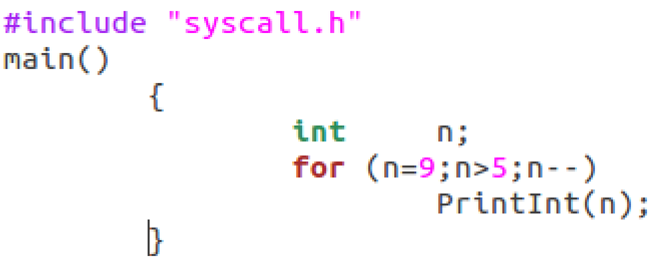
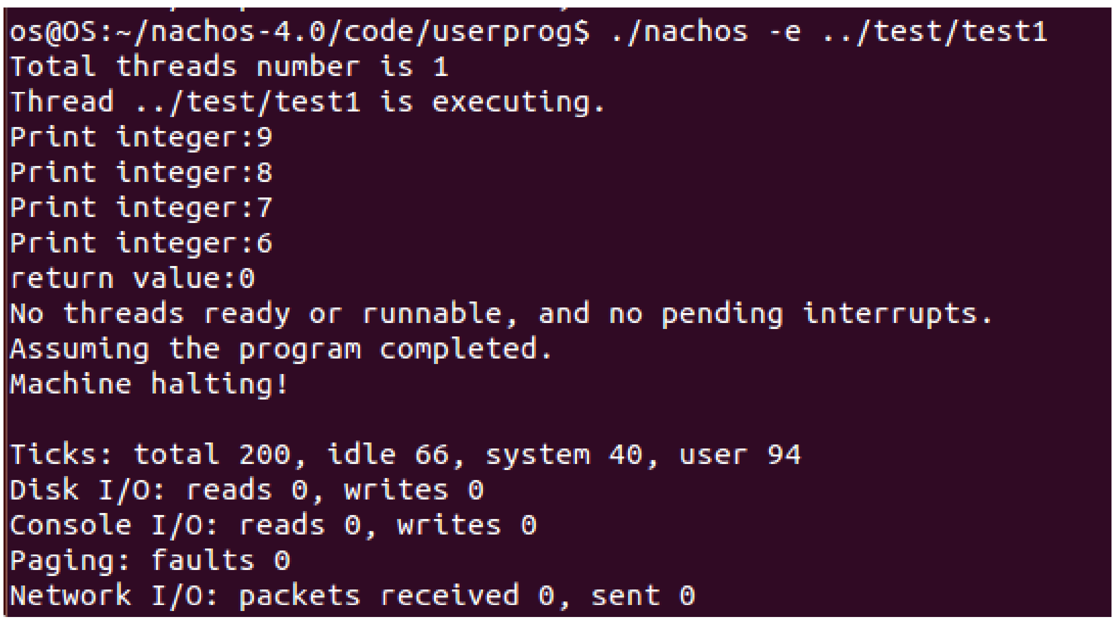
OS Project-1 Report

R11921008 羅恩至

test1 (source code and execution result)
執行結果為 9 ~ 6 依序輸出

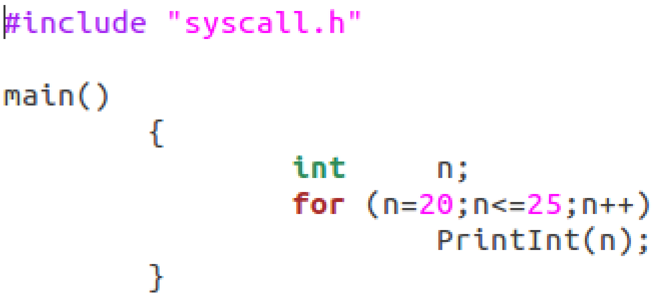


```c
#include "syscall.h"
main()
	{
		int	n;
		for (n=9;n>5;n--)
			PrintInt(n);
	}
```

