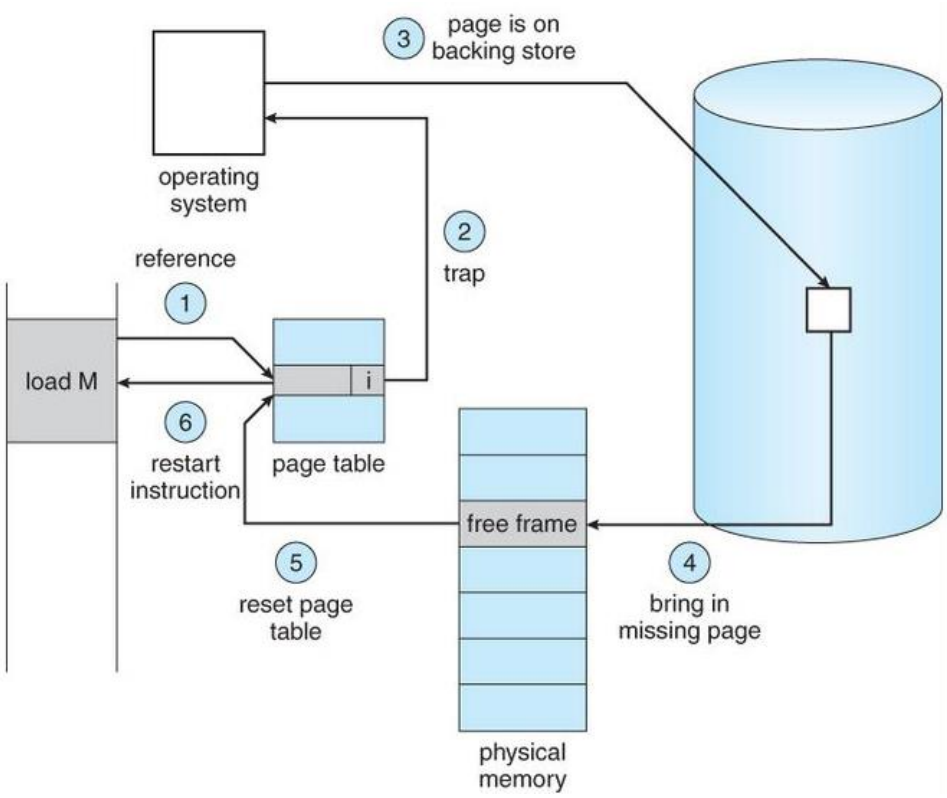
HW3 Memory Management

R08942087 吳彬睿

1. Motivation

(1) Memory Management

沒有 implement virtual memroy，所以當執行這兩個 process 的時候，memory 爆掉了，會 segmentation fault，所以沒有執行成功。



需要實做一個 page table 跟 frame table，swap 的對照表，當今天記憶體已經吃滿，但是又須要開一個新的空間的時候，這時候會把東西寫入 virtual memory 裡面，當之後有記憶體被 free 掉了，有多餘的空間，並且 process 需要用到這個 page 的時候，才會把這塊 page load 回去記憶體空間。

2. Implementation

(1) Memory Management

- ./userprog/userkernel.h

```
19 class SynchDisk;
20 class UserProgKernel : public ThreadedKernel {
21   public:
22     UserProgKernel(int argc, char **argv);
23                 // Interpret command line arguments
24     ~UserProgKernel();      // deallocate the kernel
25
26     void Initialize(SchedulerType);     // initialize the kernel
27
28     void Run();         // do kernel stuff
29
30     void SelfTest();        // test whether kernel is working
31
32     /////
33     SynchDisk *vm_Disk; // to save pages which main memory has no free space
34
35 // These are public for notational convenience.
36     Machine *machine;
37     FileSystem *fileSystem;
38
39     ////
40     bool debugUserProg;
41
42 #ifdef FILESYS
43     SynchDisk *synchDisk;
44 #endif // FILESYS
45
46   private:
47     //bool debugUserProg;       // single step user program
48     Thread* t[10];
49     char*   execfile[10];
50     int execfileNum;
51 };
```

