Content

Reference

1. coffeered Nachos (解決 dependency 問題)
https://github.com/coffeered/Nachos
1. Nachos Project1 Report
https://github.com/taldehyde/os-project-1/blob/master/report.md
2. OS::NachOS::HW1
http://blog.terrynini.tw/tw/OS-NachOS-HW1/
3. 向 NachOS 4.0 作業進發 (1)
http://morris821028.github.io/2014/05/24/lesson/hw-nachos4/

I. Motivation

- Motivation and the problem analysis
- What's your plan to deal with the problem?

The result is not congruent with expected.

```
r08921005@r08921005-VirtualBox:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:6
Print integer:7
Print integer:8
Print integer:9
Print integer:10
Print integer:12
Print integer:13
Print integer:14
Print integer:15
Print integer:16
Print integer:16
Print integer:17
Print integer:18
Print integer:19
Print integer:20
Print integer:17
Print integer:18
Print integer:19
Print integer:20
Print integer:21
Print integer:21
Print integer:23
Print integer:24
Print integer:25
return value:0
Print integer:26
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

首先我先檢視了 test1.c 與 test2.c 兩個檔案，分別為下：

```c
#include "syscall.h"
main()
        {
                int     n;
                for (n=9;n>5;n--)
                        PrintInt(n);
        }
```

test1.c

應是遞減輸出從 9~6

```c
#include "syscall.h"

main()
        {
                int     n;
                for (n=20;n<=25;n++)
                        PrintInt(n);
        }
```

test2.c

應是遞增輸出 20~25

輸出的結果是錯的

原本的 NachOS 並沒有為多個程式做記憶體的管理，雖然有開啟虛擬記憶體，但是基本上沒有作為，所以當多份程式同時執行時就會重疊到其他程式正在使用的 pageTable，然後發生錯誤。

先備知識:

將「 Virtual Address 」對應到「 Physical Address 」。

Virtual Address -> process 執行的過程中存取資料的地方 / process 在看的記憶體

Physical Address -> 電腦實際儲放資料的地方 / 真正的記憶體位置

Trace the following files and find out why the result is wrong

userkernel.cc

```
void
UserProgKernel::Run()
{

        cout << "Total threads number is " << execfileNum << endl;
        for (int n=1;n<=execfileNum;n++)
                {
                t[n] = new Thread(execfile[n]);
                t[n]->space = new AddrSpace();
                t[n]->Fork((VoidFunctionPtr) &ForkExecute, (void *)t[n]);
                cout << "Thread " << execfile[n] << " is executing." << endl;
                }
```

理論上兩個 thread 應該要使用不同的 pageTable，但原先 NachOS 的程式碼並沒有做此設定，因而導致兩個 thread 用到的是同一個 pageTable，才會發生此錯誤。

為此要修改 nachos-4.0/code/userprog/addrspace.cc, nachos-4.0/code/userprog/addr-space.h，使程式的虛擬記憶體映射到沒有人使用的實體記憶體，而不是互相糾纏。

Problem Formulation

1. 記憶體浪費：

   當建立一個 thread 時，pageTable 直接宣告了最大的 Physical Page (NumPhysPages)

2. 不當 thread 管理：

   當建立多個 thread 時，每個 thread 的程式碼可能對應到了相同的 physical page

addrspace.cc

```
AddrSpace::AddrSpace()
{
    pageTable = new TranslationEntry[NumPhysPages];
    for (unsigned int i = 0; i < NumPhysPages; i++) {
    pageTable[i].virtualPage = i;   // for now, virt page # = phys page #
    pageTable[i].physicalPage = i;
//  pageTable[i].physicalPage = 0;
    pageTable[i].valid = TRUE;
//  pageTable[i].valid = FALSE;
    pageTable[i].use = FALSE;
    pageTable[i].dirty = FALSE;
    pageTable[i].readOnly = FALSE;
    }