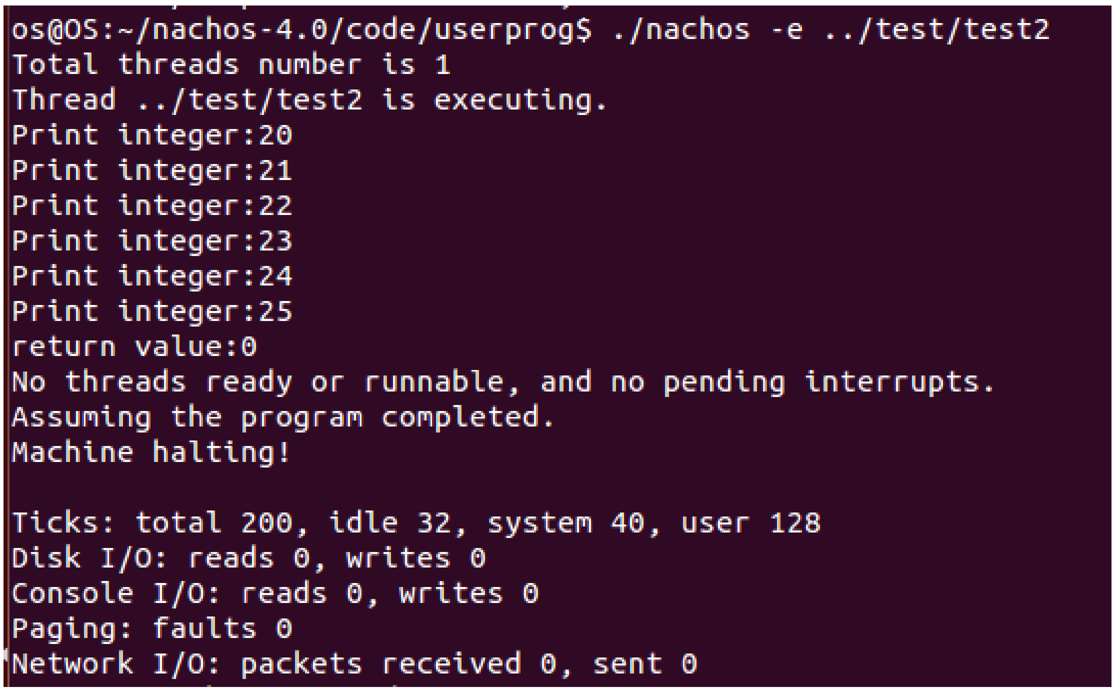


```
os@OS:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test1
Total threads number is 1
Thread ../test/test1 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:6
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 200, idle 66, system 40, user 94
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
```

test2 (source code and execution result)
執行結果為 20 ~ 25 依序輸出

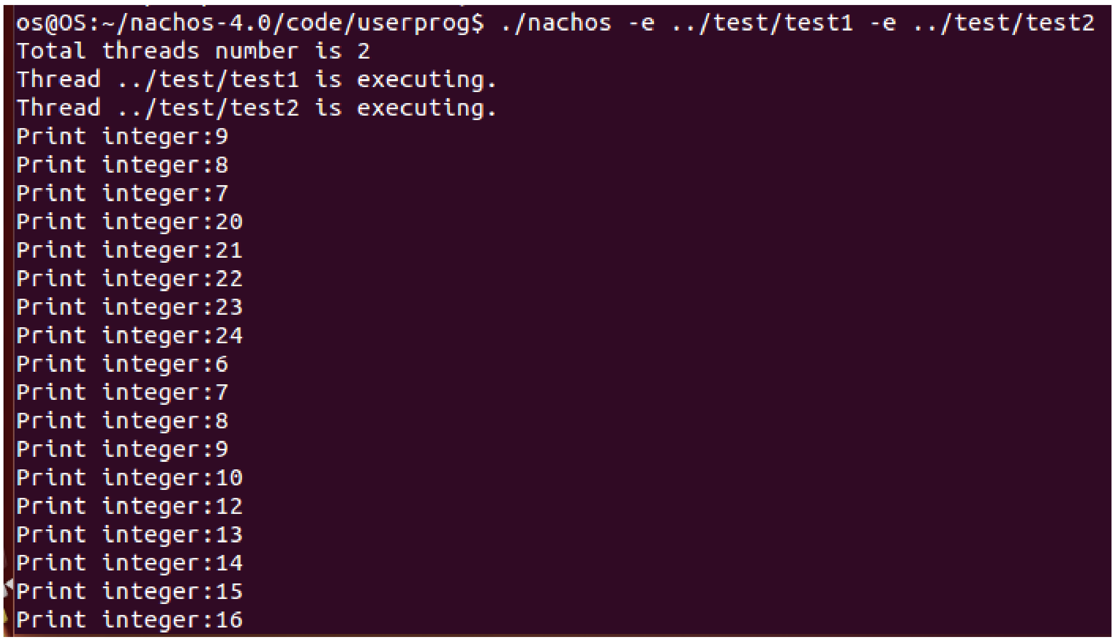


```c
#include "syscall.h"

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



