HW4 – Virtual Memory

分工：

B10615045 陳尚富: 主要撰寫 HW4

B10601019 鞠傳豐: 二退

B10601033 陳彥憑: 參與討論。

想法和部分重要 code：

Requirement1:

因為要實現 page fault 的功能，所以先去查看 translate 的內容，發現到只要發生錯誤，就會 RaiseException，然後 OneInstruction 就會重新執行發生錯誤的 PC。經過思考後，在上一次作業的基礎上做更動，設計 page fault 的執行流。一個 thread 在被創建後，或是 physical memory 還有 frame 可以使用就將其直接放到 memory 中，若是 physical memory 已經滿了，就將 data 存到 backing store 中。等到需要被引用時，就會發生 page fault。

這邊會想要這麼做，而不做 pure demand paging 的原因，是因為我們在 nachos 會執行的程式大多 page 很少，而且大多執行的程式會馬上被執行。若是我做 pure demand paging，先將讀檔出來 data 通通存到 backing store，再通過需要使用時產生 page fault 來逐漸塞滿 physical memory，那麼我的 page fault 數量會變多，也就會有更多的 swap in, swap out，導致我用我目前實施的方法在目前的方式裡，可能會比 pure demand paging 來得好。

```
170 //@shungfu : Edit at Hw3
171     pageTable = new TranslationEntry[numPages];     // PageTable initial
172
173     // Deploy the virtual and physical page tables.
174     for(unsigned int i = 0; i < numPages; i++){
175
176         // Same operation of Enough/Not Enough phyical memory page
177         pageTable[i].virtualPage = i;
178         pageTable[i].use = FALSE;
179         pageTable[i].dirty = FALSE;
180         pageTable[i].readOnly = FALSE;
181
182         if(kernel->physPageTable->numFreePhyPages < 1){    // Not enough Physical page
183             int sec = 0;
184             while(kernel->vmDisk->SectorUsed[sec] == USED){  // SynchDisk sector have been Used
185                 sec++;
186             }
187
188             ASSERT(sec < NumSectors);   // check if setor number is in safe zone
189             kernel->vmDisk->SectorUsed[sec] = USED;  // set sector to USED
190
191             pageTable[i].physicalPage = sec;    // points to the disk sector
192             pageTable[i].valid = FALSE;
193
194             // Copy from Real disk to SynchDisk(Nachos), unit: page, ReadAt(char* into, int numBytes, int position)
195             char *buffer = new char [PageSize];
196             DEBUG(dbgHw3, "No Free Physiacal Memory Page, get free disk sector: " << sec);
197             executable->ReadAt( buffer, PageSize, noffH.code.inFileAddr + (i*PageSize) );
198             kernel->vmDisk->WriteSector(sec, buffer);
199         }
```

```
200         else {  // Remaining Pages
201             int ppn = 0;    // physical page number#
202 
203             // Deploy on only available index
204             while(ppn < NumPhysPages && kernel->physPageTable->used[ppn] == USED){
205                 ppn++;
206             }
207 
208 
209             pageTable[i].physicalPage = ppn;
210             pageTable[i].valid = TRUE;
211             kernel->physPageTable->numFreePhyPages--;
212             kernel->physPageTable->used[ppn] = USED;
213             kernel->physPageTable->virtPage[ppn] = i;
214             if(kernel->memManageUnit->getPagingType() == FIFO){     // FIFO used
215                 kernel->physPageTable->load_time[ppn] = kernel->memManageUnit->loading_time;
216                 kernel->memManageUnit->loading_time++;
217                 DEBUG(dbgHw3, "Free Physical Memory page #" << ppn << ", loading time: " << kernel->memManageUnit->loading_time-1);
218             }
219             else{   // LRU used
220                 kernel->physPageTable->load_time[ppn] = kernel->memManageUnit->counter;
221                 kernel->memManageUnit->counter++;
222                 DEBUG(dbgHw3, "Free Physical Memory page #" << ppn << ", counter: " << kernel->memManageUnit->counter-1);
223             }
224 
225             // Copy from Real disk to Physical Memory(Nachos), unit: page
226             executable->ReadAt( &(kernel->machine->mainMemory[ppn * PageSize]), PageSize, noffH.code.inFileAddr + (i*PageSize) );
227         }
228         DEBUG(dbgHw3, "Put " << kernel->currentThread->getName() << "'s vpn " << i << " in.");
229     }
230 
```

