Content

[I. Motivation 2](#_Toc57656279)

[Motivation and the problem analysis 2](#_Toc57656280)

[What’s your plan to deal with the problem 2](#_Toc57656281)

[II. Implementation 3](#_Toc57656282)

[You can including some important code segments and comments 3](#_Toc57656283)

[III. Result 6](#_Toc57656284)

[Experiment result and some discussion 6](#_Toc57656285)

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各創立一個thread與他們的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。

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/FrameInfoEntry.h | 適用於FrameTable跟SwapTable  一個frame代表一個physical page |

## You can including some important code segments and comments

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。

/code/userprog/addrspace.cc (Line 186~253)

if (noffH.code.size > 0) {

bool has\_free\_physical\_page = true; // flag

char \*buf = new char[PageSize]; // a buffer to read code from disk

for (unsigned int i=0,j=0,k=0; i < numPages; i++) { // find available virtual page

pageTable[i].virtualPage = i; // for now, virt page # = phys page #

// find available physical page

if(has\_free\_physical\_page){

// usedPhyPage has only NumPhysPages entries.

while(usedPhyPage[j]!=FALSE && j < NumPhysPages ) ++j;

if(j >= NumPhysPages){ // no free physical page.

has\_free\_physical\_page = false;

}

}

// if physical page j < NumPhysPages is found

if(has\_free\_physical\_page){

ptrPageTable[j] = &pageTable[i];

usedPhyPage[j]=TRUE;

pageType[j] = 1;

pageTable[i].physicalPage = j;

pageTable[i].valid = TRUE;

pageTable[i].use = FALSE;

pageTable[i].dirty = FALSE;

pageTable[i].readOnly = FALSE;

executable->ReadAt( &(kernel->machine->mainMemory[PageSize\*j] ), PageSize, noffH.code.inFileAddr + (PageSize\*i) );

}

// already ran out of physical pages

else

{

// find available virtual page

while(usedVirPage[k]!=FALSE) ++k;

usedVirPage[k]=TRUE;

pageTable[i].virtualPage = k; // virtual page id

pageTable[i].valid = FALSE; // not in physical memory now!

pageTable[i].use = FALSE;

pageTable[i].dirty = FALSE;

pageTable[i].readOnly = FALSE;

// read this page of code into a buffer, then write to swap space

executable->ReadAt(buf, PageSize, noffH.code.inFileAddr + (PageSize \* i));

kernel->swapDisk->WriteSector(k, buf);

}

}

delete[] buf; // buf is useless now, delete it.

}

這一段紀錄virtual address到physical address的對應 (Address Mapping)

/code/userprog/addrspace.cc (Line 255~274)

if (noffH.initData.size > 0) {

int virtualPageN = noffH.initData.virtualAddr/PageSize;

int virtualPageLocation = noffH.initData.virtualAddr% PageSize;

int physicalPageN = pageTable[virtualPageN].physicalPage;

int memoryLocation = physicalPageN \* PageSize + virtualPageLocation;

//virtualPageLocation=physicalPageLocation

executable->ReadAt(&(kernel->machine->

mainMemory[memoryLocation]),noffH.initData.size,noffH.initData.inFileAddr);

}

Page-fault exception handler。(實作Random replacement)

SwapTable 🡪 ptrPageTable

FrameTable 🡪 pageTable

先隨機選擇一個victim，再把ptrPageTable中vpn (vpn指的是哪一個virtial page被放在這個frane) 與pageTable的victim做交換，完成page-fault handle

/code/userprog/addrspace.cc (Line 382~385 & Line 399~423)

void AddrSpace::pageFaultHandle(int badVAddrReg){

// TODO: handle pagefault exception here!

DEBUG(dbgPageFault, "Page fault ocurred.");

kernel->stats->numPageFaults++;

unsigned int i = 0;

int vpn = (unsigned) badVAddrReg / PageSize;

…

else{

char \*buffer1 = new char[PageSize];

char \*buffer2 = new char[PageSize];

int victim;

while(1){

victim = rand() % NumPhysPages;

if(pageType[victim]==1)break;

}

DEBUG(dbgPageFault, "Page "<<victim<<" swapping out.");

bcopy(&kernel->machine->mainMemory[victim \* PageSize], buffer1, PageSize);

kernel->swapDisk->ReadSector(pageTable[vpn].virtualPage, buffer2);

bcopy(buffer2, &kernel->machine->mainMemory[victim \* PageSize], PageSize);

kernel->swapDisk->WriteSector(pageTable[vpn].virtualPage, buffer1);

ptrPageTable[victim]->virtualPage = pageTable[vpn].virtualPage;

ptrPageTable[victim]->valid = FALSE;

ptrPageTable[victim] = &pageTable[vpn];

pageTable[vpn].valid = TRUE;

pageTable[vpn].physicalPage = victim;

DEBUG(dbgPageFault, "Page swap done.");

}

}

# Result

## Experiment result and some discussion

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

|  |
| --- |
| 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) |

2. 使用Nachos內建的執行多個檔案測試

./nachos -e ../test/matmult -e ./test/sort -e ./test/test1 -e ./test/test2

|  |
| --- |
| Fig. 3執行test1、test2、matmult、sort使用 Random Page Replacement |

Page Fault 發生的次數和排程的演算法關係非常的大，這次專題先使用了隨機的方式找到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。