OS project 1 r12631055 林東甫

Part1: Multi-Programming

test1 (source code and execution result)

執行結果為 9 ~ 6 依序輸出

```
#include "syscall.h"
main()
        {
                int     n;
                for (n=9;n>5;n--)
                        PrintInt(n);
        }
```

```
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$ ./nachos -e ..
/test/test1
Total threads number is 1
Thread ../test/test1 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:6
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 200, idle 66, system 40, user 94
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$
```

test2 (source code and execution result)

執行結果為 20 ~ 25 依序輸出

```
#include "syscall.h"

main()
        {
                int     n;
                for (n=20;n<=25;n++)
                        PrintInt(n);
        }
```

```
Network I/O: packets received 0, sent 0
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$ ./nachos -e ..
/test/test2
Total threads number is 1
Thread ../test/test2 is executing.
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 200, idle 32, system 40, user 128
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$
```

executing test1 and test2 simultaneously, the result seems weird. The output is not consistent with source code.

同時執行兩程式,結果並未按照原先方式輸出

```
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$ ./nachos -e ..
/test/test1 -e ../test/test2
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:6
Print integer:7
Print integer:8
Print integer:9
Print integer:10
Print integer:12
Print integer:13
Print integer:14
Print integer:15
Print integer:16
Print integer:16
Print integer:17
Print integer:18
Print integer:19
Print integer:20
Print integer:17
Print integer:18
```

可以觀察到執行到interger:6時，程式開始出錯，開始遞增數列，粗估推測會導致程式執行結果而與預期不同的原因可能是：Nachos系統的預設並沒有對多個程式做記憶體管理，會將全部的實體記憶體分頁都分配出去，因此當多個執行緒在執行時，兩個程式會被分到同一個page，導致執行時出現錯誤。雖然有開啟虛擬記憶體，但是基本上沒有作為，所以當多程式同時執行時就會重疊到其他程式正在使用的page ，然後發生錯誤，為此我們要修改addrspace，使程式的虛擬記憶體映射到沒有人使用的實體記憶體，而不是互相糾纏。

要讓程式的虛擬記憶體能map到實體記憶體上，首先第一步先修改addrspace.h
在addrspace.h檔中，多增加一行變數
static bool usedPhysicalPage[NumPhysPages]，用來記錄page的使用情形，確認當下有哪些page是可以使用的。

```
class AddrSpace {
  public:
    AddrSpace();                        // Create an address space.
    ~AddrSpace();                       // De-allocate an address space

    void Execute(char *fileName);       // Run the the program
                                        // stored in the file "executable"

    void SaveState();                   // Save/restore address space-specific
    void RestoreState();                // info on a context switch
    static bool usedPhysicalPage[NumPhysPages];
  private:
    TranslationEntry *pageTable;        // Assume linear page table translation
                                        // for now!
    unsigned int numPages;              // Number of pages in the virtual
                                        // address space

    bool Load(char *fileName);          // Load the program into memory
                                        // return false if not found

    void InitRegisters();               // Initialize user-level CPU registers,
                                        // before jumping to user code
```

然後在addrspace.cc檔中初始化宣告usedPhysicalPage：

```
// of liability and disclaimer of warranty provisions.

#include "copyright.h"
#include "main.h"
#include "addrspace.h"
#include "machine.h"
#include "noff.h"

bool AddrSpace::usedPhysicalPage[NumPhysPages] = {0};
//----------------------------------------------------------------------
// SwapHeader
//      Do little endian to big endian conversion on the bytes in the
//      object file header, in case the file was generated on a little
//      endian machine, and we're now running on a big endian machine.
//----------------------------------------------------------------------

static void
SwapHeader (NoffHeader *noffH)
{
    noffH->noffMagic = WordToHost(noffH->noffMagic);
```

