

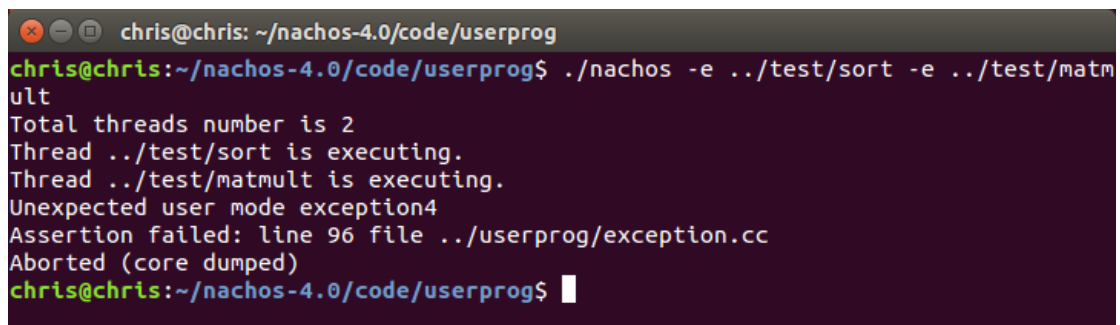
OS Project 3: Memory Management

Student ID: R06921095

Name: Jui-Che Wu

➤ Motivation

以下的圖表為同時執行 `sort` 以及 `matmult` 的執行結果，適用原本的 `main memory` 設定會造成記憶體不足的狀況，即使透過更改 `memory size` 來執行，硬體仍有其限制。此時可以透過於你記憶體的管理，透過 `page replacement algorithm` 去挑選 `swap out` 的 `page`，讓城市有擴充記憶體的假象，來解決記憶體不足的情形。此次的 `project` 就是要在 `nachos` 上完成虛擬記憶體的管理，讓 `sort` 以及 `matmult` 兩隻城市能夠同時執行。



```
chris@chris: ~/nachos-4.0/code/userprog
chris@chris:~/nachos-4.0/code/userprog$ ./nachos -e ../test/sort -e ../test/matmult
Total threads number is 2
Thread ../test/sort is executing.
Thread ../test/matmult is executing.
Unexpected user mode exception4
Assertion failed: line 96 file ../userprog/exception.cc
Aborted (core dumped)
chris@chris:~/nachos-4.0/code/userprog$
```

➤ Implementation

首先在 `userkernel.h` 的檔案中，先新增一個料型態為 `SynchDisk` 的物件 `Swap_Area`，創造此物件的目的為創造一個硬碟區域來儲存沒辦法進入主要記憶體的 `page`，讓這些 `page` 存進此 `swap area`。

```

#ifndef USERKERNEL_H
#define USERKERNEL_H

#include "kernel.h"
#include "fileSYS.h"
#include "machine.h"
#include "synchdisk.h"
class SynchDisk;
class UserProgKernel : public ThreadedKernel {
public:
    UserProgKernel(int argc, char **argv);
    ~UserProgKernel(); // deallocate the kernel

    void Initialize(); // initialize the kernel

    void Run(); // do kernel stuff

    void SelfTest(); // test whether kernel is working

    SynchDisk *Swap_Area; // create swap area for virtual memory
    // These are public for notational convenience.
    Machine *machine;
    FileSystem *fileSystem;
    bool debugUserProg;
#ifdef FILESYS
    SynchDisk *synchDisk;
#endif // FILESYS

private:
    Thread* t[10]; // single step user program

    char* execfile[10];
    int execfileNum;
};

```

接著在 userkernel.cc 的檔案中的初始化中動態分配 swap area 的記憶體空間。

```

//-----
// UserProgKernel::Initialize
// Initialize Nachos global data structures.
//-----
void
UserProgKernel::Initialize()
{
    ThreadedKernel::Initialize(); // init multithreading

    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    Swap_Area = new SynchDisk("New Disk for Swap Area");//Create swap area for virtual memory
#ifdef FILESYS
    synchDisk = new SynchDisk("New SynchDisk");
#endif // FILESYS
}

```

接著在 machine.cc 的檔案中，新增幾個成員函數來記錄主要記憶體的 frame 的使用狀況，並同時紀錄 ID, Sector number, main memory 地區的使用狀況等...。之後