```
os@OS:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test2
Total threads number is 1
Thread ../test/test2 is executing.
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 200, idle 32, system 40, user 128
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
```

But when executing test1 and test2 simultaneously, the result seems weird. The output is not consistent with source code.
兩程式並未按照原先方式輸出

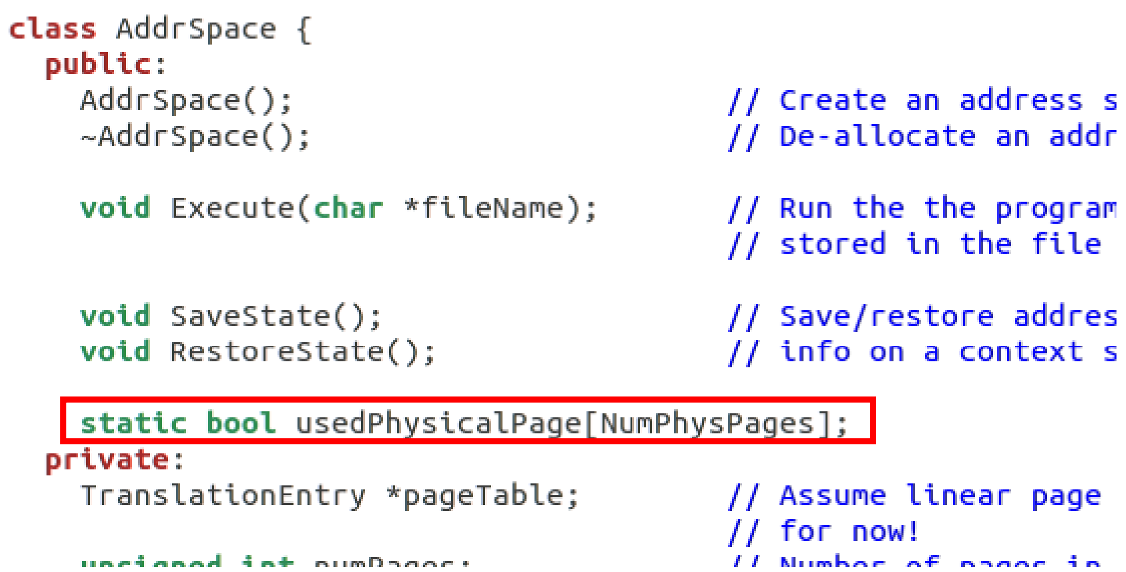


```
os@OS:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
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
```

1. 推測導致程式執行結果與預期不一致的原因：
Nachos 系統的預設並沒有對多個程式做記憶體管理，會將全部的實體記憶體分頁都分配出去，因此當多個執行緒在執行時，兩個程序會被分到同一個 page，導致執行時出現錯誤。

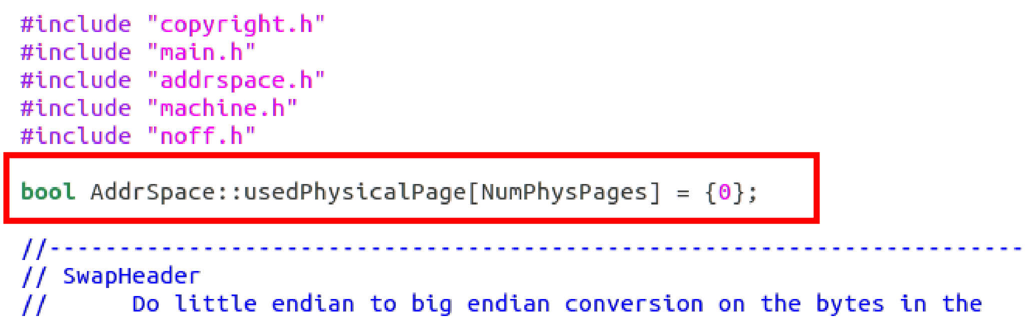
2. 解決方式
欲解決此問題，需讓程式的虛擬記憶體能 map 到實體記憶體上，詳細之操作過程如下所述：

(1) 在 addrspace.h 檔中，多增加一行變數
static bool usedPhysicalPage[NumPhysPages]，用來記錄 page 的使用情形，確認當下有哪些 page 是可以使用的。



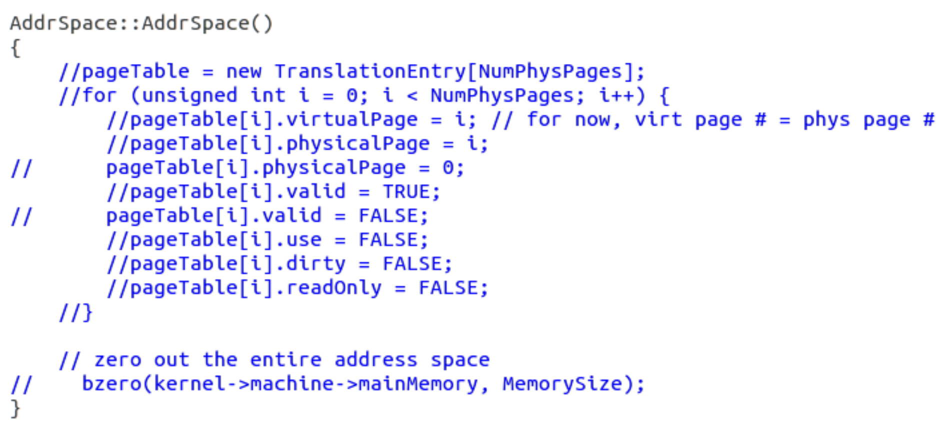
```
class AddrSpace {
  public:
    AddrSpace();                        // Create an address s
    ~AddrSpace();                       // De-allocate an addr