Create 一個 vm_Disk 的空間，待會要來放我們的 swap 的記憶體空間。

- ./userprog/userkernel.cc

```
54 void
55 UserProgKernel::Initialize(SchedulerType type)
56 {
57     ThreadedKernel::Initialize(type);   // init multithreading
58
59     machine = new Machine(debugUserProg);
60     fileSystem = new FileSystem();
61
62     // to save pages which main memory has not enough space to save
63     vm_Disk = new SynchDisk("New Disk");
64
65 #ifdef FILESYS
66     synchDisk = new SynchDisk("New SynchDisk");
67 #endif // FILESYS
68 }
69
```

- ./userprog/addrspace.h

```
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-specific
31     void RestoreState();        // info on a context switch
32     ////
33     int ID;
34
35   private:
36     TranslationEntry *pageTable;    // Assume linear page table translation
37                     // for now!
38     unsigned int numPages;      // Number of pages in the virtual
39                     // address space
40
41     bool Load(char *fileName);      // Load the program into memory
42                     // return false if not found
43
44     void InitRegisters();       // Initialize user-level CPU registers,
45                     // before jumping to user code
46     ////
47     bool pt_is_load;
48
49     static bool PhyPageStatus[NumPhysPages];
50     static int NumFreePages;
51
52 };
53
54 #endif // ADDRSPACE_H
```

每一個 process 多開一個變數，判斷此 process 是不是 loading on CPU 正在執行。

- ./userprog/addrspace.cc

```
54 AddrSpace::AddrSpace()
55 {
56     ID=(kernel->machine->ID_num)+1;
57     kernel->machine->ID_num = kernel->machine->ID_num+1;
58     /*pageTable = new TranslationEntry[NumPhysPages];
59     for (unsigned int i = 0; i < NumPhysPages; i++) {
60         pageTable[i].virtualPage = i;   // for now, virt page # = phys page #
61         pageTable[i].physicalPage = i;
62         //  pageTable[i].physicalPage = 0;
63         pageTable[i].valid = TRUE;
64         //  pageTable[i].valid = FALSE;
65         pageTable[i].use = FALSE;
66         pageTable[i].dirty = FALSE;
67         pageTable[i].readOnly = FALSE;
68       }
69 */
70     // zero out the entire address space
71 //    bzero(kernel->machine->mainMemory, MemorySize);
72 }
```

Process 的 ID。

```
84 AddrSpace::~AddrSpace()
85 {
86     /*
87     for(int i = 0; i < numPages; i++){
88         AddrSpace::PhyPageStatus[pageTable[i].physicalPage] = PAGE_FREE;
89         AddrSpace::NumFreePages++
90     }
91     */
92     delete pageTable;
93 }
94
```

AddrSpace::Load()

```
131     //////
132     pageTable = new TranslationEntry[numPages];
133
134     size = numPages * PageSize;
135
136     // ASSERT(numPages ≤ NumPhysPages);        // check we're not trying
137                         // to run anything too big --
138                         // at least until we have
139                         // virtual memory
140
141 ////////////////////////////
142     //ASSERT(numPages ≤ NumFreePages);
143     pageTable = new TranslationEntry[numPages];
144
145     /*
146     for(unsigned int i = 0, idx = 0; i < numPages; i++) {
147         pageTable[i].virtualPage = i;
148         while(idx < NumPhysPages && AddrSpace::PhyPageStatus[idx] == PAGE_OCCU)
149             idx++;
150         AddrSpace::PhyPageStatus[idx] = PAGE_OCCU;
151         AddrSpace::NumFreePages--;
152         //清空即將分配的 page
153         bzero(&kernel->machine->mainMemory[idx * PageSize], PageSize);
154         pageTable[i].physicalPage = idx;
155         pageTable[i].valid = true;
156         pageTable[i].use = false;
157         pageTable[i].dirty = false;
158         pageTable[i].readOnly = false;
159     }
160     */
161 ////////////////////////////////
162
163     // DEBUG(dbgAddr, "Initializing address space: " << numPages << ", " << size);
```