    // zero out the entire address space
//    bzero(kernel->machine->mainMemory, MemorySize);
}
```

II. Implementation

- How do you implement to solve the problem in Nachos?
- You can include some (not all) important code segments and comments

輸出 n 的結果錯誤，猜測為 context-switch 上發生問題。

因此，需要為 physical pages 做標記。因此預計在 addrspace.h 新增 static 變數

∵static 是屬於 class 共享的

1. 用來記錄所有 physical pages 的使用狀況
2. 知道當前還有多少 physical pages 是可以用的

首先先在 addrspace.h 新增變數 static bool usedPhysPages 紀錄哪些是被使用過的

addrspace.h

```
#ifndef ADDRSPACE_H
#define ADDRSPACE_H

#include "copyright.h"
#include "filesys.h"
#include <string.h>

#define UserStackSize       1024    // increase this as necessary!

class AddrSpace {
  public:
    AddrSpace();            // Create an address space.
    ~AddrSpace();           // De-allocate an address space

    void Execute(char *fileName);   // Run the the program
                    // stored in the file "executable"

    void SaveState();           // Save/restore address space-specific
    void RestoreState();        // info on a context switch

  private:
    TranslationEntry *pageTable;    // Assume linear page table translation
                    // for now!
    unsigned int numPages;      // Number of pages in the virtual
                    // address space

    bool Load(char *fileName);      // Load the program into memory
                    // return false if not found

    void InitRegisters();       // Initialize user-level CPU registers,
                    // before jumping to user code
    static bool usedPhysPages[]; // an array records used physical pages

};

#endif // ADDRSPACE_H
```

而 static 變數一定要在 addrspace.cc 中做過一次初始化

addrspace.cc

```
18  #include "copyright.h"
19  #include "main.h"
20  #include "addrspace.h"
21  #include "machine.h"
22  #include "noff.h"
23
24  //----------------------------------------------------------------------
25  // SwapHeader
26  //  Do little endian to big endian conversion on the bytes in the
27  //  object file header, in case the file was generated on a little
28  //  endian machine, and we're now running on a big endian machine.
29  //----------------------------------------------------------------------
30  bool AddrSpace::usedPhysPages[NumPhysPages] = {FALSE};
31
```

在 addrspace.cc 中，constructor 直宣告一個跟實體記憶體一樣大的 pageTable，但是其實程式沒這麼大，只要分配到跟程式一樣大就好了，然而程式大小必須讀了檔案格式才會知道，所以可以把這整段 code 註解掉，把映射記憶體這件事移到 AddrSpace::Load 中。

addrspace.cc

```
55  AddrSpace::AddrSpace()
56  {/*
57      pageTable = new TranslationEntry[NumPhysPages];
58      for (unsigned int i = 0; i < NumPhysPages; i++) {
59      pageTable[i].virtualPage = i;   // for now, virt page # = phys page #
60      pageTable[i].physicalPage = i;
61  //  pageTable[i].physicalPage = 0;
62      pageTable[i].valid = TRUE;
63  //  pageTable[i].valid = FALSE;
64      pageTable[i].use = FALSE;
65      pageTable[i].dirty = FALSE;
66      pageTable[i].readOnly = FALSE;
67      }*/
68
69      // zero out the entire address space
70  //    bzero(kernel->machine->mainMemory, MemorySize);
71  }
```

到 addrspace.cc 的 AddrSpace::Load 去重新分配 physicalPage，同時維護剛剛新增的變數，每次要 assign physicalPage 前，先去新增的變數檢查看該位置是否已經被別的 physical address 所佔據，如果是，則程式會去找下一個未被使用的 physicalPage，並把它 assign 給特定的 virtual page。

addrspace.cc

