Content

Reference

1. MrOrz/nachos. [Online]. Available: https://github.com/MrOrz/nachos
https://github.com/taldehyde/os-project-1/blob/master/report.md
2. Caching and Virtual Memory. [Online]. Available:
http://neuron.csie.ntust.edu.tw/homework/93/OS/homework_3/A9315010-%E7%AC%AC%E4%B9%9D%E7%B5%84/phase%203%20operation.htm

I. Motivation

Motivation and the problem analysis

✓ 若在還沒有改 nachos-4.0 的原始碼以前，同時執行 兩個 program

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

這時會因為超過預先設定的 physical page number (預設為 32)，而產生 core dump。

如下圖所示：

```
r08921005@r08921005-VirtualBox:~/nachos/3/nachos-4.0/code/userprog$ ./nachos -e ../test/matmult -e ../test/sort
Total threads number is 2
Thread ../test/matmult is executing.
Thread ../test/sort is executing.
Assertion failed: line 134 file ../userprog/addrspace.cc
Aborted (core dumped)
```

Fig. 1 修改前所遇到的問題

由於 matmult 與 sort 兩者 thread 所佔用到的 physical page 會超過預設的 32 導致 Aborted

原先 nachos 一次只保留一個 user program 在主記憶體中。

此專題目標在於使用 paging mechanism 來實作 multiprogramming，

也就是在主記憶體中保留多個 user programs，而 pages 只有在這個 program 被執行的時候才會被 loaded，也就是 pure demand paging。

What's your plan to deal with the problem

以 Program matmult 與 Program sort 為例。

輸入 nachos -e ../test/matmult -e ../test/sort 之後

1. 先幫這兩個 Program 各創一個 thread 與 page table，也就是生成 thread A 與 thread B。
2. 利用 AddrSpace::Load() 函式把 Program matmult 與 Program sort 之 page table 匯入進硬碟中，同時也把這兩個 thread 丟進 ready queue 裡面，這個時候 OS 會決定要讀取的程式碼在的起始地址與總共要載入多少分頁到硬碟裡。當作業系統開始執行後(執行 kernel->machine->Run()後)，選擇其中一個 thread 來執行。當在 page Table 中找不到要跑的 thread 時，發生 page fault，並去磁碟機者到對應的 page (demanded page)與執行 context switching(timer 讓 interrupt 取引發 context switching)，開始執行另外一個 thread。

在此專題中沒有去製造新的 Memory Manager，而是把 virtual address 與 physical address 的對應關係寫在 AddrSpace::Load()裡，利用 Page Size 跟 virtual address 去找對應 physical address。

$$\boldsymbol{Physical\ Address} = \boldsymbol{pageTable}\left[\left(\frac{\boldsymbol{virtual\ address}}{\boldsymbol{PageSize}}\right)\right].\boldsymbol{physicalPage} * \boldsymbol{PageSize} + (\boldsymbol{virtual\ address}\ \%\ \boldsymbol{PageSize})$$

排程的 page replacement 使用 LRU(Least Recently Used).

同時，這邊使用 3 種 table 來維護 physical page 資訊與 swap 進入或換出主記憶體的資訊。

PageTable	One page table per process. Decide your virtual page number. Kernel->machine->pageTable = currentThread ->pageTable
FrameTable	整個 system 一個 紀錄每個 physical page 的資訊
SwapTable	整個 system 一個 紀錄 swap 內每個 sector 的資訊

II. Implementation

How do you implement to solve the problem in Nachos

相較於第二個 project，改動程式碼檔案如下表：

/userprog/userkernel.h /userprog/userkernel.cc	新增 swapDisk
/ userprog/exception.cc	新增 PageFaultException
/userprog/addrspace.h /userprog/addrspace.cc	Address Mapping 檢查有沒有閒置的 physical page
/machine/translate.h /machine/translate.cc	實作 LRU
/machine/machine.h	紀錄 sector number, frame paged 和被 main memory, virtual memory 占用的 frame

You can including some important code segments and comments

新增 swapDisk

```
34    FileSystem *fileSystem;
```

Original Text

```
34    FileSystem *fileSystem;
35    SynchDisk *swapDisk;
```

Changed Text

/code/userprog/userkernel.h

```
59    fileSystem = new FileSystem();
```

Original Text

```
59    fileSystem = new FileSystem();
60    swapDisk = new SynchDisk("New VMDisk");
```