Load process 的時候，幫 process create 一個專屬的 pageTable。

```
166     if (noffH.code.size > 0) {
167         // DEBUG(dbgAddr, "Initializing code segment.");
168         // DEBUG(dbgAddr, noffH.code.virtualAddr << ", " << noffH.code.size);
169         for(unsigned int j=0,i=0; i<numPages; i++){
170             j=0;
171             while(kernel->machine->usedPhyPage[j] != FALSE && j<NumPhysPages){
172                 j++;
173             }
174 
175             if(j<NumPhysPages)
176                 // if memeory is enough, just put data in without using virtual memory
177                 kernel->machine->usedPhyPage[j]=TRUE;
178                 kernel->machine->PhyPageName[j]=ID;
179                 kernel->machine->main_tab[j]=&pageTable[i];
180                 pageTable[i].physicalPage = j;
181                 pageTable[i].valid = TRUE;
182                 pageTable[i].use = FALSE;
183                 pageTable[i].dirty = FALSE;
184                 pageTable[i].readOnly = FALSE;
185                 pageTable[i].ID = ID;
186                 pageTable[i].count ++ ; // for LRU, count+1 when save in memeory
187                 pageTable[i].reference_bit = FALSE; // for second chance algo.
188                 executable->ReadAt(&(kernel->machine->mainMemory[j*PageSize]),
189                                    PageSize, noffH.code.inFileAddr+(i*PageSize));
190                 /*
191                 executable->ReadAt(
192                 &(kernel->machine->mainMemory[pageTable[noffH.code.virtualAddr/PageSize].physicalPag
    e * PageSize + (noffH.code.virtualAddr%PageSize)]),
193                                    noffH.code.size, noffH.code.inFileAddr);
194                 */
195             } else {
196                 // Use virtual memory when memory isn't enough
197                 char *buf = new char [PageSize];
198                 k=0;
199                 while(kernel->machine->usedvirPage[k] != FALSE){
200                     k++;
201                 }
202 
203                 kernel->machine->usedvirPage[k] = true;
204                 pageTable[i].virtualPage = k; //record which virtualpage you save
205                 pageTable[i].valid = FALSE; // not load in main_memory
206                 pageTable[i].use = FALSE;
207                 pageTable[i].dirty = FALSE;
208                 pageTable[i].readOnly = FALSE;
209                 pageTable[i].ID = ID;
210                 executable->ReadAt(buf, PageSize, noffH.code.inFileAddr+(i*PageSize));
211                 kernel->vm_Disk->WriteSector(k,buf); // call virtual_disk write in virtual memory
212             }
213         }
```

判斷如果 memory page 還有多的，那就讓它寫入新的 page，如果 memory 已經滿的話，那就要寫入 virtual memory 了。

```
217     }
218     if (noffH.initData.size > 0) {
219         // DEBUG(dbgAddr, "Initializing data segment.");
220         // DEBUG(dbgAddr, noffH.initData.virtualAddr << ", " << noffH.initData.size);
221 
222         executable->ReadAt(&(kernel->machine->mainMemory[noffH.initData.virtualAddr]),
223                            noffH.initData.size, noffH.initData.inFileAddr);
224         /*
225          executable->ReadAt(
226                 &(kernel->machine->mainMemory[pageTable[noffH.initData.virtualAddr/PageSize].physica
    lPage * PageSize + (noffH.code.virtualAddr%PageSize)]),
227                           noffH.initData.size, noffH.initData.inFileAddr);
228         */
229 
230         /*executable->ReadAt(
231         &(kernel->machine->mainMemory[noffH.initData.virtualAddr]),
232             noffH.initData.size, noffH.initData.inFileAddr);*/
233     }
234 
235     delete executable;          // close file
236     return TRUE;            // success
237 }
```

```
247 void
248 AddrSpace::Execute(char *fileName)
249 {
250     /////
251     pt_is_load = FALSE;
252
253     if (!Load(fileName)) {
254     cout << "inside !Load(FileName)" << endl;
255     return;             // executable not found
256     }
257
258     //kernel->currentThread->space = this;
259     this->InitRegisters();      // set the initial register values
260     this->RestoreState();       // load page table register
261
262     ////
263     pt_is_load = TRUE;
264
265     kernel->machine->Run();     // jump to the user progam
266
267     ASSERTNOTREACHED();         // machine->Run never returns;
268                     // the address space exits
269                     // by doing the syscall "exit"
270 }
271
```

當 process 在執行的時候，把 is load 的 variable 設定成 true。

```
314 void AddrSpace::SaveState()
315 {
316     ////
317     if(pt_is_load){
318         pageTable=kernel->machine->pageTable;
319         numPages=kernel->machine->pageTableSize;
320     }
321 }
322
```