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

    void SaveState();                   // Save/restore addres
    void RestoreState();                // info on a context s

    static bool usedPhysicalPage[NumPhysPages];
  private:
    TranslationEntry *pageTable;        // Assume linear page
                                        // for now!
    unsigned int numPages;              // Number of pages in
```

並在 addrspace.cc 檔中初始化 usedPhysicalPage 參數：



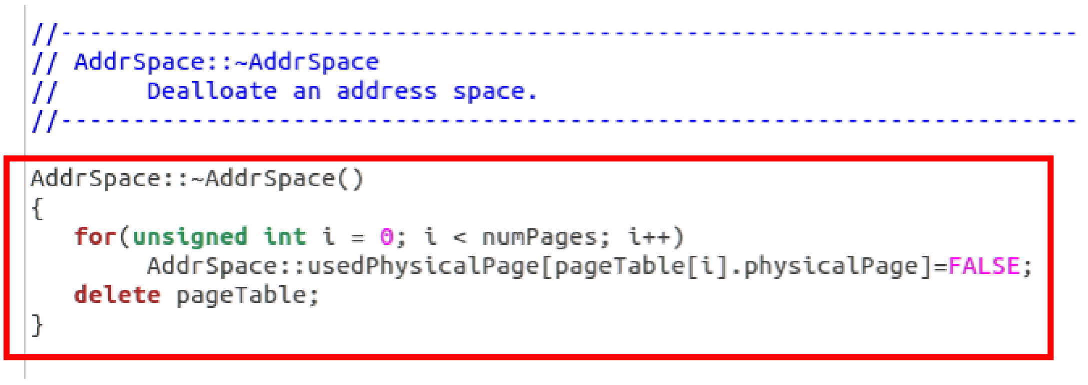
```
#include "copyright.h"
#include "main.h"
#include "addrspace.h"
#include "machine.h"
#include "noff.h"

bool AddrSpace::usedPhysicalPage[NumPhysPages] = {0};