在從 file 存入到 backing store 的方法，我是使用 Nachos 提供的 synchDisk 來達成。他是一個建立在 Disk 之上的 interface。對應到 backing store 的方式，我是利用將當前無法放進 memory 的 page 的 physical page 存放應該要放入的 sector。至於怎麼知道他是對應到 backing store 還是 memory，用 valid 來判定。而為了要判斷這個 sector 有沒有被使用到，我在 synchDisk 中添加 SectorUsed。

```
47 //@shungfu: Edit at Hw3
48     bool *SectorUsed;    // Record which sector have been used
```

在修改上述程式碼時，認為應該要另外寫一個 class 來存放 physical memory 相關的東西，會更方便簡潔，所以我創建了一個 PhysPageTable(.h/.cc)。

```
 1 // physPageTable.h
 2 // Data Structers to keep track of physical page table.
 3 //
 4 //@shungfu: Edit at Hw3
 5 
 6 #ifndef PHYSPAGETABLE_H
 7 #define PHYSPAGETABLE_H
 8 
 9 #include "copyright.h"
10 
11 #define NOT_USED false
12 #define USED true
13 
14 class PhysPageTable{
15     public:
16         PhysPageTable();
17         ~PhysPageTable();
18 
19         bool *used;    // Used to record the physical page table that have been used, default set not used
20         int *load_time;  // used to record load time at FIFO
21         int *virtPage; // back points to virtual Page#
22         int numFreePhyPages;    // Number of physical pages can be used
23         void CleanUp(int page); // Clean up when a userprog is returned
24 };
25 #endif
```

這邊直得留意的是，到時候再做 swap in, swap out 的時候，會發現 page table 對應到 physical memory 應該要是雙向的 entry 比較方便，所以我在這邊多加了 virtPage 來紀錄是哪一個#page 指到這個#frame。

```
22 class SynchDisk;
23 class UserProgKernel : public ThreadedKernel {
24   public:
25     UserProgKernel(int argc, char **argv);
26                 // Interpret command line arguments
27     ~UserProgKernel();      // deallocate the kernel
28
29     void Initialize();      // initialize the kernel
30
31     void Run();         // do kernel stuff
32
33     void SelfTest();        // test whether kernel is working
34
35 // These are public for notational convenience.
36     Machine *machine;
37     FileSystem *fileSystem;
38
39 //@shungfu: Edit at Hw3
40     MMU *memManageUnit; // MMU
41     PhysPageTable *physPageTable;
42     SynchDisk *vmDisk;  // Backing Store in Nachos.
43                         // not using synchDisk because not sure what will happen
44
45 #ifdef FILESYS
```

接著就處理 translate 錯誤導致的 Page Fault Exception。所以在 Exception Handler 中新增了 PageFaultException，來處理 page fault。這邊我先開了一個 Lock，為了是確保同時只會有一個 thread 進入去對 memory 和 backing store 做存取，避免可能有同時存取問題，導致發生一些難以預期的狀況。接著我就把 page fault 的統計數增加，然後將 Machine::RaiseException 中處理好的錯誤 virtual address 讀入，計算他的 virtual page number，然後送到我新增的 class MMU 來詳細處理 page fault。

```
99     // @shungfu: Edit ad Hw3
100    case PageFaultException:    // Page Fault happens
101
102        if (pageLock == NULL){  // make a semaphore lock if there is not
103            pageLock = new Lock("PageLock");    // only one process can do page replacement at on time
104        }
105        pageLock->Acquire();
106
107        kernel->stats->numPageFaults++; // Add statistic Number of Page Faults
108        int Error_VAddr;
109        unsigned int vpn;
110
111        Error_VAddr = kernel->machine->ReadRegister(BadVAddrReg);   // the virtual addr makes page fault.
112        vpn = (unsigned)Error_VAddr / PageSize;   // virtual page number
113
114        cout << "\n-------------------- Page Fault --------------------" << endl;
115        kernel->memManageUnit->pageFault(vpn);
116        cout << "----------------------------------------------------\n" << endl;
117        pageLock->Release();
118
119        return;
```

```
100 void
101 Machine::RaiseException(ExceptionType which, int badVAddr)
102 {
103     DEBUG(dbgMach, "Exception: " << exceptionNames[which]);
104
105     registers[BadVAddrReg] = badVAddr;
106     DelayedLoad(0, 0);          // finish anything in progress
107     kernel->interrupt->setStatus(SystemMode);
108 //  cout << "entering system mode...\n";
109     ExceptionHandler(which);        // interrupts are enabled at this point
110     kernel->interrupt->setStatus(UserMode);
111 //  cout << "entering user mode...\n";
112 }
```

```h
// MMU.h
// Memory Management Unit
// Used to handle the translation between virtual and physical page table.
//
//#shungfu: Edit at Hw3

#ifndef MMU_H
#define MMU_H

#include "copyright.h"

enum PagingType{
    FIFO,
    LRU,
};

class MMU{
    public:
        MMU(PagingType);
        ~MMU();

        void pageFault(unsigned int vpn);   // Will called by ExceptionHandler()
        int getVictim();    // get the victim page (now is just for FIFO and LRU)
        PagingType getPagingType(){return type;};
        void swap_out(char* buf, int sec, int virtPage);
        void swap_in(unsigned int vpn, int victim);
        int loading_time;   // used at FIFO, will increased while page fault happend and initial memory
        int counter;    // used at LRU, will increased while access the physical memory frame
    private:
        PagingType type;    // Algorithm choosed
};
#endif
```

