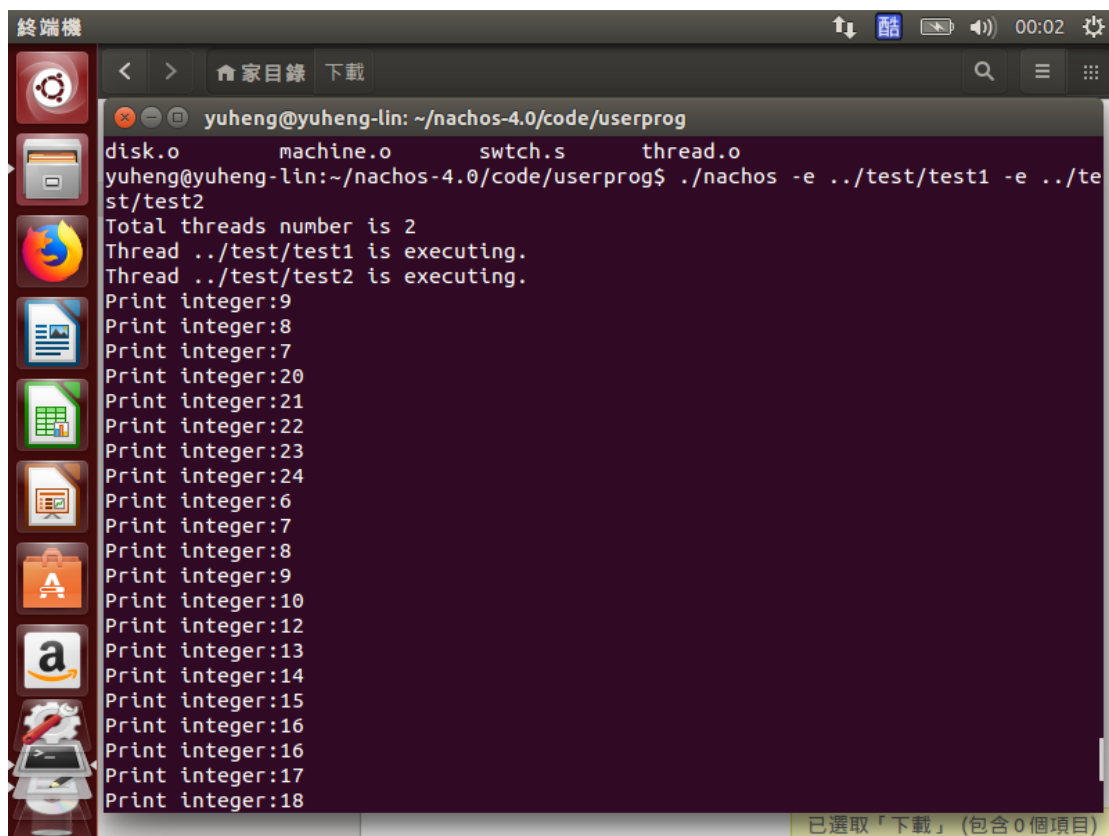


作業系統 Report

作業一 R10945061 林宇恆

Why the result is not congruent with the expected?

下圖為直接 make 後的結果，正常來說單獨測試 test1 和 test2 後得到結果應為一個遞增以及一個遞減，對應到了 test file 中的 test1.c 以及 test2.c，但同時測試 test1 以及 test2 得到以下之結果，兩者都最後都遞增輸出，原因應該不是 content switch 的問題，因為分別輸出的結果甚至一起輸出後數字都是測試檔規範的數字，而原因應該是兩個測試檔沒有儲存在規劃好的 memory page，由於 nachos 設計沒有 multithread 的記憶體配置的規劃，因此同時執行後會重疊到彼此的 memory page。

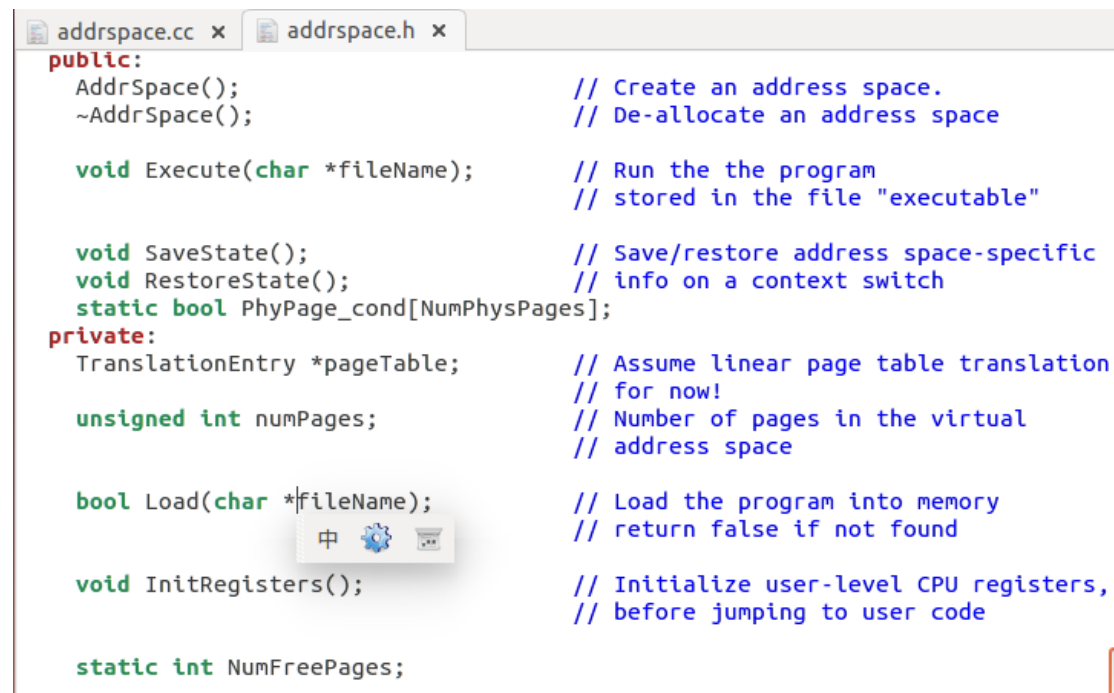


```
終端機
yuheng@yuheng-lin: ~/nachos-4.0/code/userprog
disk.o      machine.o      swtch.s      thread.o
yuheng@yuheng-lin:~/nachos-4.0/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
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
```

