HW2_report

B10632026 吳苡瑄

B10632014 陳品羚

作業要求:

實現 virtual memory 並可以處理 page fault

實作 Page Replacement Algorithm 的 FIFO 與 LRU

切換演算法:

在跑測試程式後加上參數-FIFO 或-LRU 即可切換

大致說明:

在 **userkernel.h** 和 **userkernel.cc** 中加上輔助用的記憶體

```
    // 利用SynchDisk作為輔助記憶體
    SynchDisk *virtualMemoryDisk;
    // 加上vmtype來切換LRU和FIFO
    int vmtype;
```

```
void
UserProgKernel::Initialize()
{
    ThreadedKernel::Initialize();   // init multithreading

    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    // 加上這一行增加輔助記憶體
    virtualMemoryDisk = new SynchDisk("Virtual Memory");
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}
```

在 **/code/machine/machine.h** 中加上會使用到的變數

```
class Machine {
  public:
    bool usedPhyPage[NumPhysPages];     // 紀錄那些physical memory已經被使用了
    bool usedvirPage[NumPhysPages];     // 記錄那些virtual memory已經被使用了
    int  ID_num;                        // 每個thread都有一個專屬ID_num作為識別用的
    int PhyPageName[NumPhysPages];
    int count[NumPhysPages];            // LRU用的counter

    TranslationEntry *main_tab[NumPhysPages];
    static int fifo;
```

/code/userprog/addrspace.cc

在 Load 這裡開一個專屬這個 thread 的 page table，在 load 時一直往下找記憶體直到找到沒被用到的或是到底為止，以此來判斷 memory 夠不夠用，若不夠就需要用到輔助記憶體，還需要存好一些重要參數以便要用的時候找得到。

```
//////////////////////////////////////////////////////////////////////
    // 先創造一個新的page table然後把裡面的內容初始化
    pageTable = new TranslationEntry[numPages];
    for(unsigned int i = 0, j = 0; i < numPages; i++) {
        pageTable[i].virtualPage = i;
        pageTable[i].physicalPage = i;
        pageTable[i].valid = true;
        pageTable[i].use = false;
        pageTable[i].dirty = false;
        pageTable[i].readOnly = false;
        pageTable[i].count = 0;
    }
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
    if (noffH.code.size > 0) {
        for(unsigned int j=0,i=0;i < numPages ;i++) {
            j=0;
            // 一直往下找記憶體直到找到沒被用到的或是到底為止
            while(kernel->machine->usedPhyPage[j] != false && j < NumPhysPages){ j++; }

            // 若找到的是沒被用過的代表記憶體足夠，就直接用
            if(j < NumPhysPages) {
                kernel->machine->usedPhyPage[j] = true;
                kernel->machine->PhyPageName[j] = ID;
                kernel->machine->main_tab[j] = &pageTable[i];
                pageTable[i].physicalPage = j;
                pageTable[i].valid = true;
                pageTable[i].use = false;
                pageTable[i].dirty = false;
                pageTable[i].readOnly = false;
                pageTable[i].ID = ID;
                //printf("%d",pageTable[i].count);
                pageTable[i].count++;       // 如果用了counter就+1
                executable->ReadAt( &(kernel->machine->mainMemory[j * PageSize]), PageSize,
                    noffH.code.inFileAddr + (i*PageSize));
            }
            // 若找到底了代表記憶體不夠，需要用到virtual memory了
            else {
                char *buffer;
                buffer = new char[PageSize];
                j = 0;
                // 往下找到沒被用過的virtual page
                while(kernel->machine->usedvirPage[j] != false){ j++; }

                kernel->machine->usedvirPage[j]=true;
                pageTable[i].virtualPage = j;       // 存你用的virtual page是哪一個
                pageTable[i].valid = false;         // 代表他不在真的main memory中
                pageTable[i].use = false;
                pageTable[i].dirty = false;
                pageTable[i].readOnly = false;
                pageTable[i].ID = ID;
                executable->ReadAt(buffer, PageSize, noffH.code.inFileAddr + (i * PageSize));
                kernel->virtualMemoryDisk->WriteSector(j, buffer);      // 把東西寫到Disk中
            }
        }
    }
//////////////////////////////////////////////////////////////////////
    if (noffH.initData.size > 0) {
        executable->ReadAt(
            &(kernel->machine->mainMemory[noffH.initData.virtualAddr]),
            noffH.initData.size, noffH.initData.inFileAddr);
    }

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

/code/machine/tranlate.cc

主要都是改 `else if (!pageTable[vpn].valid)` 裡面的東西，FIFO 就是一直照順序當 victim，

LRU 則是用一個 count 來計被用到的次數，LRU 用的 count 是用一個 totalcount 來更新的，每次只要 access physical page 就會往上加來更新，並更新被 access 的那個 page 的 count，在找 victim page 時，選擇 count 最少的，也就是最久沒被 access 過的。找完 victim 之後再去做 swap in swap out 的動作。

