Content

[I. Motivation 2](#_Toc57723760)

[Motivation and the problem analysis 2](#_Toc57723761)

[What’s your plan to deal with the problem 2](#_Toc57723762)

[II. Implementation 3](#_Toc57723763)

[How do you implement to solve the problem in Nachos 3](#_Toc57723764)

[You can including some important code segments and comments 3](#_Toc57723765)

[III. Result 6](#_Toc57723766)

[Experiment result and some discussion 6](#_Toc57723767)

Reference

1. MrOrz/nachos. [Online]. Available: <https://github.com/MrOrz/nachos>

<https://github.com/taldehyde/os-project-1/blob/master/report.md>

1. Caching and Virtual Memory. [Online]. Available: <http://neuron.csie.ntust.edu.tw/homework/93/OS/homework_3/A9315010-%E7%AC%AC%E4%B9%9D%E7%B5%84/phase%203%20operation.htm>

# Motivation

## Motivation and the problem analysis

* 若在還沒有改nachos-4.0的原始碼以前，同時執行 兩個program
  + ./nachos -e ../test/matmult -e ../test/sort

這時會因為超過預先設定的physical page number (預設為32)，而產生core dump。

如下圖所示：

|  |
| --- |
| Fig. 1 修改前所遇到的問題  由於matmult與sort兩者thread所佔用到的physical page會超過預設的32導致Aborted |

原先nachos一次只保留一個user program在主記憶體中。

此專題目標在於使用paging mechanism來實作multiprogramming，

也就是在主記憶體中保留多個user programs，而pages只有在這個program被執行的時候才會被loaded，也就是pure demand paging。

## What’s your plan to deal with the problem

以Program matmult與Program sort 為例。

輸入nachos -e ../test/matmult -e ../test/sort之後

1. 先幫這兩個Program各創一個thread與page table，也就是生成thread A與thread B。
2. 利用AddrSpace::Load() 函式把Program matmult與Program sort之page table匯入進硬碟中，同時也把這兩個thread丟進ready queue裡面，這個時候OS會決定要讀取的程式碼在的起始地址與總共要載入多少分頁到硬碟裡。當作業系統開始執行後(執行kernel->machine->Run()後)，選擇其中一個thread來執行。當在page Table中找不到要跑的thread時，發生page fault，並去磁碟機者到對應的page (demanded page)與執行context switching(timer讓interrupt取引發context switching)，開始執行另外一個thread。

在此專題中沒有去製造新的Memory Manager，而是把virtual address與physical address的對應關係寫在AddrSpace::Load()裡，利用Page Size跟virtual address去找對應physical address。

排程的page replacement 使用 LRU(Least Recently Used).

同時，這邊使用3種table來維護physical page資訊與swap進入或換出主記憶體的資訊。

|  |  |
| --- | --- |
| PageTable | One page table per process.  Decide your virtual page number.  Kernel->machine->pageTable = currentThread ->pageTable |
| FrameTable | 整個system一個 紀錄每個physical page的資訊 |
| SwapTable | 整個system一個 紀錄swap內每個sector的資訊 |

# Implementation

## How do you implement to solve the problem in Nachos

相較於第二個project，改動程式碼檔案如下表：

|  |  |
| --- | --- |
| /userprog/userkernel.h  /userprog/userkernel.cc | 新增swapDisk |
| / userprog/exception.cc | 新增PageFaultException |
| /userprog/addrspace.h  /userprog/addrspace.cc | Address Mapping  檢查有沒有閒置的physical page |
| /machine/translate.h  /machine/translate.cc | 實作LRU |
| /machine/machine.h | 紀錄 sector number, frame paged  和被main memory, virtual memory  占用的frame |

## You can including some important code segments and comments

|  |  |  |  |
| --- | --- | --- | --- |
| 新增swapDisk | | | |
|  | | | |
| Original Text | | Changed Text | |
| /code/userprog/userkernel.h | | | |
|  | | | |
| Original Text | Changed Text | | |
| /code/userprog/userkernel.cc | | | |
|  | | | |
| Changed Text | | |
| /code/userprog/exception.cc | | | |

------------------------------------------------------------------------------------------------------------------------

將程式載入虛擬記憶體，也就是磁碟機中。載入的同時會紀錄virtual page的起始位置還有要載入多少分頁到磁碟機中。然而如果可以用的physical page已經沒有了，則會把這段程式碼Load進virtual page裡面，並把它寫到swap區域中。

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/userprog/addrspace.h | |

|  |  |
| --- | --- |
| ---    --- | |
| Original Text | Changed Text |
| /code/userprog/addrspace.cc  主要更改在Load function.   1. 主要執行程式碼會allocated to main memory，相關page table紀錄. 2. 如果 frame number in main memory 足夠，the pages 會存在main memory. ;   如果不夠，最後一個 pages 會寫入 virtual memory by Write Sector.  相關 page table 會被記錄並設定為 invalid. | |

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/machine/translate.h  Add ID and LRU\_times in class TranslationEntry | |

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/machine/translate.cc  **LRU implement**.  將page table的valid bit以確保page的位置，i.e. in main memory or in virtual memory.  如果page在virtual memory，檢查main memory中現在是否有可用空間。  如果main memory沒有空間，將發生page替換。  2 buffers拿來宣告swap in/out.。LRU，使用for loop來查找least used page  靠implied by bcopy, ReadSector, and WriteSector.完成  讀取virtual memory， 並複製到main memory，選擇the swap out page 到virtual memory | |

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/machine/machine.h  紀錄 sector number, frame paged, 和被main memory, virtual memory 占用的frame. | |

# Result

## Experiment result and some discussion

1. 以Random replacement方式同時執行/test/matmult.c與/test/sort.c

./nachos -e ../test/matmult -e ../test/sort

|  |
| --- |
| …      Fig. 2 修改後，由於virtual address與physical address的維護加上random replacement的方式，可以成功執行，並輸出7220 (matmult) 與1023(sort) |

**Conclusion**

Page Fault 發生的次數和排程的演算法關係非常的大，這次先使用隨機的方式找到victim，之後可以再進一步的測試在課堂上有交到過的各種演算法，來測試不同演算法發生page fault的次數，來決定最佳的排成方式，找到一個最好的結果。