How to solve the issue?

因此我們要做的就是標記使用過的 address，每一個 process 的都有 address，因此我們把有沒有紀錄過的 binary address（有使用過或沒使用過）可以對應到實際 address，也就是 pageTable 對應 pageTable[i].physicalPage

首先建構一個和實體一樣大的 `pageTable`，下圖為在 `addrspace.h` 中宣告一個 `binary` 的 `pageTable`，我稱之為 `PhyPage_cond`，而大小為 `NumPhysPages`，相當於實體的 `memory` 大小，使用 `static` 可以讓 `class` 中的成員共同享有 `pageTable` 的記錄狀況，另外定義 `NumFreePages` 用來得知目前還有多少 `page` 可以使用。



```
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
    static bool PhyPage_cond[NumPhysPages];
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 int NumFreePages;
```

下圖為在 `addrspace.cc` 中前兩個變數的定義，`PhyPage_cond` 即初始化把所有的空間設為零，而 `NumFreePages` 初始化為完全都還沒使用過，因此會等於 `NumPhysPages`，一個奇怪的點是我當初在定義是直接將 `PhyPage_cond` 的值設為 `{0}`，照理來說應該得到相同的結果，但是在 `Make` 時卻出現 `error`，也因此改定義兩個 `variable` 作為 `true` and `false`。

```
#define PAGE_OCCU true
#define PAGE_NONE false

bool AddrSpace::PhyPage_cond[NumPhysPages] = {PAGE_NONE};
int AddrSpace::NumFreePages = NumPhysPages;
```

在 `Load function` 中分配記憶體的配置，使用 `for loop` 來一個一個找，`index` 為 `page` 的標號，若使用過(`while` 中描述)則跳過他，反之若看到沒有使用過的 `page` 就將它拿來用，使用前 `NumFreePage` 要減一代表可以使用的 `page` 少一個，另外使用前使用它內建的清空函數 `bzero` 來初始化想要使用的 `Page`，最後就是對 `page` 進行標記。另外要記得更改讀取到 `page` 的 `address`。

```

pageTable = new TranslationEntry[numPages];
for(unsigned int i = 0, index = 0; i < numPages; i++) {
    pageTable[i].virtualPage = i;
    while(index < NumPhysPages && PhyPage_cond[index] == PAGE_OCCU) index++;
    PhyPage_cond[index] = PAGE_OCCU;
    NumFreePages--;
    //清空即將分配的 page
    bzero(&kernel->machine->mainMemory[index * PageSize], PageSize);
    pageTable[i].physicalPage = index;
    pageTable[i].valid = true;
    pageTable[i].use = false;
    pageTable[i].dirty = false;
    pageTable[i].readOnly = false;
}

```

最後在 destructors(解構子)的部分需要 free 掉先前使用的 page，往後在使用程式才比較不會出現問題。

```

AddrSpace::~AddrSpace()
{
    for(int i = 0; i < numPages; i++){
        PhyPage_cond[pageTable[i].physicalPage] = PAGE_NONE;
        NumFreePages++;
    }
    delete pageTable;
}

```

Experiment result

修正後測試的結果如下圖，可以看到兩個測試檔都能夠按照自己 for loop 中描述的一個遞增，一個遞減，total thread 為 2 也顯示這是 multithread 的過程。

```

yuheng@yuheng:~$ ./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

```

Discussion

此次作業需要對電腦中資料存放的位置以及 **memory** 的使用有一定的了解，因此起初在了解題目這塊花了不少的時間，也是透過網路上有關於 **nachos** 的資料來更了解整個作業的目的以及實際上在模擬電腦的功能，經過此次作業後對 **multithread** 這個部分有更多的了解，除了在分配 **address** 上需要經過設計才不會讓兩個程式互相打架，也要記得在分配後要記得 **free** 要求的資源，才不會讓電腦產生 **stack overflow**。