再做 page replacement 才有尋找依據。

```
TranslationEntry *tlb; // this pointer should be considered
                        // "read-only" to Nachos kernel code

TranslationEntry *pageTable;
unsigned int pageTableSize;
bool ReadMem(int addr, int size, int* value);
int Identity;
int SectorNum; // record sector number
int FrameName[NumPhysPages];
bool Occupied_frame[NumPhysPages]; // record which frame in the main memory is occupied.
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];

private:
```

接著在 addrspace.cc 的檔案中，需要將幾行 comment 掉，否則若 page 的總數大於實體的 main memory 的 page 數，作業系統就會將其停止。

```
bool
AddrSpace::Load(char *fileName)
{
    OpenFile *executable = kernel->fileSystem->Open(fileName);
    NoffHeader noffH;

    unsigned int size, tmp;

    if (executable == NULL) {
        cerr << "Unable to open file " << fileName << "\n";
        return FALSE;
    }
    executable->ReadAt((char *)&noffH, sizeof(noffH), 0);
    if ((noffH.noffMagic != NOFFMAGIC) &&
        (WordToHost(noffH.noffMagic) == NOFFMAGIC))
        SwapHeader(&noffH);
    ASSERT(noffH.noffMagic == NOFFMAGIC);

    // how big is address space?
    size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
          + UserStackSize; // we need to increase the size
                          // to leave room for the stack
    numPages = divRoundUp(size, PageSize);
    // cout << "number of pages of " << fileName << " is " << numPages << endl;

    pageTable = new TranslationEntry[numPages];

    size = numPages * PageSize;

    // ASSERT(numPages <= NumPhysPages); // check we're not trying
    // to run anything too big --
    // at least until we have
    // virtual memory

    // DEBUG(dbgAddr, "Initializing address space: " << numPages << ", " << size);

    // then, copy in the code and data segments into memory
```

接著一樣在 `addrspace.cc` 的檔案中，將欲執行的程式分配到 `main memory` 內並記錄對應的 `page table`。如果 `main memory` 的 `frame` 數不夠時，將 `page` 存到虛擬記憶體中並對應 `page table` 並在 `page table` 中標記成 `invalid`。比較需要的是函數 `ReadAt` 來紀錄檔案儲存的位置、檔案大小、開始讀取的 `offset`。以及 `WriteSector` 來把 `page` 寫入虛擬記憶體。

```
if (noFFH.code.size > 0) {
    // DEBUG(dbgAddr, "Initializing code segment.");
    // DEBUG(dbgAddr, noFFH.code.virtualAddr << " ", " << noFFH.code.size);

    for(int j=0,i=0;i < numPages ;i++){
        j=0;
        while(kernel->machine->Occupied_frame[j] != FALSE && j < NumPhysPages)
            j += 1;

        //if memory is enough, just put data in without using virtual memory
        if(j<NumPhysPages){
            pageTable[i].physicalPage = j;
            pageTable[i].use = FALSE;
            pageTable[i].dirty = FALSE;
            pageTable[i].ID =ID;
            pageTable[i].readOnly = FALSE;
            pageTable[i].valid = TRUE;
            kernel->machine->Occupied_frame[j]=TRUE;
            kernel->machine->FrameName[j]=ID;
            kernel->machine->main_tab[j]=&pageTable[i];
            pageTable[i].LRU_times++;
            executable->ReadAt(&(kernel->machine->mainMemory[j*PageSize]),PageSize, noFFH.code.inFileAddr+(i*PageSize));
        }
        //Use virtual memory technique
        else{
            char *buffer;
            buffer = new char[PageSize];
            tmp=0;
            while(kernel->machine->Occupied_virpage[tmp]!=FALSE){tmp++;}
            pageTable[i].virtualPage=tmp;
            pageTable[i].ID =ID;
            pageTable[i].valid = FALSE;
            pageTable[i].dirty = FALSE;
            pageTable[i].readOnly = FALSE;
            pageTable[i].use = FALSE;
            kernel->machine->Occupied_virpage[tmp]=true;
            executable->ReadAt(buffer,PageSize, noFFH.code.inFileAddr+(i*PageSize));
            kernel->Swap_Area->WriteSector(tmp, buffer); //call virtual_disk write in virtual memory
        }
    }
}
```