在addrspace.cc中的AddrSpace::AddrSapce()函式，將函式中的所有程式做註解，因為進行多執行緒時，只需足夠大小的記憶體即可，無須分配實際記憶體的大小，之後在一併把映射記憶體這件事挪到 AddrSpace::Load 中

```
AddrSpace::AddrSpace()
{
    //pageTable = new TranslationEntry[NumPhysPages];
    //for (unsigned int i = 0; i < NumPhysPages; i++) {
        //pageTable[i].virtualPage = i; // for now, virt page # = phys page #
        //pageTable[i].physicalPage = i;
//      pageTable[i].physicalPage = 0;
        //pageTable[i].valid = TRUE;
//      pageTable[i].valid = FALSE;
        //pageTable[i].use = FALSE;
        //pageTable[i].dirty = FALSE;
        //pageTable[i].readOnly = FALSE;
    //}

    // zero out the entire address space
//    bzero(kernel->machine->mainMemory, MemorySize);
}
```

在解構子AddrSpace::~AddrSapce()函式中，用for迴圈把所有有用到的Page都改回false，釋放資源以供之後的程式使用。

```
//----------------------------------------------------------------------
// AddrSpace::~AddrSpace
//      Dealloate an address space.
//----------------------------------------------------------------------

AddrSpace::~AddrSpace()
{
   for(unsigned int i = 0; i < numPages; i++)
        AddrSpace::usedPhysicalPage[pageTable[i].physicalPage]=FALSE;
   delete pageTable;
}
```

在AddrSpace::Load()函式中，把原本於AddrSpace::AddrSapce()註解掉的程式移至此並稍作修改。透過while迴圈查看每個physical page是否已被使用，若已被使用(值為true)則看下一頁，直到找到值為false的page，將其設為true並將資料存至該頁。

```
    numPages = divRoundUp(size, PageSize);
//      cout << "number of pages of " << fileName<< " is "<<numPages<<endl;

    pageTable = new TranslationEntry[NumPhysPages];
    for (unsigned int i = 0; i < NumPhysPages; i++) {
        pageTable[i].virtualPage = i;   // for now, virt page # = phys page #
        while(j<NumPhysPages && AddrSpace::usedPhysicalPage[j]==true){
            j++;
        }
        AddrSpace::usedPhysicalPage[j]=TRUE;
        PageTable[i].physicalPage = j;
        //pageTable[i].physicalPage = i;
//      pageTable[i].physicalPage = 0;
        pageTable[i].valid = TRUE;
//      pageTable[i].valid = FALSE;
        pageTable[i].use = FALSE;
        pageTable[i].dirty = FALSE;
        pageTable[i].readOnly = FALSE;
        }
    size = numPages * PageSize;
```

最後是在AddrSpace::Load()函式中，因為要找出map後的位置，因此更改程式碼中兩個executable->ReadAt的部分，使virtualAddr先除以PageSize，以此得知使用的是第幾個page，再透過pageTable找到所對應實體頁的頁數，並乘上每個page的大小得到該頁的physical memory，接著再加上virtualAddr除以PageSize的餘數值，以得到page內的offset，如此即得到相對應的physical address。

```
// then, copy in the code and data segments into memory
        if (noffH.code.size > 0) {
        DEBUG(dbgAddr, "Initializing code segment.");
        DEBUG(dbgAddr, noffH.code.virtualAddr << ", " << noffH.code.size);
                executable->ReadAt(
                &(kernel->machine->mainMemory[pageTable[noffH.code.virtualAddr/
PageSize].physicalPage*PageSize+(noffH.code.virtualAddr%PageSize)]),
noffH.code.size, noffH.code.inFileAddr);
    }
        if (noffH.initData.size > 0) {
        DEBUG(dbgAddr, "Initializing data segment.");
        DEBUG(dbgAddr, noffH.initData.virtualAddr << ", " <<
noffH.initData.size);
        executable->ReadAt(
                &(kernel->machine->mainMemory[pageTable[noffH.code.virtualAddr/
PageSize].physicalPage*PageSize+(noffH.code.virtualAddr%PageSize)]),
noffH.code.size, noffH.code.inFileAddr);
    }

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

修改之後，再同時執行test1與test2，即可發現程式已能正常執行。

