Operating System

MP3 : CPU scheduling

Report

Team 20 member :

105060011 蔡悅承

105060016 謝承儒

Team member contribution :

Implement : 蔡悅承 謝承儒

Report : 蔡悅承 謝承儒

Goal

一. 將 scheduler 改寫成 multilevel feedback queue，每個 Thread 都有自己的 priority(範圍 0~149，149 最高、0 最低)

1. L1 : Preemptive SJF
   (1) Priority : 100~149
   (2) execution time is approximated using the equation:

$$t(i) = 0.5 \cdot T + 0.5 \cdot t(i-1)$$

2. L2 : Non-preemptive priority
   (1) Priority : 50~99
3. L3 : Round-robin
   (1) Priority : 0~49
   (2) time quantum 100 ticks

二. 做出 Aging 機制

該 Thread 每過 1500 個 tick，就將 priority 上升 10。

三. 使用 “**-ep**”來給予該 Thread 的 priority

Implement multilevel feedback queue in NachOS

一. 實作想法

1. 利用”-ep”來給予priority，並根據priority，將thread放入對應的ready queue
2. 先從L1找Thread執行，若是沒有就找L2，再沒有就找L3
3. 當找到要執行的Thread時，切換成相對應的scheduler algorithm
4. 固定做Aging來保證Thread的priority

二. 修改內容解釋

1. Thread.*
   (1) 變數

   以下的每個變數都有相對應的input/output function

   a. priority

      根據該值來放進相對應的ready queue

      L1 : 100~149

      L2 : 50~99

      L3 : 0~49

   b. starttime

      第一次進到ready queue的時間

   c. predictburstTimes

      這一次CPU Burst Time應該要做多少tick

   d. burstTime

      這次CPU Burst跑了多少Tick

   e. agecount

      已經aging幾次

   (2) 函式

   a. calculateNextBurstTime

      預測下一次CPU Burst Time做多少tick，從I/O回來(Thread::Sleep)時做運算

2. Scheduler
   (1) 變數

   a. preemptive

      可能會發生Preempt

      若其值為TRUE，會在OneTick()時呼叫Thread::Yield，檢查是否需要Context Switch

3. Timer.*
   (1) 函式

   a. Constructor

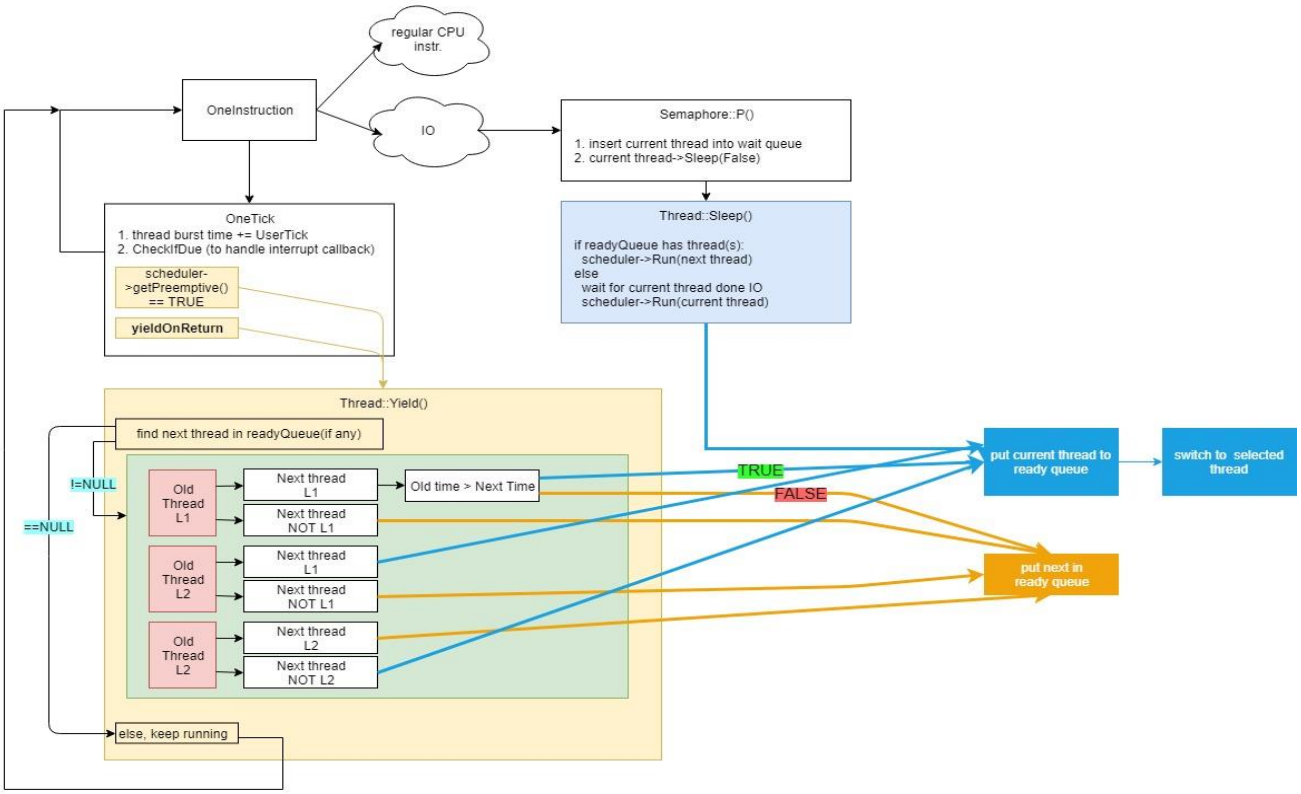
      disable = false=>先將Timer關閉，等確定是L3的thread要執行時，再打開