```
94  bool
95  AddrSpace::Load(char *fileName)
96  {
97      OpenFile *executable = kernel->fileSystem->Open(fileName);
98      NoffHeader noffH;
99      unsigned int size;
100
101     if (executable == NULL) {
102     cerr << "Unable to open file " << fileName << "\n";
103     return FALSE;
104     }
105     executable->ReadAt((char *)&noffH, sizeof(noffH), 0);
106     if ((noffH.noffMagic != NOFFMAGIC) &&
107         (WordToHost(noffH.noffMagic) == NOFFMAGIC))
108         SwapHeader(&noffH);
109     ASSERT(noffH.noffMagic == NOFFMAGIC);
110
111 // how big is address space?
112     size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
113             + UserStackSize;    // we need to increase the size
114                         // to leave room for the stack
115     numPages = divRoundUp(size, PageSize);
116 //  cout << "number of pages of " << fileName<< " is "<<numPages<<endl;
117     size = numPages * PageSize;
118
119     ASSERT(numPages <= NumPhysPages);       // check we're not trying
120                         // to run anything too big --
121                         // at least until we have
122                         // virtual memory
123    /*** new line to construct */
124     pageTable = new TranslationEntry[numPages];
125     for (unsigned int i = 0, j = 0; i < numPages; i++) {
126         pageTable[i].virtualPage = i;  // for now, virt page # = phys page #
127       while(true){
128         if(usedPhysPages[j] == true)
129             j++;
130         else break;
131       }
132       pageTable[i].physicalPage = j;
133       usedPhysPages[j] = true;
134    //   pageTable[i].physicalPage = 0;
135       pageTable[i].valid = TRUE;
136    //   pageTable[i].valid = FALSE;
137       pageTable[i].use = FALSE;
138       pageTable[i].dirty = FALSE;
139       pageTable[i].readOnly = FALSE;
140     }
141    /*********************************/
142
143     DEBUG(dbgAddr, "Initializing address space: " << numPages << ", " << size);
144
```

之後將指令複製到 memory 上，physicalpage 的 address 要用 pageTable 來換算。使用 linear time 去搜尋第一個未使用的 page 進行填入。載入確定後，便要開始執行，

執行時，要去算程序進入點，進入點的位置即是 main memory address。

1. 算出第幾頁，乘上 PageSize 就是 page base
2. page offset = code.address%PageSize
3. page base + page offset 就是所需要的程序進入點。

addrspace.cc

```
    DEBUG(dbgAddr, "Initializing address space: " << numPages << ", " << size);

// then, copy in the code and data segments into memory
    if (noffH.code.size > 0) {
        DEBUG(dbgAddr, "Initializing code segment.");
        DEBUG(dbgAddr, noffH.code.virtualAddr << ", " << noffH.code.size);
        unsigned int vAddr = noffH.code.virtualAddr;
        unsigned int pAddr = pageTable[vAddr/PageSize].physicalPage * PageSize
                           + vAddr%PageSize;

            executable->ReadAt(
          &(kernel->machine->mainMemory[pAddr]),
            noffH.code.size, noffH.code.inFileAddr);
        }

    delete executable;          // close file
    return TRUE;            // success
}
```

最後在 process 執行完後，要把記憶體歸回給 CPU，釋放資源。

把標記為使用中的 page 設回未使用

addrspace.cc

```
78   AddrSpace::~AddrSpace()
79   {
80       for(unsigned int i=0; i<NumPhysPages; i++) {
81           usedPhysPages[pageTable[i].physicalPage] = FALSE;
82       }
83       delete[] pageTable;
84
85   }
```

III. Result

- Experiment result and some discussion
- Extra effort or observation

做完以上動作之後，執行./nachos -e ../test/test1 -e ../test/test2 的結果如下：

```
r08921005@r08921005-VirtualBox:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:6
return value:0
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 300, idle 8, system 70, user 222
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

最終執行結果

程式正確的執行了，並且是用 Round-Robin(Nachos 預設的 scheduling)的方式輪流執行。但這樣的程式會產生一個問題。當執行多個 thread 時，會因為 physical page 的數量不夠而導致程式出現 segmentation fault。如下面結果，因此在這邊應該隨著執行程式的 thread 數量增加 physical page 的數目，以維持足夠的數量供給 thread 使用

執行./nachos -e ../test/test1 -e ../test/test2 -e ../test/test1 的結果如下

```
r08921005@r08921005-VirtualBox:~/nachos-4.0/code/userprog$ ./nachos -e ../test/t
est1 -e ../test/test2 -e ../test/test1
Total threads number is 3
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Thread ../test/test1 is executing.
Print integer:9
Print integer:8
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Unexpected user mode exception4
Assertion failed: line 91 file ../userprog/exception.cc
Aborted (core dumped)
```