```
else if (!pageTable[vpn].valid) {
    printf("page fault\n");
    kernel->stats->numPageFaults++;
    j = 0;
    while(kernel->machine->usedPhyPage[j] != false && j < NumPhysPages){ j++; }
    if (j < NumPhysPages) {
        char *buffer;
        buffer = new char[PageSize];
        kernel->machine->usedPhyPage[j] = true;
        kernel->machine->PhyPageName[j] = pageTable[vpn].ID;

        kernel->machine->main_tab[j] = &pageTable[vpn];
        pageTable[vpn].physicalPage = j;
        pageTable[vpn].valid = true;
        pageTable[vpn].count++; //for LRU

        kernel->virtualMemoryDisk->ReadSector(pageTable[vpn].virtualPage, buffer);
        bcopy(buffer, &mainMemory[j * PageSize], PageSize);
    }
    else {
        char *buffer1;
        buffer1 = new char[PageSize];
        char *buffer2;
        buffer2 = new char[PageSize];

        //FIFO
        if (kernel->vmtype == 0){
            victim = fifo % 32; // 32 is NumPhysPages
        }
```

```
        //LRU
        if (kernel->vmtype == 1) {
            int min = pageTable[0].count;
            victim = 0;
            for(int i = 0; i < 32; i++){
                //printf("%d",pageTable[i].count);
                if(min > pageTable[i].count){
                    min = pageTable[i].count;
                    victim = i;
                }
            }
            pageTable[victim].count++;
        }


        printf("page %d swapped\n",victim);

        // 把東西存到disk裡面
        bcopy(&mainMemory[victim * PageSize], buffer1, PageSize);
        kernel->virtualMemoryDisk->ReadSector(pageTable[vpn].virtualPage, buffer2);
        bcopy(buffer2, &mainMemory[victim*PageSize], PageSize);
        kernel->virtualMemoryDisk->WriteSector(pageTable[vpn].virtualPage, buffer1);

        main_tab[victim]->virtualPag e= pageTable[vpn].virtualPage;
        main_tab[victim]->valid = false;

