# 作業系統 - 作業一

**系級：電機所碩一　姓名：楊冠彥　學號：R11921091**

tags: `Ubuntu 14.04`, `Operating system`

## Pre(寫作業前)

我們需要安裝Linux 及 VM虛擬機軟體方便我們寫作業。 我安裝的OS Version為 Ubuntu 14.04 32bit 我安裝好的VMWare 虛擬機器檔案可從此處下載：https://drive.google.com/file/d/1Bvmuxz-eQIfNsLtdw-WxDS8s2k4xMPay/view?usp=sharing

裡面為Ubuntu 14.04 32bit 的作業系統並安裝所有Nachos hw1 需要的工具與程式碼包。 另安裝我習慣使用的Sublime text 3及相關套件，詳細套件清單請看：https://drive.google.com/file/d/1LWH4s6s0d565_uRSxnOOWbWx9yM2gS89/view?usp=sharing

## 分析問題：為什麼結果與預期不一致？ (Why the result is not congruent with expected?)

先來看看尚未做更改的結果。

`Output`:

```
u14@ubuntu:~/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
```

看到這個問題我首先思考導致這樣的原因，我的第一反應是回頭看test1.c和test2.c的程式碼。

*test1.c*

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

```
            PrintInt(n);
    }
```

*test2.c*

```
#include "syscall.h"

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

可發現原本test2的輸出是遞增，但在載入test1之後，test2的輸出卻變成了遞減，就像是test1被覆蓋過去，所以第一個想法是是否有context switching，但因為作業的投影片有寫，本次是做Thread Management，所以看到這樣的呈現結果，我猜測是因為虛擬記憶體沒有被多個程式管理。當多個程式同時執行時，程式可能都分配到同一塊虛擬記憶體，因此有部分重疊，導致執行緒同時進行程式執行時會互相干擾，程式互相重疊使用physicalPage。另外，助教有提供提示，讓我們去檢查三個檔案，因此看過程式碼後，確認我的思考應該是沒問題的。(但其實code/userprog/userkernel.cc 我認為不需要更改，只是可透過它看thread是怎麼跑)

# 解決方式

## 推測的解決方案

在程式的虛擬記憶體和未使用的記憶體間進行映射(mapping)，並為程式分配實體地址。 可能需更改的檔案 code/userprog/addrspace.cc code/userprog/addrspace.h

## 全域紀錄實體page使用

首先我們需要一個全域紀錄實體記憶體使用的陣列。

*code/userprog/addrspace.h*

### Code

```
class AddrSpace {
  public:
    ...
    static bool usedPhyPage[NumPhysPages];

  private:
    ...
};
```

## 宣告實體Page

接著，在 addrspace.cc 宣告實體，且初始化為 false，表示全部都還沒有用過。

### Code

```
bool AddrSpace::usedPhyPages[NumPhysPages] = {false};
```

## 刪除不必要的映射

將多餘做映射的code註解掉。

### Code

*code/userprog/addrspace.cc*

```
AddrSpace::AddrSpace()
{

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

## 釋放記憶體空間

我在page table想法就是從頭開始找physical address，有能用的就先佔下來，不能用就跳過。

### Code

*code/userprog/addrspace.cc*

```
AddrSpace::~AddrSpace()
{

    // free memory space
    for(int i = 0; i < numPages; ++i) {
        usedPhyPages[pageTable[i].physicalPage] = FALSE;
    }

    delete pageTable;
}
```

## 分配空的Pages

逐Page檢查以找到第一個未使用的Physical Page，並分配適當大小的Phisical Page。

### Code

*code/userprog/addrspace.cc*

```


    pageTable = new TranslationEntry[numPages];
    for (unsigned int i = 0, j = 0; i < numPages; i++) {

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

```
        while(usedPhyPages[j++] != FALSE)  // find first not-used phys. page
            { /* empty */ }
        usedPhyPages[j-1] = TRUE;
        pageTable[i].physicalPage = j - 1;

        //  pageTable[i].physicalPage = 0;
        pageTable[i].valid = TRUE;
        //  pageTable[i].valid = FALSE;
        pageTable[i].use = FALSE;
        pageTable[i].dirty = FALSE;
        pageTable[i].readOnly = FALSE;
    }

    size = numPages * PageSize;

    ASSERT(numPages <= NumPhysPages);       // check we're not trying
                        // to run anything too big --
                        // at least until we have
                        // virtual memory

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

要找出映射後的實體記憶體位置，所以我抓noffH.code.virtualAddr / PageSize 求得是第幾個 page，再索引 pageTable 找到對應的Physical Page是第幾頁，接著乘上每個 page 的大小得到該Physical Page的Physical memory，這時我們已經知道在第幾個Physical Page了，但卻不知道在這一Page的哪個位置，所以我們拿本來的 virtualAddr mod PageSize 求得在 page 內的offset(偏移)，加上剛剛拿到的Physical Page，就是對應的實體位址。

## Code

*code/userprog/addrspace.cc*

```
    // 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);

        // find the physical address (= physical page number + offset)
        physicalAddr = pageTable[noffH.code.virtualAddr/PageSize].physicalPage * PageSize
            + noffH.code.virtualAddr % PageSize;

        executable->ReadAt(
                &(kernel->machine->mainMemory[physicalAddr]),
                noffH.code.size, noffH.code.inFileAddr);
    }
    if (noffH.initData.size > 0) {
        DEBUG(dbgAddr, "Initializing data segment.");
        DEBUG(dbgAddr, noffH.initData.virtualAddr << ", " << noffH.initData.size);

        // find the physical address (= physical page number + offset)
        physicalAddr = pageTable[noffH.initData.virtualAddr/PageSize].physicalPage * PageSize
            + noffH.initData.virtualAddr % PageSize;

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

    delete executable;          // close file
```

```
    return TRUE;            // success
}
```

## 測試結果

可喜可賀！測試結果達成預期。

```
u14@ubuntu:~/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
```

## The difficulties I encountered

1. 幫三位同學debug，其中一位少加offset，另兩位都是變數不一致。
2. 幫一位同學解決Virtual box無法雙向複製貼上。
3. 幫兩位同學解決Ubuntu沒有sudoer權限之問題。
4. 向多位同學解釋作業欲達成之預期結果(輸出)為何？ 蠻多人在VM部分有問題的，所以花了有點多時間在幫其他人解決問題VM上的問題，且多數都是裝環境及VM軟體的問題，所以才會有最上面的VMWare 虛擬機器檔，供實在不會設定虛擬機環境之同學使用。