Changed Text

/code/userprog/userkernel.cc

```
68              case SC_Sleep:
69                      val=kernel->machine->ReadRegister(4);
70                      cout << "Sleep Time " << val << "(ms) " << endl;
71                      kernel->alarm->WaitUntil(val);
72                      return;
```

Changed Text

/code/userprog/exception.cc

將程式載入虛擬記憶體，也就是磁碟機中。載入的同時會紀錄 virtual page 的起始位置還有要載入多少分頁到磁碟機中。然而如果可以用的 physical page 已經沒有了，則會把這段程式碼 Load 進 virtual page 裡面，並把它寫到 swap 區域中。

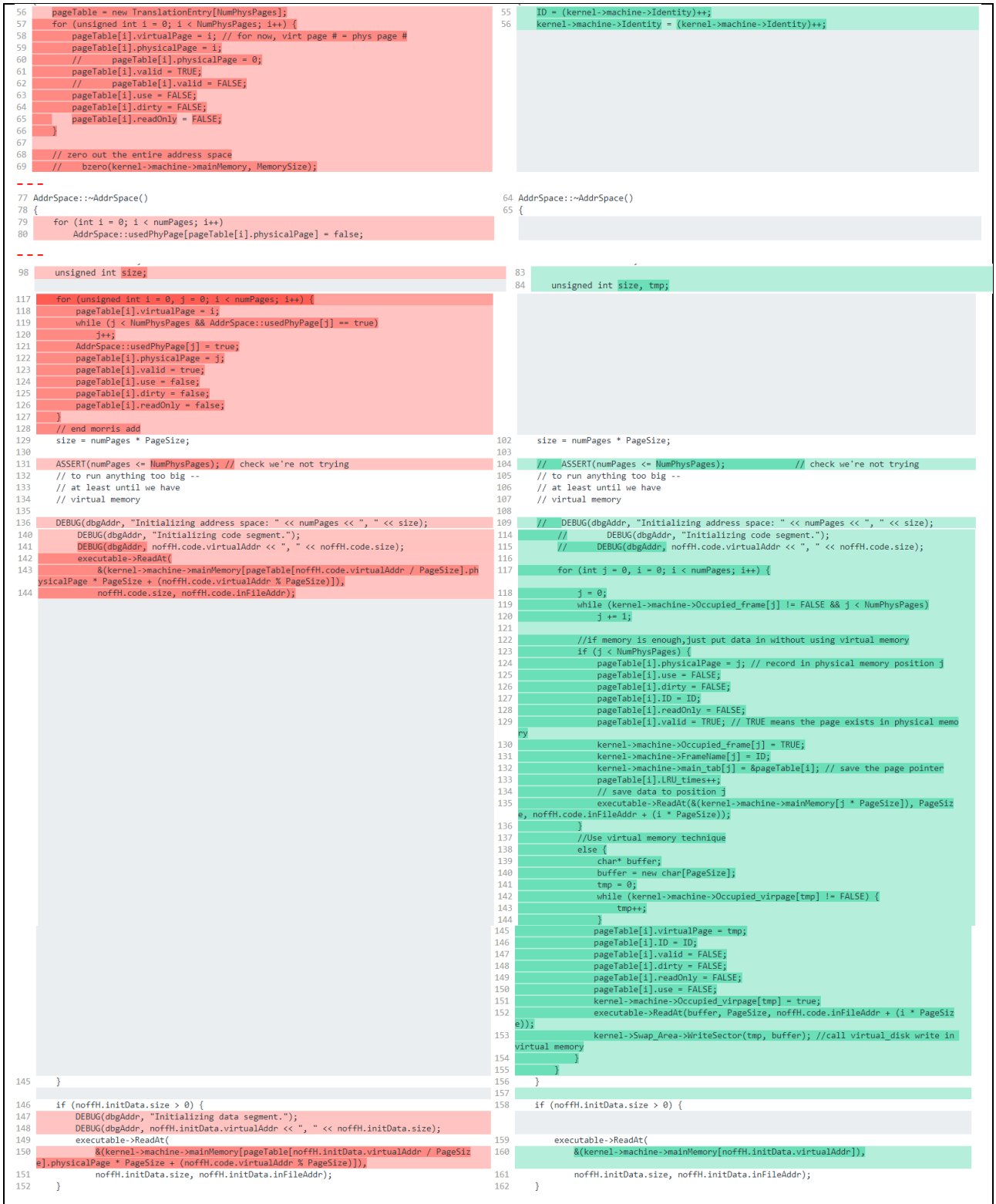
```
32
33  private:
34    TranslationEntry *pageTable;       // Assume linear page table translation
35                                       // for now!
36    unsigned int numPages;             // Number of pages in the virtual
37                                       // address space
38
39    bool Load(char *fileName);         // Load the program into memory
40                                       // return false if not found
41
42    void InitRegisters();              // Initialize user-level CPU registers,
43                                       // before jumping to user code
```