Requirement2:

Exception Handler 執行後，會先進入 MMU 的 pagefault()。如果發生 page fault 時 physical memory 還有可用的 frame，那就將其直接 swap in 到可用的 physical memory frame。如果沒有可用的 frame 就要做 page replacement algorithm。流程：先用不同演算法找到 victim page，然後將其 swap out 到 backing store，然後把要存取的 virtual address swap in 到 physical memory。

```h
void
MMU::pageFault(unsigned int vpn){
//    DEBUG(dbgHw3, "MMU::pageFault vpn: " << vpn);
//    DEBUG(dbgHw3, "MMU::PageTable: " << &kernel->machine->pageTable[vpn]);

    int victim = 0;

    if(kernel->physPageTable->numFreePhyPages > 0){ // Still got free Physical page

        while(victim < NumPhysPages && kernel->physPageTable->used[victim] == USED){
            victim++;
        }
        cout << "Found physical page " << victim << " free to use." << endl;
        cout << "Swap in " << kernel->currentThread->getName() <<  "'s virtual page: " << vpn << endl;
        swap_in(vpn, victim);
    }
    else{
        if(type == FIFO || type == LRU){
            // get the smallest load time page as victim
            victim = getVictim();

            // copy victim's physical memory data into buf
            char *buf = new char [PageSize];
            bcopy(&kernel->machine->mainMemory[victim*PageSize], buf, PageSize);

            // Get disk sec where victim to go
            int sec = kernel->machine->pageTable[vpn].physicalPage;

            // Get victim's virtaul page
            int virtPage = kernel->physPageTable->virtPage[victim];

            cout << "Swap out physical page: " << victim << endl;
            cout << "Swap in " << kernel->currentThread->getName() << "'s virtual page: " << vpn << endl;

            // swap in
            swap_in(vpn, victim);

            // swap out
            swap_out(buf, sec, virtPage);

        }
    }
}
```

由於 FIFO 和 LRU 演算法，都是找最小的 load time，只是差在 FIFO 只有放新資料到 memory 時才會 load time++，LRU 只要有存取到 memory 就要 load time++，所以找 Victim 的方法一樣。

```
63 int
64 MMU::getVictim(){
65     int smallest = kernel->physPageTable->load_time[0];
66     int victim = 0;
67     for(int i=1;i <NumPhysPages;i++){
68         if(smallest > kernel->physPageTable->load_time[i]){
69             smallest = kernel->physPageTable->load_time[i];
70             victim = i;
71         }
72     }
73     return victim;
74 }
```

