|  |
| --- |
| 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

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

該Thread每過1500個tick，就將priority上升10。

1. 使用“**-ep**”來給予該Thread的priority

Implement multilevel feedback queue in NachOS

1. 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
2. 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

* + - 1. priority

根據該值來放進相對應的ready queue

L1 : 100~149

L2 : 50~99

L3 : 0~49

* + - 1. starttime

第一次進到ready queue的時間

* + - 1. predictburstTimes

這一次CPU Burst Time應該要做多少tick

* + - 1. burstTime

這次CPU Burst跑了多少Tick

* + - 1. agecount

已經aging幾次

* + 1. 函式
       1. calculateNextBurstTime

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

* 1. Scheduler
     1. 變數
        1. 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

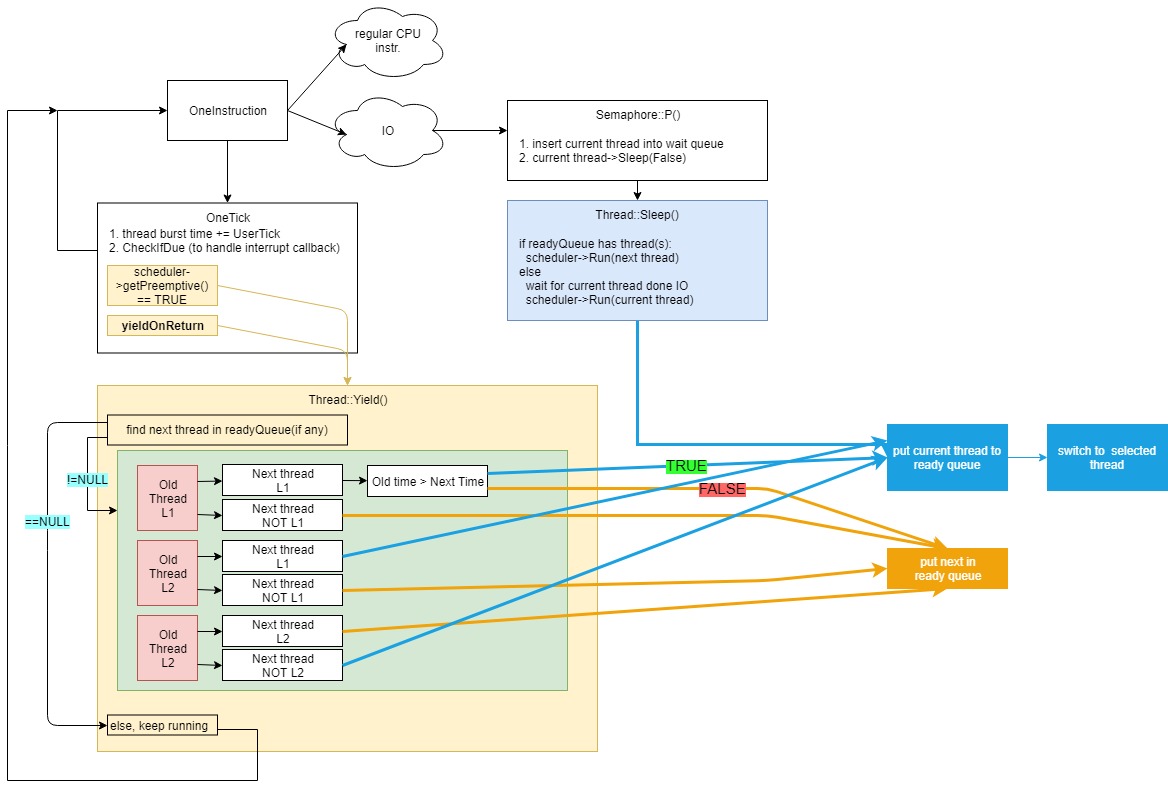
* 1. Timer.\*
     1. 函式
        1. Constructor

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

* + - 1. Enable()

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

1. 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

觸發時機 :

* + - 1. Current Thread進到I/O
    1. Thread::Yield

觸發時機 :

詳見下方 2.Yield

會拿出下一個Thread來檢查是不是要Preempt

* + - 1. 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

* + - 1. Old thread is L2

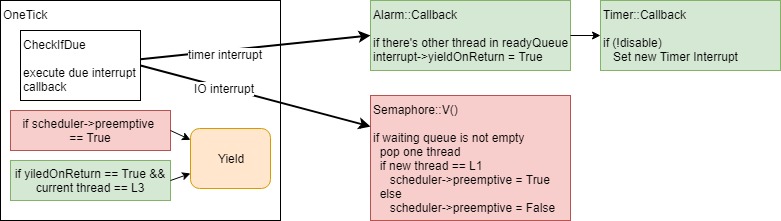
New thread 是 L1 => Preempt

New thread 不是 L1 => New thread放入Ready queue

* + - 1. Old thread is L3

New thread 是 L2 => New thread放入Ready queue

New thread 不是 L2 => Preempt

* 1. Yield觸發時機
     1. 觸發時機 :

當有屬於L1的Thread從I/O回來時 (紅色區塊)

Timer Interrupt => round-robin(綠色區塊)

* + 1. round-robin機制過程

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

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

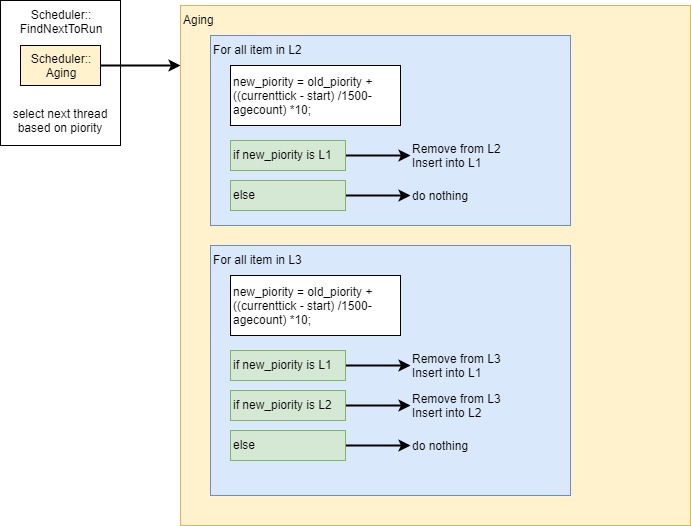
Timer的Callback會檢查disable，來看是否要繼續放下一個Timer Interrupt

=>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

Onetick()裡呼叫Yield()來切換下一個Thread

* 1. Aging



* + 1. 步驟

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

live time(存活時間) = Currenttick-start

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

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

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

* 1. 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

* + 1. 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)