把 kernel 的 page table 丟給自己的 page table，做同步。

- ./machine/machine.h

```
132     TranslationEntry *pageTable;
133     unsigned int pageTableSize;
134     bool ReadMem(int addr, int size, int* value);
135
136     ////////////
137     bool usedPhyPage[NumPhysPages]; // record which page in the main memory is used
138     bool usedvirPage[NumPhysPages];
139     int ID_num;
140     int PhyPageName[NumPhysPages];
141     int count[NumPhysPages];          // for LRU
142     bool reference_bit[NumPhysPages]; // for second chance algo.
143     int sector_number;
144
145     TranslationEntry *main_tab[NumPhysPages];
146
147   private:
148
```

當主記憶體不夠的時候，需要把未分配的 page 放的 virtual memory 裡面。

- ./machine/translate.h

```
30 class TranslationEntry {
31   public:
32     unsigned int virtualPage;   // The page number in virtual memory.
33     unsigned int physicalPage;  // The page number in real memory (relative to the
34             //  start of "mainMemory"
35     bool valid;         // If this bit is set, the translation is ignored.
36             // (In other words, the entry hasn't been initialized.)
37     bool readOnly;  // If this bit is set, the user program is not allowed
38             // to modify the contents of the page.
39     bool use;           // This bit is set by the hardware every time the
40             // page is referenced or modified.
41     bool dirty;         // This bit is set by the hardware every time the
42             // page is modified.
43     ////
44     int ID;
45     int count;
46     bool reference_bit;
47 };
48
49 #endif
```

Translate 是在換記憶體的地方。

- ./machine/translate.cc

```
219     } else if (!pageTable[vpn].valid) {
220         //DEBUG(dbgAddr, "Invalid virtual page # " << virtAddr);
221         //return PageFaultException;
222         printf("page falut'\n");
223         kernel->stats->numPageFaults++;
224         j=0;
225         while(kernel->machine->usedPhyPage[j]!=FALSE && j<NumPhysPages){
226             j++;
227         }
228         // add the page into the main memory if the main memory isn't full
229         if(j<NumPhysPages){
230             char *buf;
231             buf = new char[PageSize];
232             kernel->machine->usedPhyPage[j] = TRUE;
233             kernel->machine->PhyPageName[j] = pageTable[vpn].ID;
234
235             kernel->machine->main_tab[j]=&pageTable[vpn];
236             pageTable[vpn].physicalPage = j;
237
238             kernel->vm_Disk->ReadSector(pageTable[vpn].virtualPage, buf);
239             bcopy(buf, &mainMemory[j*PageSize], PageSize);
240
241         }else{
242             char *buf_1 = new char[PageSize];
243             char *buf_2 = new char[PageSize];
244
245             victim = (rand()%32);
246
247             printf("Number=%d page swap out\n", victim);
248
249             //get the page victim and save in the disk
250             bcopy(&mainMemory[victim*PageSize], buf_1, PageSize);
251             kernel->vm_Disk->ReadSector(pageTable[vpn].virtualPage, buf_2);
252             bcopy(buf_2, &mainMemory[victim*PageSize], PageSize);
253             kernel->vm_Disk->WriteSector(pageTable[vpn].virtualPage, buf_1);
254
255             main_tab[victim]->virtualPage = pageTable[vpn].virtualPage;
256             main_tab[victim]->valid = FALSE;
257
258             // save page into main memory
259             pageTable[vpn].valid = TRUE;
260             pageTable[vpn].physicalPage = victim;
261             kernel->machine->PhyPageName[victim] = pageTable[vpn].ID;
262             main_tab[victim] = & pageTable[vpn];
263             // fifo = fifo+1; // for fifo
264             printf("page replactment finish\n");
265         }
266     }
267     entry = &pageTable[vpn]
```

當 main memory 沒有滿的時候，直接把 page assign 到 main memory 去，但是

當 main memory 滿的時候，需要 random select 一個 victim，然後把他請回 disk 去，並把我們想要用的東西 load 到 main memory，做一個 page replacement 的動作。

- ./test/sort.c

```c
#include "syscall.h"

int A[1024];    /* size of physical memory; with code, we'll run out of space!*/

int
main()
{
    int i, j, tmp;

    /* first initialize the array, in reverse sorted order */
    for (i = 0; i < 1024; i++)
        A[i] = 1024 - i;

    /* then sort! */
    for (i = 0; i < 1023; i++)
        for (j = 0; j < (1023 - i); j++)
       if (A[j] > A[j + 1]) {   /* out of order -> need to swap ! */
          tmp = A[j];
          A[j] = A[j + 1];
          A[j + 1] = tmp;
           }
    Exit(A[0]);     /* and then we're done -- should be 0! */
}
```