//----------------------------------------------------------------
// SwapHeader
//      Do little endian to big endian conversion on the bytes in the
```

(2) 在 addrspace.cc 中的 AddrSpace::AddrSapce()函式，將函式中的所有程式做註解，因為進行多執行緒時，只需足夠大小的記憶體即可，無須分配實際記憶體的大小。



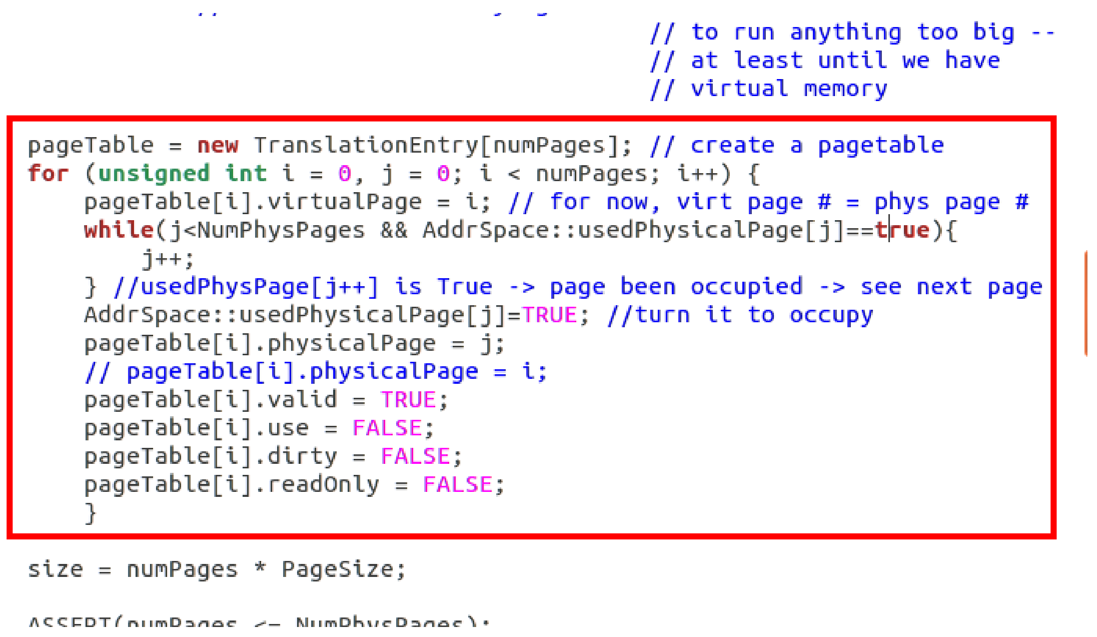
```
AddrSpace::AddrSpace()
{
    //pageTable = new TranslationEntry[NumPhysPages];
    //for (unsigned int i = 0; i < NumPhysPages; i++) {
        //pageTable[i].virtualPage = i; // for now, virt page # = phys page #
        //pageTable[i].physicalPage = i;
//      pageTable[i].physicalPage = 0;
   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

在解構子 AddrSpace::~AddrSapce()函式中，用 for 迴圈把所有有用到的 Page 都改回 false，釋放資源以供之後的程式使用。



```
//----------------------------------------------------------------------
// AddrSpace::~AddrSpace
//      Dealloate an address space.
//----------------------------------------------------------------------

AddrSpace::~AddrSpace()
{
   for(unsigned int i = 0; i < numPages; i++)
        AddrSpace::usedPhysicalPage[pageTable[i].physicalPage]=FALSE;
   delete pageTable;
}
```

(3) 在 AddrSpace::Load()函式中，把原本於 AddrSpace::AddrSapce()註解掉的程式移至此並稍作修改。透過 while 迴圈一一查看每個 physical page 是否已被使用，若已被使用(值為 true)則看下一頁，直到找到值為 false 的 page，將其設為 true 並將資料存至該頁。