        // 把東西load到main memory中
        pageTable[vpn].valid = true;
        pageTable[vpn].physicalPage = victim;
        kernel->machine->PhyPageName[victim] = pageTable[vpn].ID;
        main_tab[victim] = &pageTable[vpn];
        fifo ++;
    }
}
```

測試結果:

直接用之前的 test1 和 test2 來測試。

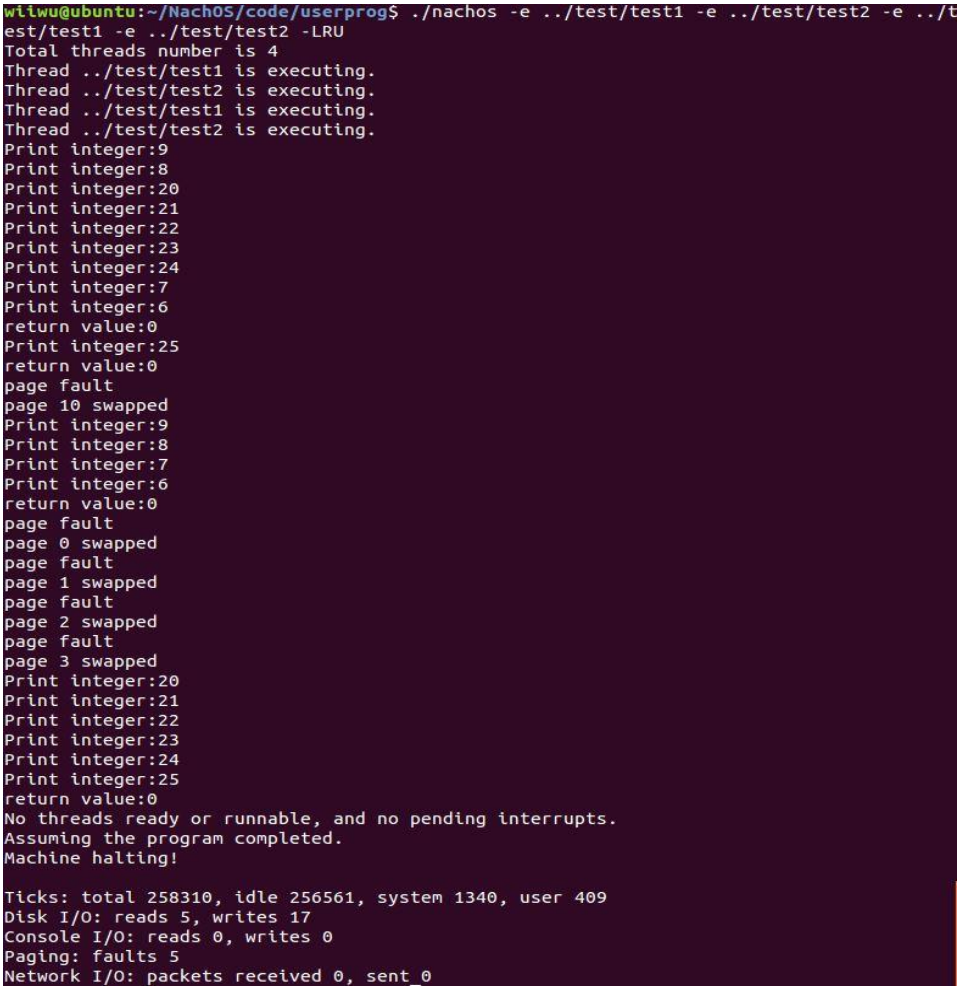
```
wiiwu@ubuntu:~/NachOS/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2 -e ../t
est/test1 -e ../test/test2
using RR
Total threads number is 4
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Unexpected user mode exception4
Assertion failed: line 100 file ../userprog/exception.cc
Aborted (core dumped)
```

沒有 page fault 時出錯的樣子

```
wiiwu@ubuntu:~/NachOS/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2 -e ../t
est/test1 -e ../test/test2 -FIFO
Total threads number is 4
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:7
Print integer:6
return value:0
Print integer:25
return value:0
page fault
page 0 swapped
Print integer:9
Print integer:8
Print integer:7
Print integer:6
return value:0
page fault
page 1 swapped
page fault
page 2 swapped
page fault
page 3 swapped
page fault
page 4 swapped
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 258310, idle 256561, system 1340, user 409
Disk I/O: reads 5, writes 17
Console I/O: reads 0, writes 0
Paging: faults 5
Network I/O: packets received 0, sent 0
```

FIFO 換頁順序 0->1->2->3->4



LRU 換頁順序 10->0->1->2->3

使用指令:

進到 **/code/userprog** 中

./nachos -e ../test/test1 -e ../test/test2 -e ../test/test1 -e ../test/test2 -FIFO

./nachos -e ../test/test1 -e ../test/test2 -e ../test/test1 -e ../test/test2 -LRU