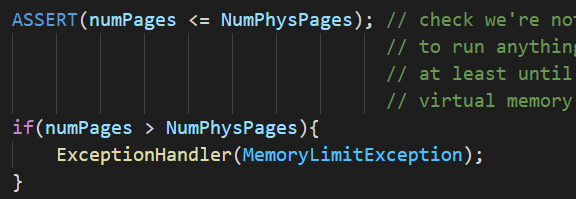
Machine.h

In order to handle MemoryLimitException, we add it into the ExceptionType

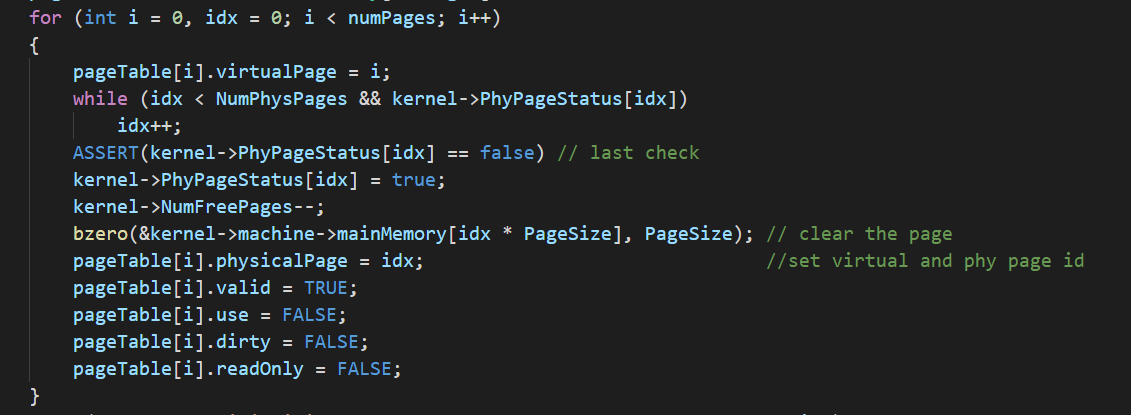
Addrspace.cc

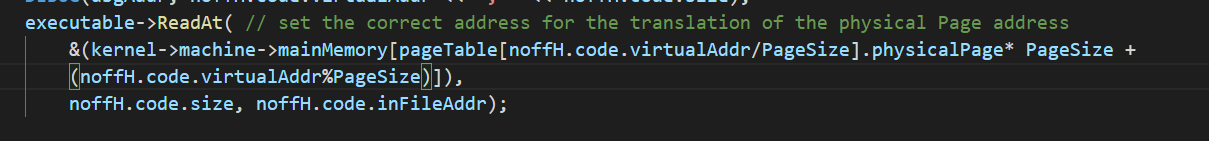
We comment all the contents of AddrSpace, we will set the values when we in AddrSpace::Load, the reason is that we need to know how many Pages does this thread need in order to set the pageTable, we could also handle the part where a program size exceed a pagesize.



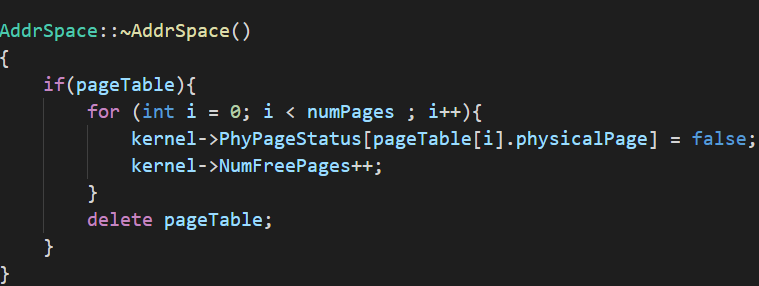


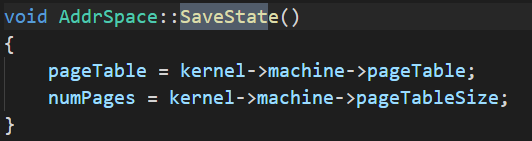
We first handle the exception that the Memory Limit is exceeded.

 Then we set the required value, it is similar to the constructor of AddrSpace but we use a for loop to get the number of how many pages the thread needs.



We then set noffH.code.virtualAddr to the PhysicalPage address.

 Modify the destructor with the addition of the two members that we added.



Modify the SaveState in order to record the two members that we added.