我是先 swap in 再 swap out，所以 swap out 需要用到且可能會被 swap in 改動的部分都要先複製一份起來。至於會先 swap in 在 swap out 的理由是因為，前面先處理了如果 physical memory frame 未滿的問題，所以這邊也先處理 swap in 比較方便。（複製了 physical memory 裡原本存有的資料、應該存到 backing tore 的哪個 sector、還有 victim 對應到的 virtual page number#）

```
76 void
77 MMU::swap_in(unsigned int vpn, int victim){
78     // Read disk data into buf
79     char *buf = new char[PageSize];
80     int sec = kernel->machine->pageTable[vpn].physicalPage;
81     kernel->vmDisk->ReadSector(sec, buf);
82
83     // save buf into physical memory
84     bcopy(buf, &kernel->machine->mainMemory[victim*PageSize], PageSize);
85     // Set disk sector as NOT USED
86     kernel->vmDisk->SectorUsed[sec] = NOT_USED;
87
88     // Update the pageTable information
89     kernel->machine->pageTable[vpn].physicalPage = victim;
90     kernel->machine->pageTable[vpn].valid = TRUE;
91
92     // Update Physical Page table information
93     kernel->physPageTable->numFreePhyPages--;
94     kernel->physPageTable->used[victim] = USED;
95     kernel->physPageTable->virtPage[victim] = vpn;
96     kernel->physPageTable->load_time[victim] = kernel->memManageUnit->loading_time;
97     kernel->memManageUnit->loading_time++;
98 }
99
100 void
101 MMU::swap_out(char* buf, int sec, int virtPage){
102     // Write into disk
103     kernel->vmDisk->WriteSector(sec,buf);
104
105     // Set disk sector as USED
106     kernel->vmDisk->SectorUsed[sec] = USED;
107
108     // Update the pageTable information
109     kernel->machine->pageTable[virtPage].physicalPage = sec;
110     kernel->machine->pageTable[virtPage].valid = FALSE;
111     kernel->machine->pageTable[virtPage].dirty = FALSE;
112
113     kernel->physPageTable->numFreePhyPages++;
114 }
```

最後，要處理 FIFO 和 LRU 的 load time 應該在哪裡++。但是為了方便 LRU 和 FIFO 計算 load time 的 counter 不一樣（loading_time 和 counter）。

```
int loading_time;    // used at FIFO, will increased while page fault happend and initial memory
int counter;     // used at LRU, will increased while access the physical memory frame
```

FIFO 只有放新資料到 memory 時才會 load time++，也就是剛開始 load address space 和 page fault 發生的時候。LRU 只要有存取到 memory 就要 load time++。

```
214             if(kernel->memManageUnit->getPagingType() == FIFO){     // FIFO used
215                 kernel->physPageTable->load_time[ppn] = kernel->memManageUnit->loading_time;
216                 kernel->memManageUnit->loading_time++;
217                 DEBUG(dbgHw3, "Free Physical Memory page #" << ppn << ", loading time: " << kernel->memManageUnit->loading_time-1);
218             }
219             else{   // LRU used
220                 kernel->physPageTable->load_time[ppn] = kernel->memManageUnit->counter;
221                 kernel->memManageUnit->counter++;
222                 DEBUG(dbgHw3, "Free Physical Memory page #" << ppn << ", counter: " << kernel->memManageUnit->counter-1);
223             }
```

AddrSpave::Load(), FIFO 和 LRU

```
248     // Do Real shit
249     entry->use = TRUE;      // set the use, dirty bits
250 //@shungfu: Edit at Hw3
251     if(kernel->memManageUnit->getPagingType() == LRU){  // LRU
252         kernel->physPageTable->load_time[pageFrame] = kernel->memManageUnit->counter;
253         kernel->memManageUnit->counter++;
254     }
```

Machine::Translate(), LRU

```
96      kernel->physPageTable->load_time[victim] = kernel->memManageUnit->loading_time;
97      kernel->memManageUnit->loading_time++;
```

MMU::swap_in(), FIFO

執行結果：

測試程式 test4.c:

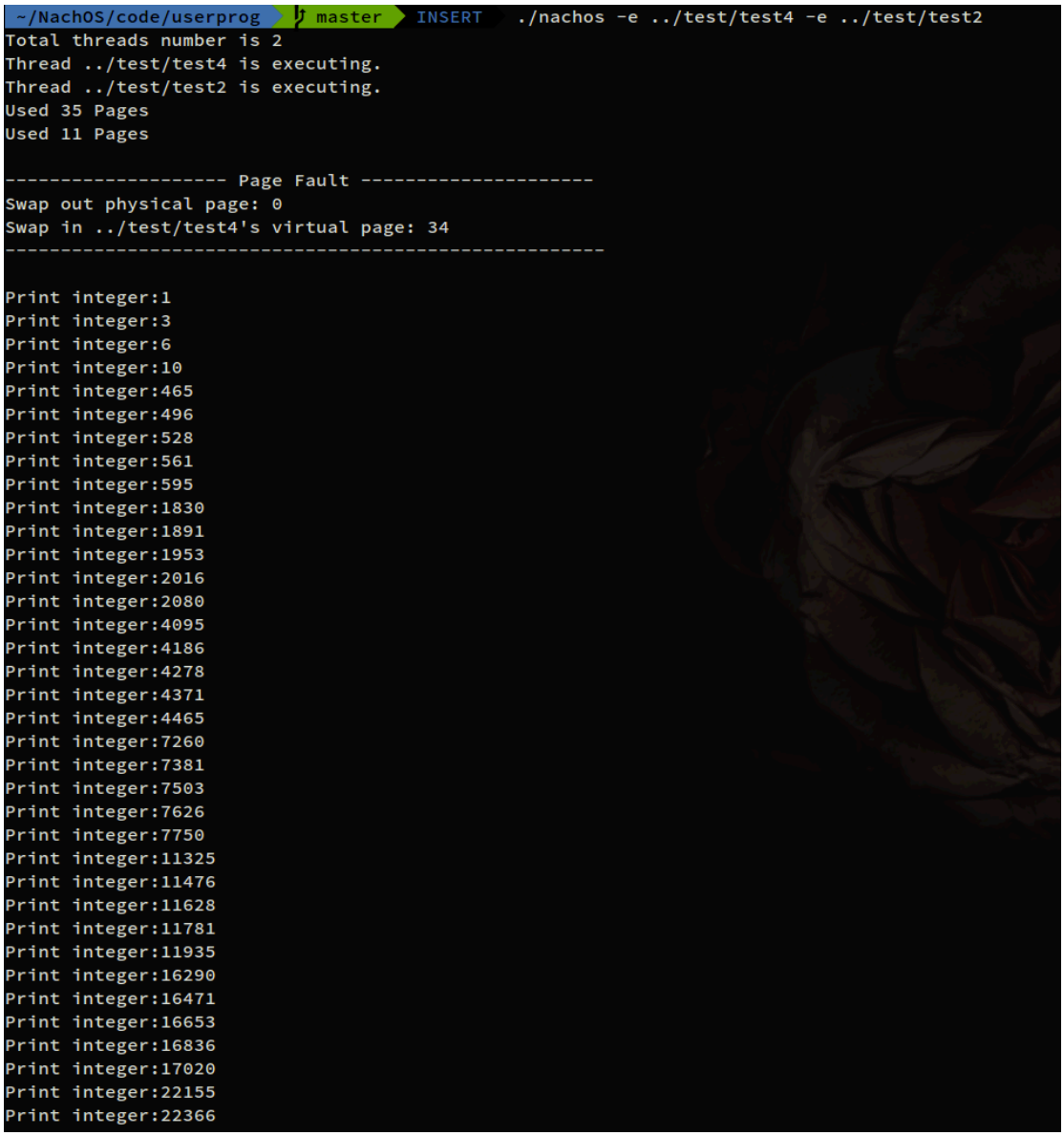
```
1 #include "syscall.h"
2
3 # define SIZE 700
4
5 int x[SIZE] = {0};
6
7 int main(){
8     int i;
9
10    x[0] = 0;
11    for(i=1;i<SIZE;++i){
12        x[i] = x[i-1] + i;
13        if((i % 30 < 5))
14            PrintInt(x[i]);
15    }
16    return 0;
17 }
```