接著一樣在 `addrspace.cc` 的檔案中，確認 `Loading` 檔案是否發生錯誤，接著更改 `SaveState` 函數，以便之後在情境轉換時把 `ProcessState` 儲存起來。

```
void
AddrSpace::Execute(char *fileName)
{
    Is_ptable_loaded=FALSE;
    if (!Load(fileName)) {
        cout << "inside !Load(fileName)" << endl;
        return; // executable not found
    }

    //kernel->currentThread->space = this;
    this->InitRegisters(); // set the initial register values
    this->RestoreState(); // load page table register
    Is_ptable_loaded=TRUE;
    kernel->machine->Run(); // jump to the user program

    ASSERTNOTREACHED(); // machine->Run never returns;
    // the address space exits
    // by doing the syscall "exit"
}
```

接著是實作 Least Recently Used 的部分，另外也時做了 Random choose。
在 translate.h 中，利用 counter 去記錄哪個 page 在主要記憶體中被使用到最少次。

```
#ifndef TLB_H
#define TLB_H

#include "copyright.h"
#include "utility.h"

// The following class defines an entry in a translation table -- either
// in a page table or a TLB. Each entry defines a mapping from one
// virtual page to one physical page.
// In addition, there are some extra bits for access control (valid and
// read-only) and some bits for usage information (use and dirty).

class TranslationEntry {
public:
    unsigned int virtualPage; // The page number in virtual memory.
    unsigned int physicalPage; // The page number in real memory (relative to the
        // start of "mainMemory"
    bool valid; // If this bit is set, the translation is ignored.
        // (In other words, the entry hasn't been initialized.)
    bool readOnly; // If this bit is set, the user program is not allowed
        // to modify the contents of the page.
    bool use; // This bit is set by the hardware every time the
        // 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
};

#endif
```

最後，在 translate.cc 中，實作 LRU 的部分

```
else if (!pageTable[vpn].valid) {

    printf("Page fault Happen!\n");
    kernel->stats->numPageFaults += 1;
    j=0;
    while(kernel->machine->occupied_frame[j]!=FALSE&&j<NumPhysPages)
        j += 1;

    if( j < NumPhysPages){
        char *buffer; //save page temporary
        buffer = new char[PageSize];
        pageTable[vpn].physicalPage = j;
        pageTable[vpn].valid = TRUE;
        kernel->machine->occupied_frame[j]=TRUE;
        kernel->machine->FrameName[j]=pageTable[vpn].ID;
        kernel->machine->main_tab[j]=&pageTable[vpn];

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

        kernel->Swap_Area->ReadSector(pageTable[vpn].virtualPage, buffer);
        bcopy(buffer,&mainMemory[j*PageSize],PageSize);
    }
}
```

```

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("page replcement finished!\n");
}

```

➤ Result

再來是實驗結果的部分，以下為執行的指令：同時執行 sort 以及 matmult。
由於 output 過多，所以將其重導到 result.txt 中

```

chris@chris:~/Desktop/OS_Nachos_Project/Project3/nachos-4.0/code/userprog$ ./nachos -e ../test/sort -e ../test/matmult > ~/Desktop/result.txt

```

從執行結果可以看出程式使用 sqap in/out 來處理 page fault。以下為 sort 的執行結果，return 值為 0 (正確)。

```
Total threads number is 2
Thread ../test/sort is executing.
Thread ../test/matmult is executing.
Page fault Happen!
Page0 swap out!
page replcement finished!
Page fault Happen!
Page1 swap out!
page replcement finished!
Page fault Happen!
Page2 swap out!
page replcement finished!
Page fault Happen!
Page3 swap out!
page replcement finished!
Page fault Happen!
Page4 swap out!
page replcement finished!
Page fault Happen!
Page5 swap out!
```

```
Page28 swap out!
page replcement finished!
Page fault Happen!
Page29 swap out!
page replcement finished!
Page fault Happen!
Page30 swap out!
page replcement finished!
Page fault Happen!
Page31 swap out!
page replcement finished!
return value:0
Page fault Happen!
Page0 swap out!
page replcement finished!
```

再來是 matmult 的執行結果，return 值為 7220 (正確)。

```
Page9 swap out!
page replcement finished!
Page fault Happen!
Page10 swap out!
page replcement finished!
Page fault Happen!
Page11 swap out!
page replcement finished!
Page fault Happen!
Page12 swap out!
page replcement finished!
return value:7220
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 449820030, idle 55104255, system 394715770, user 5
Disk I/O: reads 5645, writes 5713
Console I/O: reads 0, writes 0
Paging: faults 5645
Network I/O: packets received 0, sent 0
```

由結果可見，虛擬記憶體的技术可以用來解決 physically 上的限制，RLU 演算法雖然產生了一些額外的問題，但是也可以有效降低 page fault rate。