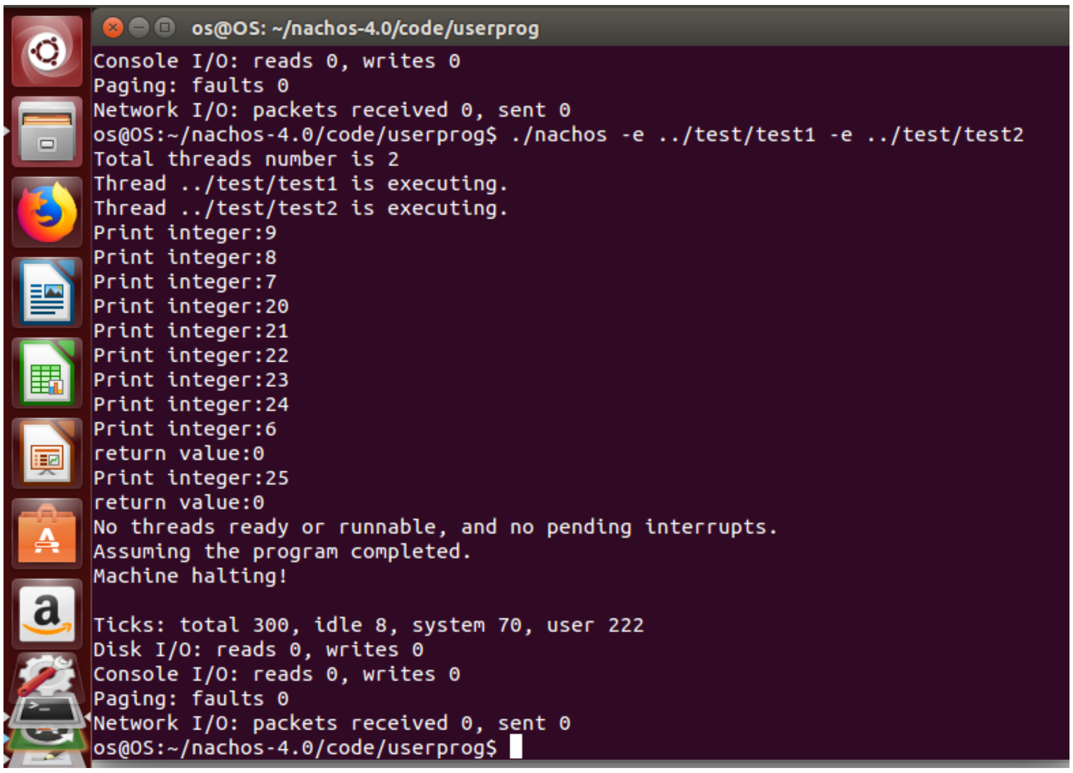


```
                                                // to run anything too big --
                                                // at least until we have
                                                // virtual memory

    pageTable = new TranslationEntry[numPages]; // create a pagetable
    for (unsigned int i = 0, j = 0; i < numPages; i++) {
        pageTable[i].virtualPage = i; // for now, virt page # = phys page #
        while(j<NumPhysPages && AddrSpace::usedPhysicalPage[j]==true){
            j++;
        } //usedPhysPage[j++] is True -> page been occupied -> see next page
        AddrSpace::usedPhysicalPage[j]=TRUE; //turn it to occupy
        pageTable[i].physicalPage = j;
        // pageTable[i].physicalPage = i;
        pageTable[i].valid = TRUE;
        pageTable[i].use = FALSE;
        pageTable[i].dirty = FALSE;
        pageTable[i].readOnly = FALSE;
        }

    size = numPages * PageSize;

    ASSERT(numPages <= NumPhysPages);
```

(4) 在 AddrSpace::Load()函式中，因為要找出 map 後的位置，因此更改程式碼中兩個 executable->ReadAt 的部分，使 virtualAddr 先除以 PageSize，以此得知使用的是第幾個 page，再透過 pageTable 找到所對應實體頁的頁數，並乘上每個 page 的大小得到該頁的 physical memory，接著再加上 virtualAddr 除以 PageSize 的餘數值，以得到 page 內的 offset，如此即得到相對應的 physical address。

```
    ASSERT(numPages <= NumPhysPages);
    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);
        executable->ReadAt( &(kernel->machine->mainMemory[pageTable
[noffH.code.virtualAddr/PageSize].physicalPage*PageSize+(noffH.code.virtualAddr%
PageSize)]), noffH.code.size, noffH.code.inFileAddr);
    }
        if (noffH.initData.size > 0) {
        DEBUG(dbgAddr, "Initializing data segment.");
        DEBUG(dbgAddr, noffH.initData.virtualAddr << ", " <<
noffH.initData.size);
        executable->ReadAt( &(kernel->machine->mainMemory[pageTable
[noffH.code.virtualAddr/PageSize].physicalPage*PageSize+(noffH.code.virtualAddr%
PageSize)]), noffH.code.size, noffH.code.inFileAddr);
    }

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

3. 修改之後，再同時執行 test1 與 test2，即可發現程式已能正常執行。



os@OS: ~/nachos-4.0/code/userprog

