HW5 REPORT

王柏舜, 0716098

***Abstract*—這個作業希望我們透過從以軟體的修改來達到程式運行效率的提升以及增加 counter 來計算 context-switch overhead , synchronization overhead，再過程中需要我們閱讀理解 freertos 的 source code 。**

I. THREAD MANAGEMENT

Freertos 的 thread 創建是透過 xtaskcreate() 這個 function 來完成的，並且在 thread 創立完成後，需要呼叫 vTaskStartScheduler()來選取要執行個 task，再我看過 source code 後我了解到 task 是在 freertos 中越來排程的單位，因此在呼叫 xTaskCreate 後這個 fucntion 會創立一個新的 task 並且將他加入 ready queue 中，這個 task 會拿到一個新的 TCB object 用來記錄 task 的名稱、priority、等等的其他細節，而這個 task 會需要 allocate memory ，這裡分配到的記憶體是由 heap 中的 pvPortMalloc()來分配的，而 task 被 create 好了之後並不能馬上被使用的，他需要呼叫 vTaskStartScheduler 才有辦法選 task 來跑，這個 function 一開始呼叫的時候，會先加入一個優先度最低的 idle task，這個 idle task 是為了卻保 CPU 再任何一個時間至少有一個 task 可以執行，在這個 function 的最後會呼叫 xPortStartScheduler 來真的拿到真的要執行的 task ，這會再 xPortStartScheduler ，中呼叫 xPortStartFirstTask() 完成，再理解完 task 的基本架構後我開始理解 task priority 之間的關係，我從 taskSELECT_HIGHEST_PRIORITY_TASK 這個 fucntion 了解，freertos 的 ready queue 有很多個不同的 level，每一個 level 是一個獨立的 queue ，要選 priority 最高的 task 會從 priority 最高的那一層 queue 開始來看，如果那一層是 empty 的話那麼就會繼續往下一層 priority 低一點的 queue 來檢查，再來是要決定 task 切換的部分，再 port.c 裡面有一個 function 可以設定 timer 發出 interrupt 的 ticks，用來引發 trap 啟動 ISR 來執行 task 的切換，這個 trap 的過程會再統計 overhead 的時候才說明，這個部分我想把重點放在 vtaskSwitchContext() 上，呼叫這個 function 後會先判斷 scheduler 是不是 suspend 的狀態，如果不是的話就會執行 task 的切換，真的在做 context switch 的是，當作完前置作業後會呼叫 xPortStartScheduler 這個 function 來拿到下一個 highest priority task 來執行。

II. MEASURE CONTEXT-SWITCHING OVERHEAD

A. 計算規則

計算介於收到 timer interrupt 到新的 thread 開始執行之間的 cycle 數。

B. 實作

再知道要計算 context switch overhead 的時候我一開始想的是因為要 context switch overhead 所以應該會有發出 software interrupt 的訊號出來表示現在跑的軟體有 trap 要把 csr 裡面新的 task 執行到的 program counter load 進來，還有當時的 register 的狀態等等的資訊進來，因此我一開始抓了 core_top.v 裡面的 irq_taken_i 到 ILA 看是不是有這樣的 interrupt 發生。以及對應的 pc，但是在設定對應的 pc 當作 trigger 來看之後發現並沒有這樣的 interrupt 發生，所以有繼續看 source code 看看是不是漏掉了甚麼，後來決定是用 vtaskswitchcontext 這個 function 進入點和 return 的 program counter 來計算，但是在更細部的看之後發現這是有問題的因為這個 function 並沒有含括最完整的部分，最後再 portASM.s 裡面發現 freertos_risc_v_trap_handler 、processed_source 這兩個 fucntion ，前者的功能是收到對應的 interrupt 後執行對應的 interrupt service routine ，而後者是將新的 task 要得 cpu 狀態都設置好，因此最後我計算的 cycle 數是從 freertos_risc_v_trap_handler 的進入點，到 processed_source 的 return 的 program counter 這個位置。

III. MEASURE SYCHRONIZATION OVERHEAD

A. 計算規則

這個部分總共有四個部分要算第一個部分是要計算進入 critical section 需要幾個 cycle，而第二個部分是要計算離開 cirtical section 需要幾個 cycle ，第三個部分是要計算拿到 mutex 需要幾個 cycle，第四個部分是要計算 give mutex 需要幾個 cycle。

B. 實作

1. 計算進入 critical section:在 freertos 的實作中 enter critical section 是利用 interrupt disabling 來完成的，再 source code 的 define 中 taskENTER_CRITICAL 是由 vtaskEnterCritical 這個 function 來完成，再這個 function 裡面會呼叫 portDISABLE_INTERRUPT 這個 function ，這個 function 才是真正關掉 interrupt 的地方。在計算 cycle 數的部分我選擇 vtaskEnterCritical 的進入點到該 function return 的 program counter。

2. 計算離開 critical section: freertos 在這部分的實作方式和進入 critical section 相同，由 define 的關係可以知道最終 interrupt enable 的地方在 vtaskExitCritical。計算 cycle 的部分是從 vtaskExitCritical 的進入點，到 return 的 program counter。

3. 計算拿到 mutex：拿到 mutex 的實作是在 xQueueSemaphoreTake 這個部分實作的，freertos 對於 semaphore 的實作是用 queue 來實作的在 xQueueSemaphoreTake 裡面會是都過 queue 的狀態以及 xTaskCheckTimeOut 的結果來決定是否能拿到 mutex 如果不能那就要被 block 住。計算 cycle 的部分我是從 xQueueSemaphoreTake 的進入點開始到其 return 的 program counter 間的 cycle 數。

4. 計算解開 mutex：解開 mutex 的實作是在 xQueueGenericSend 這個部分實作的，一樣是利用 queue 的狀態來決定如何操作，計算 cycle 的部分我是從 xQueueGenericSend 的進入點到其 return 的 program counter 之間的 cycle。

C. 統計結果

統計結果的呈現我是利用 ILA 以 program counter 的值當作 trigger 來獲得，trigger 的條件為當 task 1 被 delete 時，當時的統計數據，呈現的數值為整個程式跑完 overhead 的總額，結果為下圖。

```
context overhead 5500504
enter critical   288911951
exit  critical   300199643
get mutex        123379743
give mutex       144310537
```

D. ISSUE

在計算 mutex 的部分因為在 take，give mutex 的 function 中途會把 interrupt enable 之後再把 interrupt disable 來讓其他的 task 可以 give 或 taek semaphore 但這使的這裡的計算結果有可能不是最純正的 mutex overhead 。

TIME QUANTUM

我也有嘗試調整 time quantum 的大小來看 task 間如何搶 mutex 我是了 10ms,5ms,1ms ，結果發現 task 1 counter 的大小依序為 1ms>10ms>5ms，而 overhead 的結果沒有甚麼改變，這和我原本預期的結果不太一樣我以為 time quantum 愈小 task 之間的資源分配會稍微平均一點，但就看到的結果好像不是如此。The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g.” Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

OPTIMIZATION

這個部分我沒有成功做出來，我對於效能的評估是利用 overhead 來當作標準，因為老師提供的 source code 版本是 general purpose 的，在 trace code 的過程中，我有發現一些 comment 的內容，我有看到像是在 freertos config.h 裡面加把原本 define 的值改成別的這樣就可以使用 port optimize 的 funtion 版本，因為我有看到裡面有提到 port optimize 是支援特定的 micro artectture 我有試圖去更改我的 config 但是對於 overhead 都沒有甚麼影響。