Content

[I. Motivation 2](#_Toc55396878)

[II. Implementation 3](#_Toc55396879)

[III. Result 9](#_Toc55396880)

Reference

1. Nachos Project1 Report

<https://github.com/taldehyde/os-project-1/blob/master/report.md>

1. OS::NachOS::HW1

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

1. 向 NachOS 4.0 作業進發 (2)

<http://morris821028.github.io/2014/05/30/lesson/hw-nachos4-2/>

# Motivation

Motivation and the problem analysis

Part1: 撰寫自己的 Sleep function，將該 Thread 進入休眠。

Part2: 完成排程處理，在原始的NachOS 4.0中，基礎排程是**Round-robin**。

本次嘗試加入FCFS 先來先服務、SJF 最短工作優先、測試不同排程處理

What’s your plan to deal with the problem

Part1: 定義新的system call code並且宣告**void Sleep(int time)**函數，並新增新的exception來處理SC\_Sleep system call

預計的工作如下：

1. 在/code/userprog/system.h中加入Sleep(int time)
2. 找一下assembly language
3. 增加新的exeption來處理SC\_Sleep system call
4. 由於kernel存有alarm，也就是說每隔固定一段間，就會呼叫Alarm::CallBack()，因此，對於這個alarm來設計一個累加器Bedroom::\_current\_interrupt進行全局去記數，而Alarm的設定是每100 ticks就會產生一次interrupt並呼叫Alarm::CallBack()
5. 當有程序呼叫Sleep()時，會呼叫WaitUntil()，然後將其丟入Bedroom安眠
6. 在CallBlack()被呼叫時，去Bedroom檢查誰該醒來

Part2:

1. 在Constructor中決定要用哪一種類型的排程，並且宣告相對應的compare function
2. 如果需要搶先設計，則在Alarm::CallBack()決定是否要呼叫interrupt->YieldOnReturn()查看是否有更需要優先的process要執行
3. 在CPU Burst Time的計算上，如果採用運行時估計，可以在利用執行 user program 的時間累加器進行計算
4. 分別在在每一個 kernel 宣告的地方( Initialized() )都給一個決定 scheduler 類型的參數
5. 希望可以下這樣的command 來執行

$ cd code/threads

$ ./nachos FCFS

$ ./nachos SJF

$ ./nachos Priority

$ ./nachos RR

# Implementation

Part1:

1. 定義Sleep()函數

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/userprog/syscall.h | |

1. 在start.s中修改assembly language

.globl(c++ 寫的部分才能找得到) .ent Sleep(Sleep 的 entry 在這裡)

其餘部分大概跟其他function 一樣，依樣畫葫蘆

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/test/start.s | |

1. 修改exception.cc, 收到 SC\_Sleep 這個 syscall 編號時要做的行為

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/userprog/exception.cc | |

1. 必須實作kernel->alarm->WaitUntil(val); 設計累加器Bedroom::\_current\_interrupt。當有程序呼叫 Sleep() 時，會呼叫 WaitUntil()，然後將其丟入Bedroom安眠。

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/threads/alarm.h | |

1. 在CallBlack()被呼叫時，來去Bedroom檢查誰該醒來。WaitUntil的定義

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |
| /code/threads/alarm.cc | |

Part2:

1. 依照助教投影片指示先設計main.cc，接著設計標頭檔

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

/code/threads/main.cc

1. 為了實作多種不同的scheduler，先定義enum function並讓程式可以藉由

[- SchedulerType]來切換排程模式，在SchedulerType中多增加FIFO

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

/code/threads/scheduler.h

1. 在Constructor中決定要用哪一種類型的排程，並且宣告相對應的 compare function

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

/code/threads/scheduler.cc

1. 在Alarm::CallBack()決定是否要呼叫interrupt->YieldOnReturn()查看是否有更需要優先的process要執行

|  |  |
| --- | --- |
|  | |
| kernel->stats->userTicks 是執行 user program 的時間累加器 | |
|  | |
| Original Text | Changed Text |

/code/threads/alarm.cc

1. 為了測試排程的結果，多增加一個Thread::SchedulingTest()的函式，

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | A | B | C | D |
| Priority | 5 | 1 | 3 | 2 |
| Burst Time | 3 | 9 | 7 | 3 |

並設置各程式的 burst time 和priority特性，各個程式特性分別設置如上表，

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

/code/threads/thread.cc

1. 必須在 class Thread 增加 setPriority(), setBurstTime(), SchedulingTest() … 等 method header。