   b. Enable()

      將disable設成false，並同時放入一個Timer Interrupt

三．實作內容

1．Thread切換的過程



有兩種function可能會讓Thread切換 ：

(1) Thread::Sleep

a. 觸發時機 ：

(a) Current Thread進到I/O

(2) Thread::Yield

a. 觸發時機 ：

詳見下方 2.Yield

b. 會拿出下一個Thread來檢查是不是要Preempt

(b) Old thread 是 L1

New thread 是 L1 => 若New Thread預測時間小於Old Thread剩餘時間就Preempt；反之就將New thread放入Ready queue

New thread 不是 L1 => New thread放入Ready queue

(c) Old thread is L2

New thread 是 L1 => Preempt

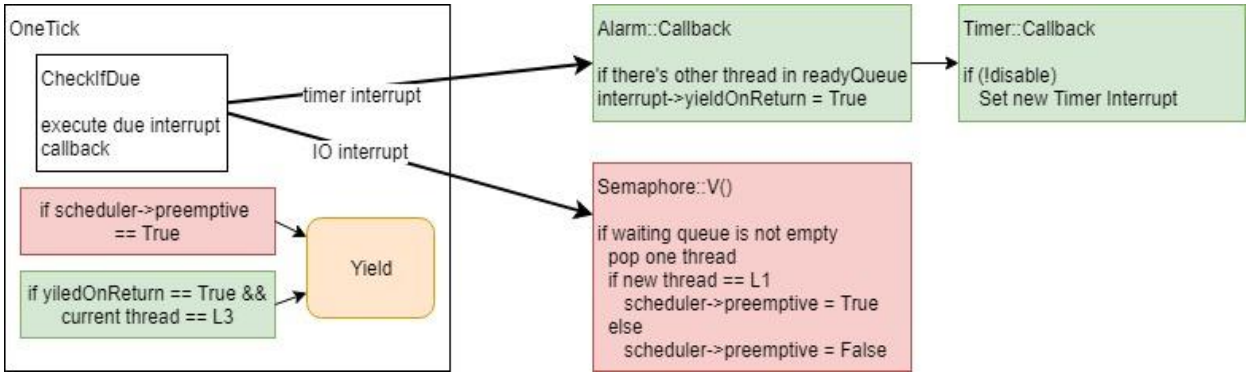
New thread 不是 L1 => New thread放入Ready queue

(d) Old thread is L3

New thread 是 L2 => New thread放入Ready queue

New thread 不是 L2 => Preempt

2. Yield觸發時機



(1) 觸發時機 :

a. 當有屬於L1的Thread從I/O回來時 (紅色區塊)

b. Timer Interrupt => round-robin(綠色區塊)