Original Text

```
33    void pageFaultHandle(int);
34    static bool usedPhyPage[NumPhysPages];
35    static bool usedVirPage[NumPhysPages];
36    static bool pageType[NumPhysPages]; //code:1 data:0;
37    static TranslationEntry *ptrPageTable[NumPhysPages];
38    // recording which virtual pages are used.
39    // It can be as large as we want it to be :P
40
41
42  private:
43    TranslationEntry *pageTable;       // Assume linear page table translation
44                                       // for now!
45    unsigned int numPages;             // Number of pages in the virtual
46                                       // address space
47
48    bool Load(char *fileName);         // Load the program into memory
49                                       // return false if not found
50
51    void InitRegisters();              // Initialize user-level CPU registers,
52                                       // before jumping to user code
53    bool pageTableLoaded;
54
```

Changed Text

/code/userprog/addrspace.h



Original Text Changed Text

/code/userprog/addrspace.cc

主要更改在 Load function.

1. 主要執行程式碼會 allocated to main memory，相關 page table 紀錄.
2. 如果 frame number in main memory 足夠，the pages 會存在 main memory.；如果不夠，最後一個 pages 會寫入 virtual memory by Write Sector.

相關 page table 會被記錄並設定為 invalid.

```
                // page is referenced or modified.
    bool dirty;         // This bit is set by the hardware every time the
                // page is modified.
```

Original Text

```
                // page is referenced or modified.
    bool dirty;         // This bit is set by the hardware every time the
                // page is modified.
    int ID;

    int LRU_times;    //for Least Recently used algorithm

```

Changed Text

/code/machine/translate.h

Add ID and LRU_times in class TranslationEntry

```
    int i;
    unsigned int vpn, offset;
    TranslationEntry *entry;
    unsigned int pageFrame;

            DEBUG(dbgAddr, "Invalid virtual page # " << virtAddr);
            return PageFaultException;
```

Original Text

```
    int i, j;
    unsigned int vpn, offset;
    TranslationEntry *entry;
    unsigned int pageFrame;

    int Swap_out_page;///find the page Swap_out_page

            printf("Page fault Happen!\n");
            kernel->stats->numPageFaults += 1; // nachos pagefault counter +1
            j = 0;
            while (kernel->machine->Occupied_frame[j] != FALSE && j < NumPhysPages)
                j += 1; // find valid frame space

            if (j < NumPhysPages) { // if find valid frame space, save the page in virtua
l memory into physical memory
                char* buffer; //save page temporary
                buffer = new char[PageSize];
                pageTable[vpn].physicalPage = j; // save physical memory position
                pageTable[vpn].valid = TRUE; // page has already in physical memory
                kernel->machine->Occupied_frame[j] = TRUE;
                kernel->machine->FrameName[j] = pageTable[vpn].ID;
                kernel->machine->main_tab[j] = &pageTable[vpn]; // save the page pointer

                pageTable[vpn].LRU_times++; //for LRU

                kernel->Swap_Area->ReadSector(pageTable[vpn].virtualPage, buffer); // rea
d data from swap area
                bcopy(buffer, &mainMemory[j * PageSize], PageSize); // save data into phy
sical memory
            }
            else {
                char* buffer1;
                char* buffer2;
                buffer1 = new char[PageSize];
                buffer2 = new char[PageSize];

                //Random
                //Swap_out_page = (rand()%32);

                //LRU

                int min = pageTable[0].LRU_times;
                Swap_out_page = 0;
                for (int cc = 0; cc < 32; cc++) {
                    if (min > pageTable[cc].LRU_times) {
                        min = pageTable[cc].LRU_times;
                        Swap_out_page = cc;
                    }
                }
                pageTable[Swap_out_page].LRU_times++;

                printf("Page%d swap out!\n", Swap_out_page);
                bcopy(&mainMemory[Swap_out_page * PageSize], buffer1, PageSize);
                kernel->Swap_Area->ReadSector(pageTable[vpn].virtualPage, buffer2);
                bcopy(buffer2, &mainMemory[Swap_out_page * PageSize], PageSize);
                kernel->Swap_Area->WriteSector(pageTable[vpn].virtualPage, buffer1);

                main_tab[Swap_out_page]->virtualPage = pageTable[vpn].virtualPage;
                main_tab[Swap_out_page]->valid = FALSE;

                pageTable[vpn].valid = TRUE;
                pageTable[vpn].physicalPage = Swap_out_page;
                kernel->machine->FrameName[Swap_out_page] = pageTable[vpn].ID;
                main_tab[Swap_out_page] = &pageTable[vpn];
                printf("Finish the page replcement!\n");
            }
```

