Operating System

Hw02 MP2: Multi-Programming

Report

Team 20 member :

105060011 蔡悅承

105060016 謝承儒

Team member contribution :

Trace Code : 蔡悅承 謝承儒

Implement : 蔡悅承 謝承儒

Report : 蔡悅承 謝承儒

Trace Code

1. 流程圖

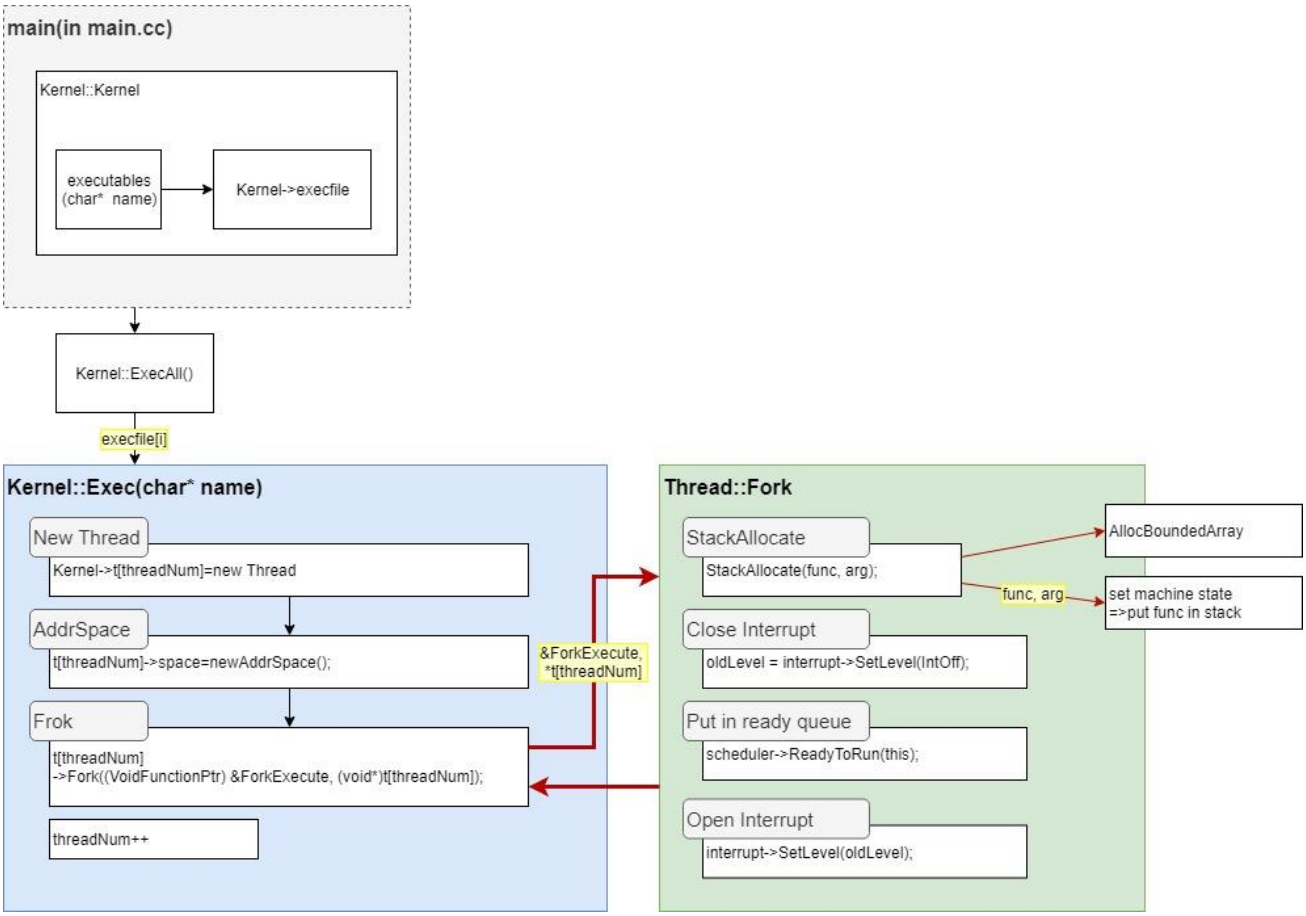

圖 1 Kernel::Exec 流程圖

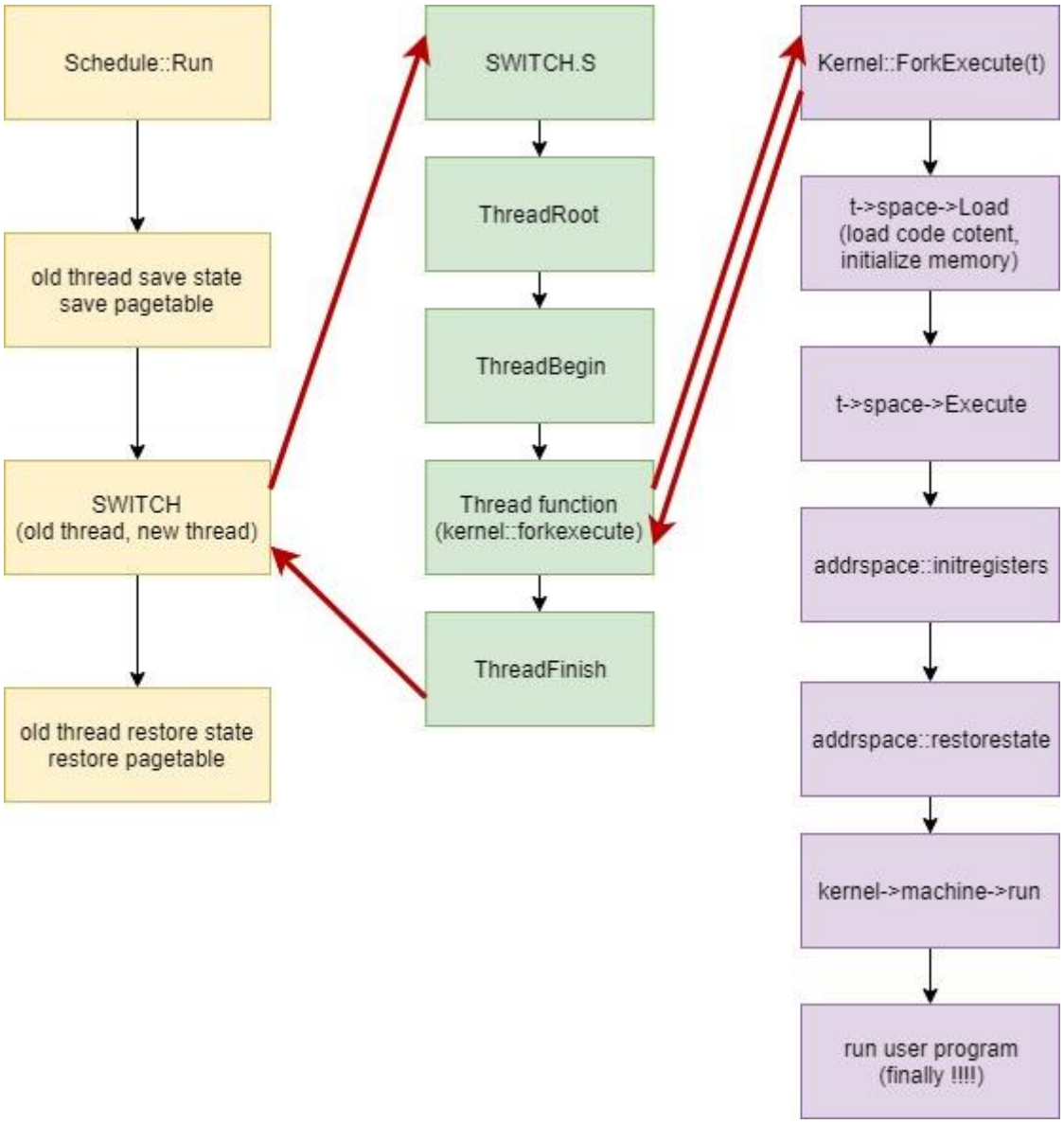

圖 2 Machine::Run 流程圖

2. 步驟

(1) Kernel::Exec

a. 在 Kernel::Kernel 時，將要執行的檔名放進 execfile[]裡。

b. 接著呼叫 Kernel::ExecAll()，會把 execfile 裡的 element 一個一個丟進 Kernel::Exec()

c. 為進來的 execfile[i]，依序做:

(a) new Thread(name, threadNum)
創建新 thread

(b) AddrSpace:: AddrSpace
給予該新 thread address space

(c) Thread::Fork((VoidFunctionPtr) &ForkExecute, (void *)t[threadNum])(接續步驟 d)

(d) threadNum++
計算 thread 的數量，為下一個 thread 做準備

d. 在 Thread::Fork 裡依序做

(a) StackAllocate
給予 stack 相對應的空間，並把 func 放入 stack

(b) 把 interrupt 關閉，避免待會 Schedular::ReadyToRun 被打斷

(c) 將 thread 放入 ready queue

(d) 打開 interruput

(2) Machine::Run

a. Shcedule::Run 開始執行 ready queue 裡的 Theread

b. 發生 Context switch，準備改成執行 new thread

c. 在 Switch.s 裡，會依序呼叫 :