這邊 sort algorithm 好像寫錯了，要改成 j=0 開始，最後結果才會是 1。

3. Result

(1) Memory Management

- ./nachos -e ../test/matmult

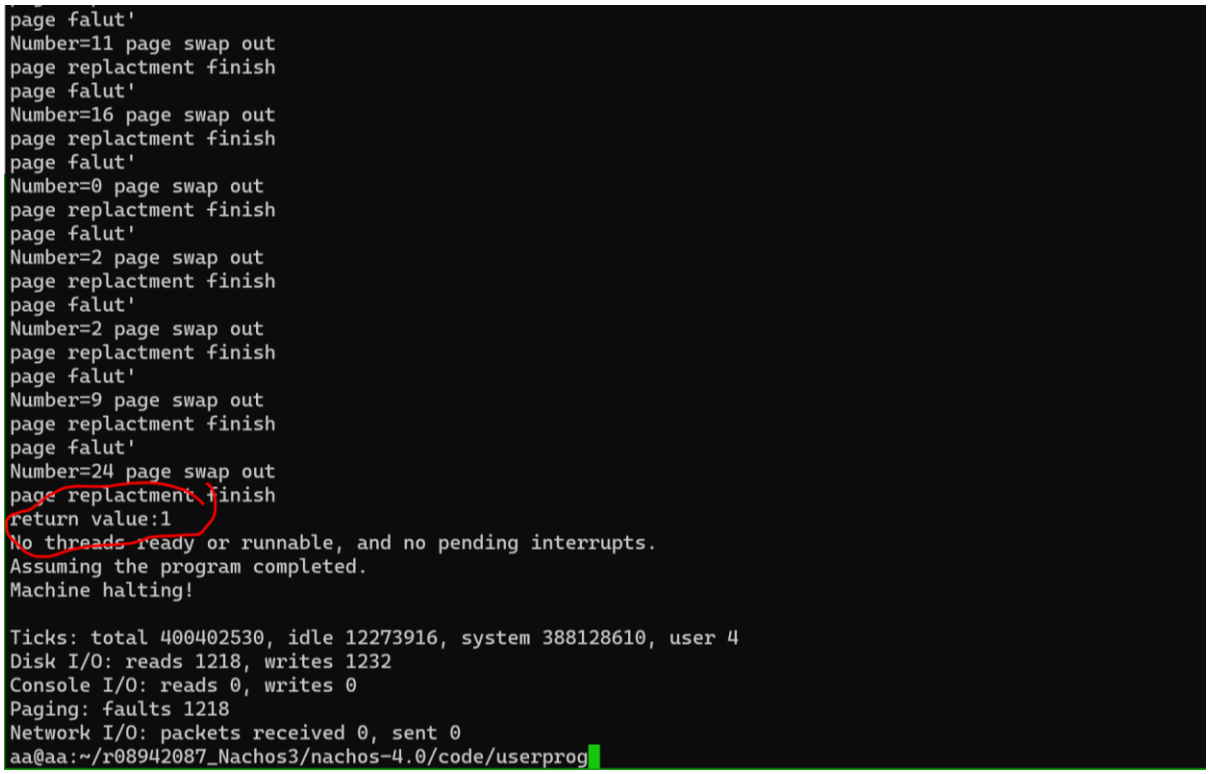
```
page falut'
Number=28 page swap out
page replactment finish
page falut'
Number=10 page swap out
page replactment finish
page falut'
Number=12 page swap out
page replactment finish
page falut'
Number=16 page swap out
page replactment finish
page falut'
Number=27 page swap out
page replactment finish
page falut'
Number=27 page swap out
page replactment finish
page falut'
Number=18 page swap out
page replactment finish
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 7651580, idle 1325676, system 6325900, user 4
Disk I/O: reads 89, writes 111
Console I/O: reads 0, writes 0
Paging: faults 89
Network I/O: packets received 0, sent 0
```

Result=7220

總共有 89 次 page fault。

- ./nachos -e ../test/sort



Result=1

總共有 1218 次 page fault。

我發現在做 sort 的時候有很多 page fault，可能的原因是因為 sort 比較是線性的去讀數值，因此每個都會被 look once，導致需要 load 很多不同的 page，也是因此產生很多 page fault。