```
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
os@OS:~/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
os@OS:~/nachos-4.0/code/userprog$ 
```

4. 討論

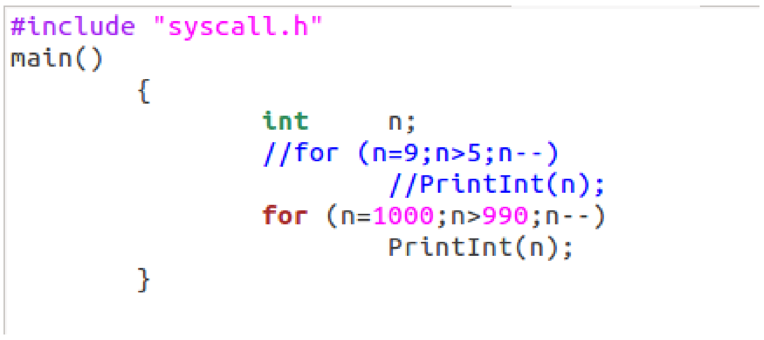
(1) 困難

在進行此作業時，因為對 nachos 系統尚未熟悉，因此花了一些時間在看資料夾中的 code 並理解其中的功用，而在嘗試做修改時，也曾因為更改錯誤以及更改其他檔案，導致在同時執行 test1 test2 時，呈現非目標輸出亦非原先所輸出的怪異結果，後來重新安裝 VM 並重新 trace code 之後才做出正確的結果。

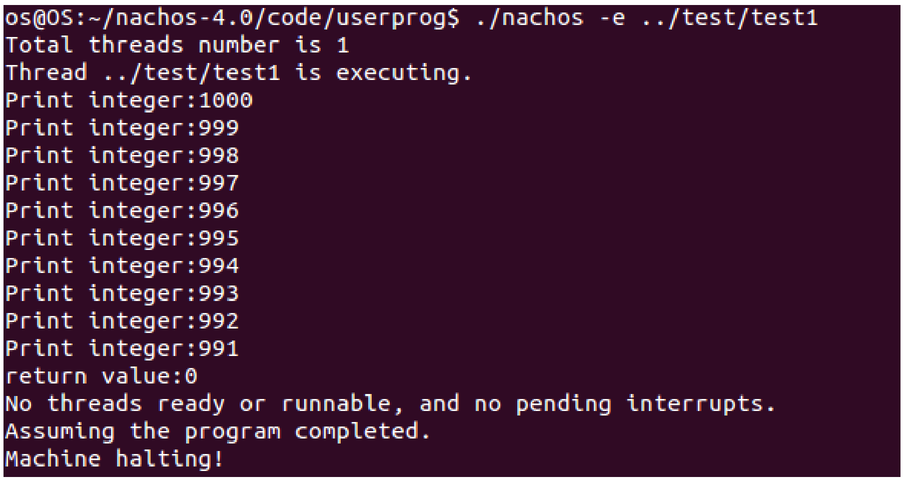
(2) 額外發現

嘗試修改 test1 test2 程式，增大數值的輸出範圍與兩程式間的數值差異

test 1：依序遞減輸出 1000~991

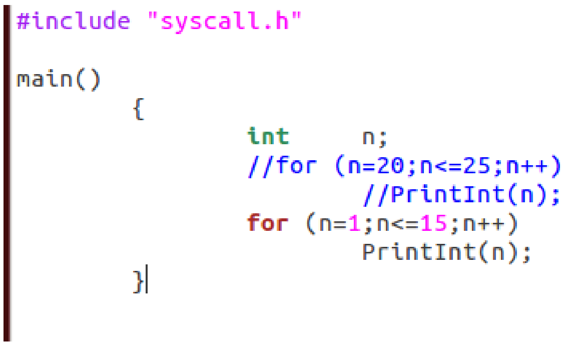


```
#include "syscall.h"
main()
	{
		int	n;
		//for (n=9;n>5;n--)
			//PrintInt(n);
		for (n=1000;n>990;n--)
			PrintInt(n);
	}
```

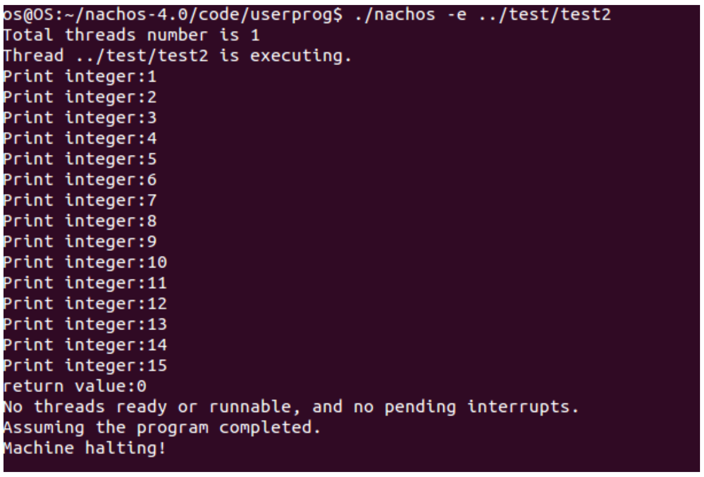


