

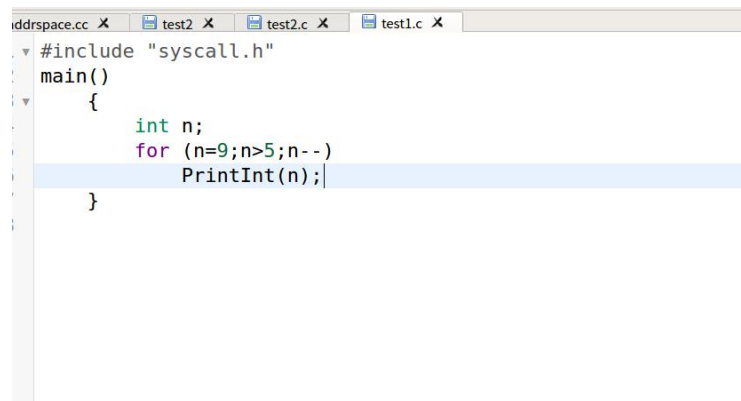
Nachos_hw1

T08902109 賈成鎔 資工

1. Motivation

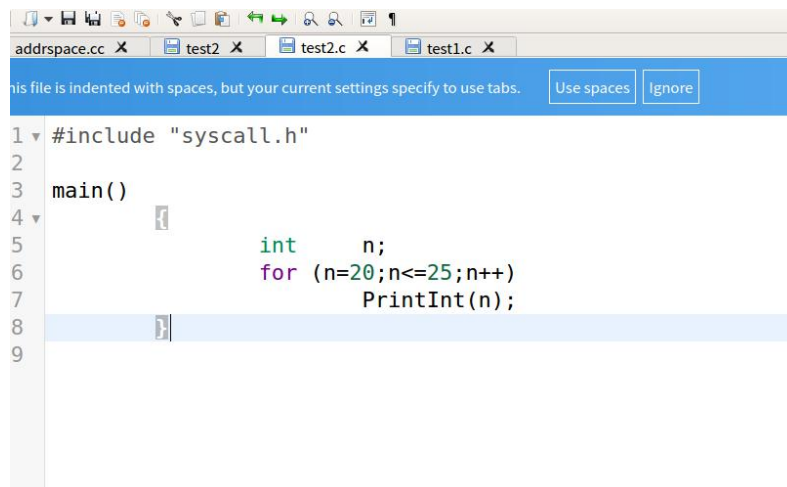
觀察 test1 和 test2 的代碼:

test1.c:



```
addrspace.cc x test2 x test2.c x test1.c x
#include "syscall.h"
main()
{
    int n;
    for (n=9;n>5;n--)
        PrintInt(n);
}
```

test2.c:



```
addrspace.cc x test2 x test2.c x test1.c x
this file is indented with spaces, but your current settings specify to use tabs. Use spaces Ignore
1 #include "syscall.h"
2
3 main()
4 {
5     int n;
6     for (n=20;n<=25;n++)
7         PrintInt(n);
8 }
9
```

test1 程序應該在 9-6 遞減輸出，而 test2 程序應該會在 20-25 遞增輸出。而實際運行結果在應該結束時遞增，結果如下：

```
jiachengyou@jiachengyou-virtual-machine: ~/nachos-4.0/code/userprog
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
```

問題分析：

為什麼會出現後面的遞增？

應該操作到了兩個程式相同區域的代碼段。並且發現輸出 `n` 的結果錯誤，可以想到應該是 `context-switch` 上發生問題。

解決方案：

為地址做標記，每一個 `process` 都會有一個 `AddrSpace`，紀錄 `logical address` 所對應在 `physical address`，也就是 `pageTable` 對應到 `pageTable[i].physicalPage`。

在作業一開始給的代碼中，所有程序的 `physicalPage` 都共用，如此一來當然會執行到同一份 `code`。

改進後當要執行某一頁的程序，則將會去找 `pageTable` 所對應的 `physicalPage`，然後去運行。

2. Implementation

(1) 做全局標記

```
14 #define ADDRSPACE_H
15
16 #include "copyright.h"
17 #include "fileys.h"
18 #include <string.h>
19
20 #define UserStackSize      1024    // increase this as necessary
21
22 class AddrSpace {
23 public:
24     AddrSpace();           // Create an address space.
25     ~AddrSpace();          // De-allocate an address space
26
27     void Execute(char *fileName); // Run the the program
28                                   // stored in the file "executable"
29
30     void SaveState();       // Save/restore address space-spec
31     void RestoreState();    // info on a context switch
32     //add
33     static bool usedPhyPage[NumPhysPages];
34 private:
35     TranslationEntry *pageTable; // Assume linear page table t
36                                   // for now!
37     unsigned int numPages;       // Number of pages in the virtual
38                                   // address space
39
40     void Load(char *fileName);    // Load the program data into
```