只有設 global 或 static，對 nachos 來說才會增加 page size。

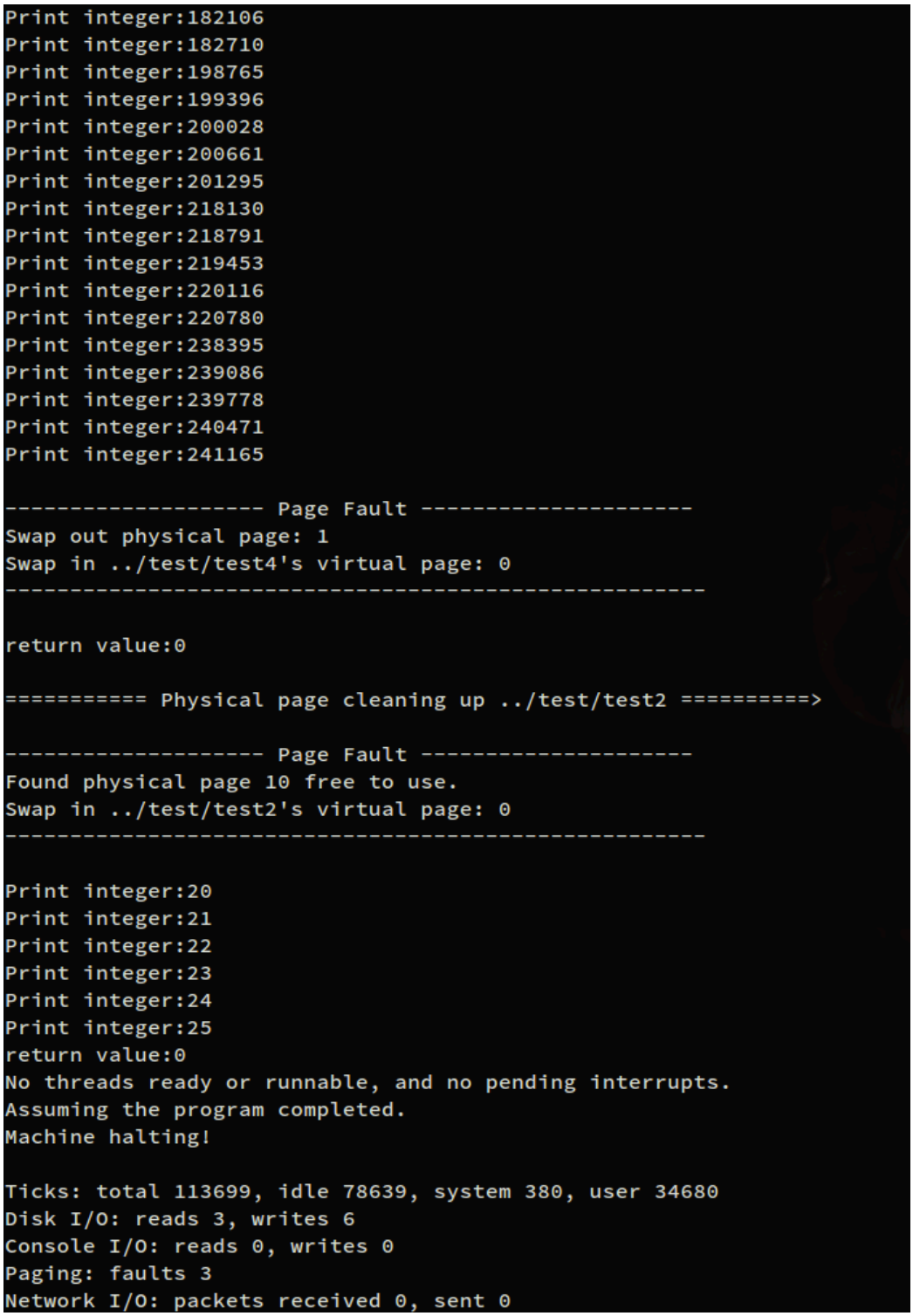
執行方法，預設為 FIFO，輸入-LRU 就會設為 LRU。

FIFO:

./nachos -e ../test/test4 -e ../test/test2



（中間省略…）沒有 page fault



LRU:

./nachos -e ../test/test4 -e ../test/test2 -LRU

```
~/NachOS/code/userprog  master  INSERT  ./nachos -e ../test/test4 -e ../test/test2 -LRU
Total threads number is 2
Thread ../test/test4 is executing.
Thread ../test/test2 is executing.
Used 35 Pages
Used 11 Pages

-------------------- Page Fault --------------------
Swap out physical page: 1
Swap in ../test/test4's virtual page: 34
----------------------------------------------------


-------------------- Page Fault --------------------
Swap out physical page: 3
Swap in ../test/test4's virtual page: 1
----------------------------------------------------


-------------------- Page Fault --------------------
Swap out physical page: 5
Swap in ../test/test4's virtual page: 3
----------------------------------------------------

Print integer:1
Print integer:3
Print integer:6
Print integer:10

-------------------- Page Fault --------------------
Swap out physical page: 6
Swap in ../test/test4's virtual page: 5
----------------------------------------------------

Print integer:465
Print integer:496
Print integer:528
Print integer:561
Print integer:595

-------------------- Page Fault --------------------
Swap out physical page: 7
Swap in ../test/test4's virtual page: 6
----------------------------------------------------

Print integer:1830
```

```
Print integer:1891
Print integer:1953
Print integer:2016
Print integer:2080

-------------------- Page Fault --------------------
Swap out physical page: 8
Swap in ../test/test4's virtual page: 7
----------------------------------------------------

Print integer:4095
Print integer:4186
Print integer:4278
Print integer:4371
Print integer:4465

-------------------- Page Fault --------------------
Swap out physical page: 9
Swap in ../test/test4's virtual page: 8
----------------------------------------------------

Print integer:7260
Print integer:7381
Print integer:7503
Print integer:7626
Print integer:7750

-------------------- Page Fault --------------------
Swap out physical page: 10
Swap in ../test/test4's virtual page: 9
----------------------------------------------------

Print integer:11325
Print integer:11476
Print integer:11628
Print integer:11781
Print integer:11935

-------------------- Page Fault --------------------
Swap out physical page: 11
```

```
Swap in ../test/test4's virtual page: 10
-------------------------------------------------------

Print integer:16290
Print integer:16471
Print integer:16653
Print integer:16836
Print integer:17020
Print integer:22155
Print integer:22366

-------------------- Page Fault ---------------------
Swap out physical page: 12
Swap in ../test/test4's virtual page: 11
-------------------------------------------------------

Print integer:22578
Print integer:22791
Print integer:23005
Print integer:28920
Print integer:29161
Print integer:29403
Print integer:29646

-------------------- Page Fault ---------------------
Swap out physical page: 13
Swap in ../test/test4's virtual page: 12
-------------------------------------------------------

Print integer:29890
Print integer:36585
Print integer:36856
Print integer:37128
Print integer:37401
Print integer:37675

-------------------- Page Fault ---------------------
Swap out physical page: 14
Swap in ../test/test4's virtual page: 13
-------------------------------------------------------

Print integer:45150
Print integer:45451
Print integer:45753
Print integer:46056
```

```
Print integer:46360

-------------------- Page Fault ---------------------
Swap out physical page: 15
Swap in ../test/test4's virtual page: 14
-------------------------------------------------------

Print integer:54615
Print integer:54946
Print integer:55278
Print integer:55611
Print integer:55945

-------------------- Page Fault ---------------------
Swap out physical page: 16
Swap in ../test/test4's virtual page: 15
-------------------------------------------------------

Print integer:64980
Print integer:65341
Print integer:65703
Print integer:66066
Print integer:66430

-------------------- Page Fault ---------------------
Swap out physical page: 17
Swap in ../test/test4's virtual page: 16
-------------------------------------------------------

Print integer:76245
Print integer:76636
Print integer:77028
Print integer:77421
Print integer:77815

-------------------- Page Fault ---------------------
Swap out physical page: 18
Swap in ../test/test4's virtual page: 17
-------------------------------------------------------
```

```
Print integer:88410
Print integer:88831
Print integer:89253
Print integer:89676
Print integer:90100

-------------------- Page Fault --------------------
Swap out physical page: 19
Swap in ../test/test4's virtual page: 18
----------------------------------------------------

Print integer:101475
Print integer:101926
Print integer:102378
Print integer:102831
Print integer:103285

-------------------- Page Fault --------------------
Swap out physical page: 20
Swap in ../test/test4's virtual page: 19
----------------------------------------------------

Print integer:115440
Print integer:115921
Print integer:116403
Print integer:116886
Print integer:117370

-------------------- Page Fault --------------------
Swap out physical page: 21
Swap in ../test/test4's virtual page: 20
----------------------------------------------------

Print integer:130305
Print integer:130816
Print integer:131328
Print integer:131841
Print integer:132355

-------------------- Page Fault --------------------
Swap out physical page: 22
Swap in ../test/test4's virtual page: 21
----------------------------------------------------

Print integer:146070
```

```
Print integer:146611
Print integer:147153
Print integer:147696
Print integer:148240

-------------------- Page Fault --------------------
Swap out physical page: 23
Swap in ../test/test4's virtual page: 22
----------------------------------------------------

Print integer:162735
Print integer:163306
Print integer:163878
Print integer:164451
Print integer:165025

-------------------- Page Fault --------------------
Swap out physical page: 24
Swap in ../test/test4's virtual page: 23
----------------------------------------------------

Print integer:180300
Print integer:180901
Print integer:181503
Print integer:182106
Print integer:182710

-------------------- Page Fault --------------------
Swap out physical page: 25
Swap in ../test/test4's virtual page: 24
----------------------------------------------------

Print integer:198765
Print integer:199396
Print integer:200028
Print integer:200661
Print integer:201295

-------------------- Page Fault --------------------
Swap out physical page: 26
Swap in ../test/test4's virtual page: 25
----------------------------------------------------

Print integer:218130
Print integer:218791
```

```
Print integer:219453
Print integer:220116
Print integer:220780
Print integer:238395
Print integer:239086

-------------------- Page Fault --------------------
Swap out physical page: 27
Swap in ../test/test4's virtual page: 26
----------------------------------------------------

Print integer:239778
Print integer:240471
Print integer:241165
return value:0

========== Physical page cleaning up ../test/test2 ==========>

-------------------- Page Fault --------------------
Found physical page 10 free to use.
Swap in ../test/test2's virtual page: 0
----------------------------------------------------

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 449699, idle 412776, system 2220, user 34703
Disk I/O: reads 26, writes 29
Console I/O: reads 0, writes 0
Paging: faults 26
Network I/O: packets received 0, sent 0
~/NachOS/code/userprog  master  INSERT
```