```
os@OS:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test1
Total threads number is 1
Thread ../test/test1 is executing.
Print integer:1000
Print integer:999
Print integer:998
Print integer:997
Print integer:996
Print integer:995
Print integer:994
Print integer:993
Print integer:992
Print integer:991
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

test 2：依序遞增輸出 1~15

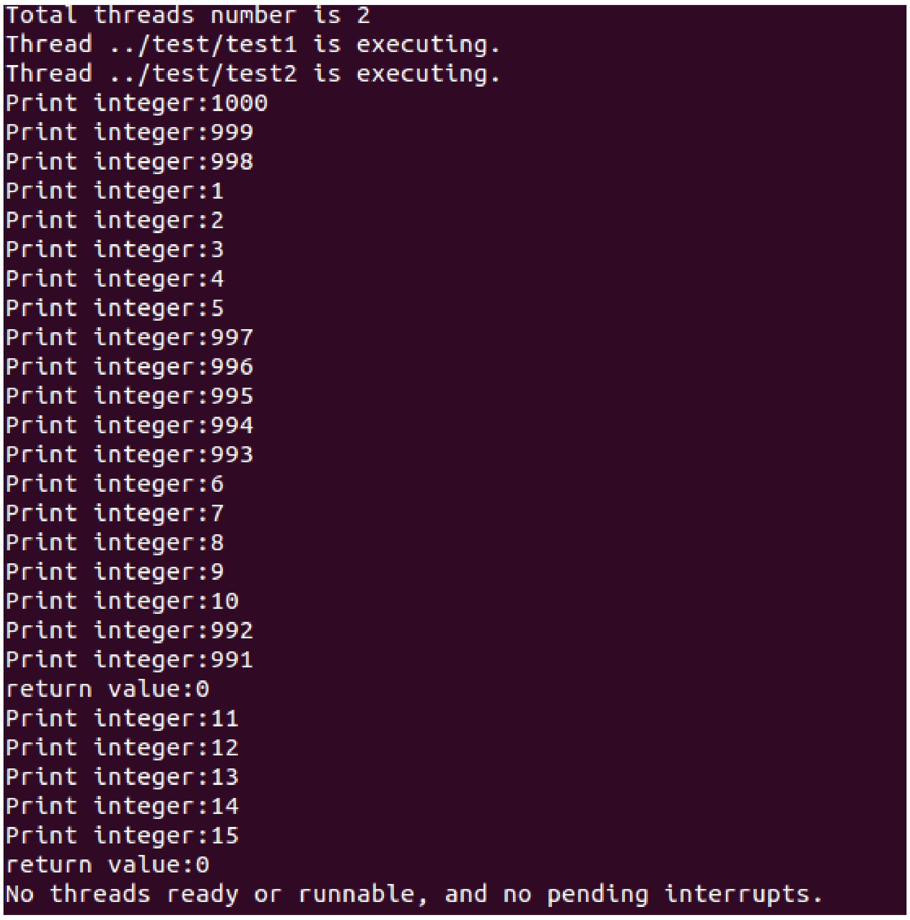


```
#include "syscall.h"

main()
	{
		int	n;
		//for (n=20;n<=25;n++)
			//PrintInt(n);
		for (n=1;n<=15;n++)
			PrintInt(n);
	}
```



```
os@OS:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test2
Total threads number is 1
Thread ../test/test2 is executing.
Print integer:1
Print integer:2
Print integer:3
Print integer:4
Print integer:5
Print integer:6
Print integer:7
Print integer:8
Print integer:9
Print integer:10
Print integer:11
Print integer:12
Print integer:13
Print integer:14
Print integer:15
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

執行結果如下：



```
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:1000
Print integer:999
Print integer:998
Print integer:1
Print integer:2
Print integer:3
Print integer:4
Print integer:5
Print integer:997
Print integer:996
Print integer:995
Print integer:994
Print integer:993
Print integer:6
Print integer:7
Print integer:8
Print integer:9
Print integer:10
Print integer:992
Print integer:991
return value:0
Print integer:11
Print integer:12
Print integer:13
Print integer:14
Print integer:15
return value:0
No threads ready or runnable, and no pending interrupts.
```

從上圖可看到，除了兩個執行緒的程式均正常執行外，也可發現 test1 和 test2 兩執行緒之間是以循環分配的方式進行：其中一個程式先執行一部分後，會執行另外一個程式，兩者相互交替執行至程式結束。

5. Reference

(1) https://wiiwu959.github.io/2019/10/10/2019-10-10-OS_HW1-2019/