Changed Text

/code/machine/translate.cc

LRU implement.

將 page table 的 valid bit 以確保 page 的位置，i.e. in main memory or in virtual memory.

如果 page 在 virtual memory，檢查 main memory 中現在是否有可用空間。

如果 main memory 沒有空間，將發生 page 替換。

2 buffers 拿來宣告 swap in/out.。LRU，使用 for loop 來查找 least used page

靠 implied by bcopy, ReadSector, and WriteSector.完成

讀取 virtual memory， 並複製到 main memory，選擇 the swap out page 到 virtual memory

```
#define PCReg           34      // Current program counter
#define NextPCReg       35      // Next program counter (for branch delay)
#define PrevPCReg       36      // Previous program counter (for debugging)
```

Original Text

```
#define PCReg           34      // Current program LRU_timeser
#define NextPCReg       35      // Next program LRU_timeser (for branch delay)
#define PrevPCReg       36      // Previous program LRU_timeser (for debugging)
    int  Identity;
    int SectorNum;//record sector number
    int FrameName[NumPhysPages];
    bool Occupied_frame[NumPhysPages];//record which frame in the main memory is occupie
d.
    bool Occupied_virpage[NumPhysPages];

    // start for page replacement //
    int LRU_times[NumPhysPages]; //for LRU
    bool reference_bit[NumPhysPages];//for second chance algorithm.
    // end //

    TranslationEntry *main_tab[NumPhysPages];
```

Changed Text

/code/machine/machine.h

紀錄 sector number, frame paged, 和被 main memory, virtual memory 占用的 frame.

III. Result

Experiment result and some discussion

1. 以 Random replacement 方式同時執行/test/matmult.c 與/test/sort.c

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

```
r08921005@r08921005-VirtualBox:~/nachos/3/nachos-4.0/code/userprog$ ./nachos -e ../test/matmult -e ../test/sort
Total threads number is 2
Thread ../test/matmult is executing.
Thread ../test/sort is executing.
Page fault Happen!
Page0 swap out!
Finish the page replcement!
Page fault Happen!
Page1 swap out!
Finish the page replcement!
Page fault Happen!
Page2 swap out!
```

…

```
Finish the page replcement!
Page fault Happen!
Page12 swap out!
Finish the page replcement!
Page fault Happen!
Page13 swap out!
Finish the page replcement!
Page fault Happen!
Page14 swap out!
Finish the page replcement!
Page fault Happen!
Page15 swap out!
Finish the page replcement!
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 7691030, idle 1365646, system 6325380, user 4
Disk I/O: reads 80, writes 102
Console I/O: reads 0, writes 0
Paging: faults 80
Network I/O: packets received 0, sent 0
r08921005@r08921005-VirtualBox:~/nachos/3/nachos-4.0/code/userprog$
```

```
Page2 swap out!
Finish the page replcement!
Page fault Happen!
Page3 swap out!
Finish the page replcement!
return value:1023
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 249484530, idle 48484195, system 201000330, user 5
Disk I/O: reads 3508, writes 3576
Console I/O: reads 0, writes 0
Paging: faults 3508
Network I/O: packets received 0, sent 0
r08921005@r08921005-VirtualBox:~/nachos/3/nachos-4.0/code/userprog$
```

Fig. 2 修改後，由於 virtual address 與 physical address 的維護加上 random replacement 的方式，可以成功執行，並輸出 7220 (matmult) 與 1023(sort)

Conclusion

Page Fault 發生的次數和排程的演算法關係非常的大，這次先使用隨機的方式找到 victim，之後可以再進一步的測試在課堂上有交到過的各種演算法，來測試不同演算法發生 page fault 的次數，來決定最佳的排成方式，找到一個最好的結果。