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

/code/threads/thread.h

1. 將 testcode 加入到 ThreadedKernel 中。

|  |  |
| --- | --- |
|  | |
|  | |
| Original Text | Changed Text |

/code/threads/ kernel.cc

1. 後來發現make 有錯誤，經過上網一查ref[3] 向 NachOS 4.0 作業進發 (2) <https://morris821028.github.io/2014/05/30/lesson/hw-nachos4-2/>問題是發生於我們在 /threads 下修改 main.cc 的關係，所以必須在每一個 kernel 宣告地方都宣告的地方( Initialized() )都給一個決定 scheduler 類型的參數。

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

userprog/userkernel.h

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

userprog/userkernel.cc

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

network/netkernel.h

|  |  |
| --- | --- |
|  | |
| Original Text | Changed Text |

network/netkernel.cc

# ****Result****

Experiment result and some discussion

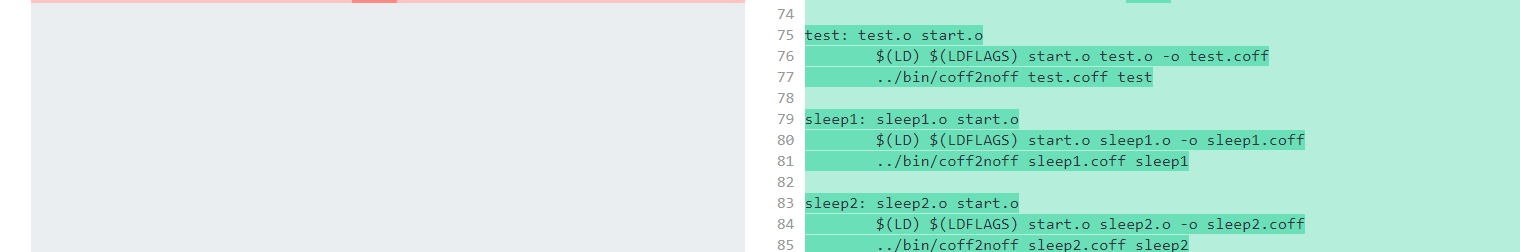
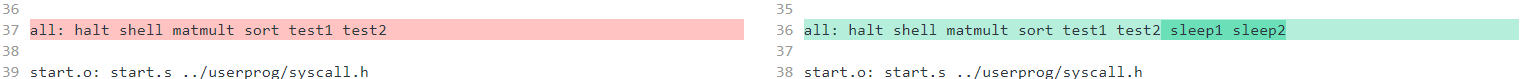
Part1:

1. 新增兩個有用到 sleep 的程式，檢查正確性，

sleep2 的週期是 sleep 的 1/10

|  |  |
| --- | --- |
| 1. #include "syscall.h" 2. main() { 3. int n; 4. for (n = 0; n < 5; n++) { 5. PrintInt(1111); 6. Sleep(1000000); 7. } 8. }   sleep1.c | 1. #include "syscall.h" 2. main() { 3. int n; 4. for (n = 0; n < 5; n++) { 5. PrintInt(2222); 6. Sleep(100000); 7. } 8. }   sleep2.c |

1. 修改 Makefile加入編譯，把 coff 轉成 NachOS 自己定義的 noff 格式



/test/Makefile

1. 下make指令產出執行檔
2. 執行 ./nachos -e ../test/sleep1 -e ../test/sleep2

|  |  |
| --- | --- |
| 前兩次會參雜 10 次的 PrintInt(1111)  但會進入休眠給sleep2  sleep2會趁sleep1還在休眠時  執行完時，因為我只有iteration 五次所以還會再等一下再把 PrintInt(2222)執行完五次 |  |

Part2:

對應剛剛Thread::SchedulingTest()的函式，各個程式特性分別設置如下表，

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | A | B | C | D |
| Priority | 5 | 1 | 3 | 2 |
| Burst Time | 3 | 9 | 7 | 3 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| ./threads/nachos FCFS | ./threads/nachos SJF | ./threads/nachos PRIORITY |

分別測試FCFS、SJF、Priority，討論其結果

|  |  |
| --- | --- |
| FCFS | 確實是按照先來後到的順序，也就是ABCD的順序做執行 |
| SJF | 具備最長時間CPU-Burst Time的B程序確實是最後一個執行的，  然而同時是最短長度的兩個程序A與D是最先被執行的 |
| Priority | 確實是優先等地的順序，也就是BDCA(1235)的順序做執行 |