這邊測完發現，LRU 竟然比 FIFO 有更多的 page fault，打開 debug 後會發現原來是因為每次我 swap out 的都是我下一次要使用的。第一個最小的 load time 之所以是 page #1 是因為，page #0 在前面已經有被存取過了所以他的 load time 增加 page #1 自然就變成最小的。

附圖：（實作前的 segment fault）

```
~/NachOS/code/userprog  master +●  INSERT  ./nachos -e ../test/test4                    ✓ 00:51:58
Total threads number is 1
Thread ../test/test4 is executing.
[9]    16411 segmentation fault (core dumped)  ./nachos -e ../test/test4
~/NachOS/code/userprog  master +●  INSERT                    SIGSEGV(11) ↵ 00:52:02
```

```
~/NachOS/code  master ● ?  INSERT  userprog/./nachos -d 3 -e test/test4
Total threads number is 1
Thread test/test4 is executing.

Used 34 Pages
code size:512
initData size: 2800
UninitData size: 0
size of AddrSpace: 4352

i: 0, Free PhyPages: 32
i: 1, Free PhyPages: 31
i: 2, Free PhyPages: 30
i: 3, Free PhyPages: 29
i: 4, Free PhyPages: 28
i: 5, Free PhyPages: 27
i: 6, Free PhyPages: 26
i: 7, Free PhyPages: 25
i: 8, Free PhyPages: 24
i: 9, Free PhyPages: 23
i: 10, Free PhyPages: 22
i: 11, Free PhyPages: 21
i: 12, Free PhyPages: 20
i: 13, Free PhyPages: 19
i: 14, Free PhyPages: 18
i: 15, Free PhyPages: 17
i: 16, Free PhyPages: 16
i: 17, Free PhyPages: 15
i: 18, Free PhyPages: 14
i: 19, Free PhyPages: 13
i: 20, Free PhyPages: 12
i: 21, Free PhyPages: 11
i: 22, Free PhyPages: 10
i: 23, Free PhyPages: 9
i: 24, Free PhyPages: 8
i: 25, Free PhyPages: 7
i: 26, Free PhyPages: 6
i: 27, Free PhyPages: 5
i: 28, Free PhyPages: 4
i: 29, Free PhyPages: 3
i: 30, Free PhyPages: 2
i: 31, Free PhyPages: 1
i: 32, Free PhyPages: 0
sec: 0
code.infileAddr: 40, code.virtualAddr: 0
[32]    2289 segmentation fault (core dumped)  userprog/./nachos -d 3 -e test/test4
```

遇到困難：

在執行兩個超過 physical memory size 程式的時候發現會壞掉，因為我有在 addrSpace 的 destructor 清掉 memory 和 backing store 的一些設定，想說是哪裡壞掉了，結果追進去發現是，thread 結束時，沒有刪除 addrSpace。所以我將 addrSpace 刪除後好像就正常了，因為沒有馬上測試後續還有做一些更動，所以不太確定是不是這個原因，但是問題解決了。

```
58 //----------------------------------------------------------------------
59 // Thread::~Thread
60 //  De-allocate a thread.
61 //
62 //  NOTE: the current thread *cannot* delete itself directly,
63 //  since it is still running on the stack that we need to delete.
64 //
65 //      NOTE: if this is the main thread, we can't delete the stack
66 //      because we didn't allocate it -- we got it automatically
67 //      as part of starting up Nachos.
68 //----------------------------------------------------------------------
69
70 Thread::~Thread()
71 {
72     DEBUG(dbgThread, "Deleting thread: " << name);
73 #ifdef USER_PROGRAM
74     delete space;
75     DEBUG(dbgHw3,"Deleting thread: " << name);
76 #endif
77     ASSERT(this != kernel->currentThread);
78     if (stack != NULL)
79     DeallocBoundedArray((char *) stack, StackSize * sizeof(int));
80 }
```