(a) ThreadRoot

(b) ThreadBegin

(c) Thread function，以上面為例，這個就是 Kernel::ForkExcute(t)(接續步驟 d)

(d) ThreadFinish

d. Kernel::ForkExcute 中會執行:

(a) 該 thread 的 addrspace->Load，將該 thread 的東西 load 進 memory

(b) 呼叫 addrspace->Execute，將 Registers 清空、page table 也 load 進來

e. Machine::Run 開始，program 開始執行

f. 當 program 執行結束後，回到步驟 c-(d)，執行 ThreadFinish，將 thread 刪除

g. Context switch 回 old thread，繼續執行 old thread

Questions

1. How Nachos allocates the memory space for new thread(process)?

   Addrspace::Addrspace給了，把它的vail設成true(代表已經y在memory)

2. How Nachos initializes the memory content of a thread(process), including loading the user binary code in the memory?

   Addrspcae::Load裡的ReadAt，利用virtual address轉成physical address後，將它load進kernel->machine->mainMemory

3. How Nachos creates and manages the page table?

   Ceate : Addrspace::Addrspace

   Manage : 每次load new thread就assign全部的memory給它，也就是說，預設只有single thread執行。

4. How Nachos translates address?

   (1) PageNumber = virtual address / PageSize
   (2) Check page number is valid
   (3) FrameNumber = 用Page number在page table查表後的結果
   (4) Check frame number is valid
   (5) PageOffset = virtual address % size
   (6) Physical address = Frame number*PageSize + PageOffset

5. How Nachos initializes the machine status (registers, etc) before running a thread(process)

   Kernel::ForkExec
   =>Addrspace::Execute=>InitialRegsiter=>RestoreState
   =>Machine::Run

6. Which object in Nachos acts the role of process control block

   Thread class

7. When and how does a thread get added into the ReadyToRun queue of Nachos CPU scheduler?

   Thread::Fork=>Scheduler::ReadyToRun

Implement page table in NachOS

0. 修改想法

本來每次thread要allocate addrspace時，會建立自己的page table，同時也會將整個memory初始化，如此memory永遠都只會儲存單個thread的東西。

我們建立Frame table來代表memory，每次thread要allocate addrspace時：

(1) 建立自己的page table

(2) 在frame table(=Memory)找出空位，和page相對應

(3) 當thread要被移除時，把和pages相對應的frame格子，從frame table釋放。

藉由(2)、(3)來做到同時儲存多個thread的資料。

1. 實作內容

修改了Kernel.h、Kernel.cc、Addrspace.cc三個code

(1) Kernel.h

a. 建立Frame table

```
bool FrameTable[NumPhysPages];
```

如果FrameTable[i]是true，代表該frame有儲存page；反之，如果是false，代表該frame是空的，可以被使用。

(2) Kernel.cc

a. Initialize frame table

```
for (int i=0; i<NumPhysPages; i++)
    FrameTable[i] = false; //all frames free
```

在Kernel開啟時，將FrameTable裡所有的entry都設為false，代表memory是全空的。

(3) Addrspace.cc

a. Addrspace():

(a) 建立page table，並初始化page table

b. Load

(a) Page和Frame配對

尋找FrameTable上的空位(=FrameTable[k])，找到後將該page的physicalPage便等於k，同時FrameTable[k]也設為true，表示該位置被使用。

此外把第j個frame所在的memory清除成0。

(b) 讀取code、data、readonlyData

本來是儲存的位置是連續的，但現在是不連續的，所以需要計算自料開始位置在哪，才能往下讀取到，計算範例:

```
code_start_page = pageTable[noffH.code.virtualAddr / PageSize].physicalPage;
code_start_offset = noffH.code.virtualAddr % PageSize;
code_start_physical = code_start_page * PageSize + code_start_offset;
```

start page :

先找出自己的virtual address在哪個page，接著利用pageTable知道在FrameTable的哪一個。

start offset :

算出在自己該page的哪個位置。

最後

$$\text{physical address} = (\text{start page} * \text{Pagesize}) + \text{start offset}$$

(c) ~AddrSpace()

```
for (int i=0; i<NumPhysPages; i++) {
    kernel->FrameTable[pageTable[i].physicalPage] = false;
}
```

把有和該thread的page相連的frame都設成false，代表這些frame是可以再被其他thread使用的。