(2) 初始化為 0。同時當 process 執行成功後，把標記使用的 page 設回未使用

```
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 i
27 // object file header, in case the file was generated on a
28 // endian machine, and we're now running on a big endian ma
29 //-----
30 //add
31 bool AddrSpace::usedPhyPage[NumPhysPages] = {0};
32 static void
33
34 //-----
35 // AddrSpace::~AddrSpace
36 // Deallocate an address space.
37 //-----
38
39 AddrSpace::~AddrSpace()
40 {
41     for(int i = 0; i < numPages; i++)
42         AddrSpace::usedPhyPage[pageTable[i].physicalPage] = false;
43     delete pageTable;
44 }
45
46 //-----
47 // AddrSpace::Load
48 // Load a user program into memory from a file.
49 //
50 // Assumes that the page table has been initialized, and that
```

(2) 當將程序載入 momory 時，要去填入 pageTable[] 對應的 physicalPage，搜索第一

個未使用的 page 進行填入。

```
// how big is address space?
size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
      + UserStackSize;    // we need to increase the size
                          // to leave room for the stack
numPages = divRoundUp(size, PageSize);
// cout << "number of pages of " << fileName << " is " << numPages << endl;
// add
pageTable = new TranslationEntry[numPages];
for(unsigned int i = 0, j = 0; i < numPages; i++) {
    pageTable[i].virtualPage = i;
    while(j < NumPhysPages && AddrSpace::usedPhyPage[j] == true)
        j++;
    AddrSpace::usedPhyPage[j] = true;
    pageTable[i].physicalPage = j;
    pageTable[i].valid = true;
    pageTable[i].use = false;
    pageTable[i].dirty = false;
    pageTable[i].readOnly = false;
}
// end add
size = numPages * PageSize;

ASSERT(numPages <= NumPhysPages);    // check we're not trying
                                      // to run anything too big --
```

(3) 載入確定後，開始執行，去算程序進入點，進入點的位置即是記憶體地址。首先算出第幾頁，然後乘上 PageSize 就是 page base，而 page offset 則是 $\text{code.address \% PageSize}$ 。page base + page offset 就是我們所需要的程序進入點。

```
142 // then, copy in the code and data segments into memory
143 if (noffH.code.size > 0) {
144     DEBUG(dbgAddr, "Initializing code segment.");
145     DEBUG(dbgAddr, noffH.code.virtualAddr << ", " << noffH.code.size);
146
147 //add
148     executable->ReadAt(
149         &(kernel->machine->mainMemory[pageTable[noffH.code.virtualAddr/PageSize].physicalPage * PageSize + (noffH.code.virtualAddr
150             noffH.code.size, noffH.code.inFileAddr);
151 // end add
152 }
153 if (noffH.initData.size > 0) {
154     DEBUG(dbgAddr, "Initializing data segment.");
155     DEBUG(dbgAddr, noffH.initData.virtualAddr << ", " << noffH.initData.size);
156 // add
157     executable->ReadAt(
158         &(kernel->machine->mainMemory[pageTable[noffH.initData.virtualAddr/PageSize].physicalPage * PageSize + (noffH.code.virtual
159             noffH.initData.size, noffH.initData.inFileAddr);
160 // end add
161 }
162
163 delete executable;    // close file
164 return TRUE;          // success
165 }
```

3. Result

實驗結果：

```
This [x] [o] [d] jiachengyou@jiachengyou-virtual-machine: ~/nachos-4.0_original/code/userprog
136 make[1]: Leaving directory '/home/jiachengyou/nachos-4.0_ori
137 jiachengyou@jiachengyou-virtual-machine:~/nachos-4.0_origina
138 jiachengyou@jiachengyou-virtual-machine:~/nachos-4.0_origina
139 ../test/test1 -e ../test/test2
140 Total threads number is 2
141 Thread ../test/test1 is executing.
142 Thread ../test/test2 is executing.
143 Print integer:9
144 Print integer:8
145 Print integer:7
146 Print integer:20
147 Print integer:21
148 Print integer:22
149 Print integer:23
150 Print integer:24
151 Print integer:6
152 return value:0
153 Print integer:25
154 return value:0
155 No threads ready or runnable, and no pending interrupts.
156 Assuming the program completed.
157 Machine halting!
158
```

額外的觀察：

輸出的結果仍然穿插，詢問助教後，知道了這跟 scheduling 有關導致他們的輸出的確會穿插。

Reference: <https://morris821028.github.io/2014/05/24/lesson>