```
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
Print integer:6
return value:0
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 300, idle 8, system 70, user 222
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

心得：光是在建置環境，研究文章跟trace code就花了不少時間。

Part2: System call tracing

```
r12631055@r12631055-VirtualBox:~/Nachos/nachos-4.0/code/userprog$ gdb ./nachos -
q
Reading symbols from ./nachos...done.
(gdb) b Interrupt::Halt
Breakpoint 1 at 0x804d3a5: file ../machine/interrupt.cc, line 236.
(gdb) r -e ../test/halt
Starting program: /home/r12631055/Nachos/nachos-4.0/code/userprog/nachos -e ../t
est/halt
Total threads number is 1
Thread ../test/halt is executing.

Breakpoint 1, Interrupt::Halt (this=0x8067ad0) at ../machine/interrupt.cc:236
236         cout << "Machine halting!\n\n";
(gdb) 
```

使用gdb來進行System call tracing

```
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x0804d3a5 in Interrupt::Halt()
                                           at ../machine/interrupt.cc:236
        breakpoint already hit 1 time
(gdb) list 236
231     //      Shut down Nachos cleanly, printing out performance statistics.
232     //----------------------------------------------------------------------
233     void
234     Interrupt::Halt()
235     {
236         cout << "Machine halting!\n\n";
237         kernel->stats->Print();
238         delete kernel;      // Never returns.
239     }
240
```

觀察內部的system code

Show the current state of call stack.

```
(gdb) bt
#0  Interrupt::Halt (this=0x8067ad0) at ../machine/interrupt.cc:236
#1  0x08054e88 in ExceptionHandler (which=SyscallException)
    at ../userprog/exception.cc:62
#2  0x08055cb7 in Machine::RaiseException (this=0x8067d58,
    which=SyscallException, badVAddr=0) at ../machine/machine.cc:109
#3  0x08057e1a in Machine::OneInstruction (this=0x8067d58, instr=0x806d5c0)
    at ../machine/mipssim.cc:558
#4  0x0805632c in Machine::Run (this=0x8067d58) at ../machine/mipssim.cc:62
#5  0x08054b56 in AddrSpace::Execute (this=0x8069420,
    fileName=0xbffff260 "../test/halt") at ../userprog/addrspace.cc:165
#6  0x080590cc in ForkExecute (t=0x8069238) at ../userprog/userkernel.cc:88
#7  0x0805a12e in ThreadRoot ()
#8  0x0805a126 in ?? ()
(gdb) 
```

以下分別來討論SC_Halt各個功能:

SC_Halt

- Machine/mipssims.cc
- 模擬 MIPS P2/3000 processor

Machine:Run()
// 模擬執行 program in thread
// 切換到 UserMode
Machine::OneInstruction()
// 定義執行一個 instruction 所需的參數與步驟
// 例如：一個 byte 大小, rs, rt, rd 等 register
// 判斷 opCode 並執行對應的 instruction

- Machine/machine.cc
- 判斷 host 的 位元組順序 big-endian 或 little-endian

Machine::RaiseException()
// 當 interrupt 或是 exception 發生時，紀錄 program 當前位置
// 設定 User Mode -> Kernel Mode
// 呼叫 Exception Handler
// 設定 Kernel Mode -> User Mode

- userprog/exception.cc

ExceptionHandler()
// entry point
// 定義 exception 發生時所對應的指令
// syscall (interrupt) 或 exception 發生時會呼叫 (Halt 為其中一種 syscall)
// 後續呼叫 SysHalt()

- machine/interrupt.cc

Interrupt::Halt()
// Show halt 的訊息
// delete kernel <- 這個動作?

Reference

https://morris821028.github.io/2014/05/24/lesson/hw-nachos4/

http://blog.terrynini.tw/tw/OS-NachOS-HW1/

https://www.twblogs.net/a/5b88b67e2b71775d1cddf529

http://blog.terrynini.tw/tw/OS-NachOS-HW1/

https://wiiwu959.github.io/2019/10/10/2019-10-10-OS_HW1-2019/