(2) round-robin機制過程

a. 在Context Switch時發現是下一個Thread是L3時，將Timer打開 (disable=FALSE)，放入Timer Interrupt

b. Timer Interrupt(Alarm的Callback)時，會將yieldOnReturn設成True

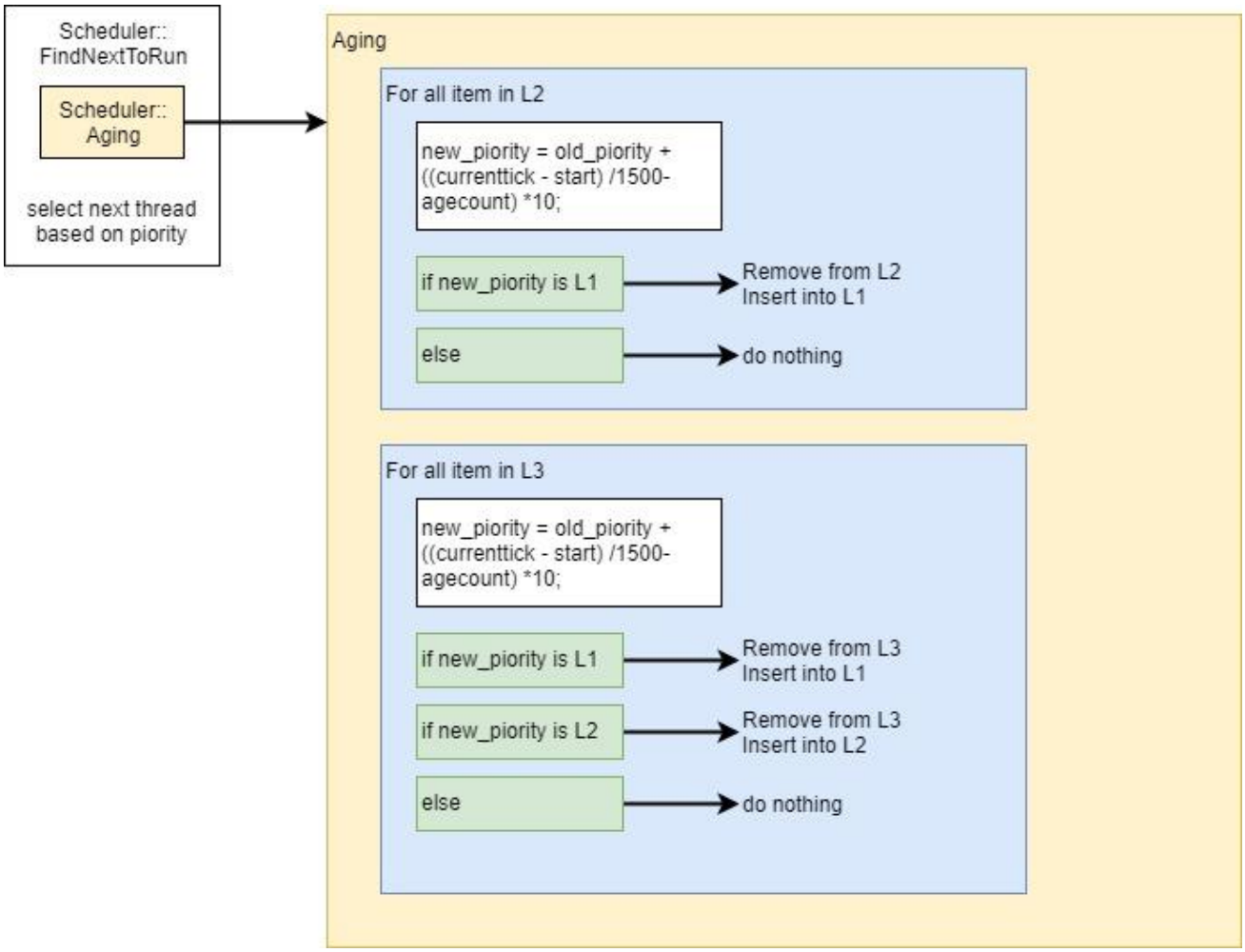
c. Timer的Callback會檢查disable，來看是否要繼續放下一個Timer Interrupt

=>會導致若下一個thread不是L3，也會放入Timer Interrupt

=>除了yieldOnReturn，還要檢查current thread是不是L3的

d. Onetick()裡呼叫Yield()來切換下一個Thread

3. Aging



(1) 步驟

a. Currenttick = total tick, start = 該Thread進來的時間

b. live time(存活時間) = Currenttick-start

c. 藉由live time和ageCount(已經ageing幾次)來更新該thread的priority，並更新ageCount

d. 檢查new priority，決定該thread是否需要改變ready queue

e. 對L2、L3的所有Thread做步驟a~d，L1已經是最高的prioity，所以不再更新

4. 注意事項

(1) 從I/O回來的Thread和CPUT的Thread相同=>對自己做Context Switch
=>檢查old和next Thread是否相同，如果相同就不做Context Switch

(2) Test2先進lock的wating queue，但還是後進的Test拿到lock

當Test1印完字後會將SynchConsoleOutput的lock解除，此時，在lock的waiting queue的Test2會出來準備得到lock。

但這時CPU裡的是Test1，所以Test1的CPU Burst Time結束後進到I/O Burst Time，又再次得到lock，Test2還是沒有拿到lock

(該情況發生在從waitng queue回來的Thread無法Preempt Current Thread的時候)