**Real-Time System hw1**

廖怡誠

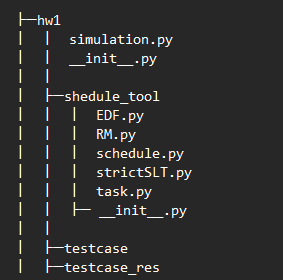
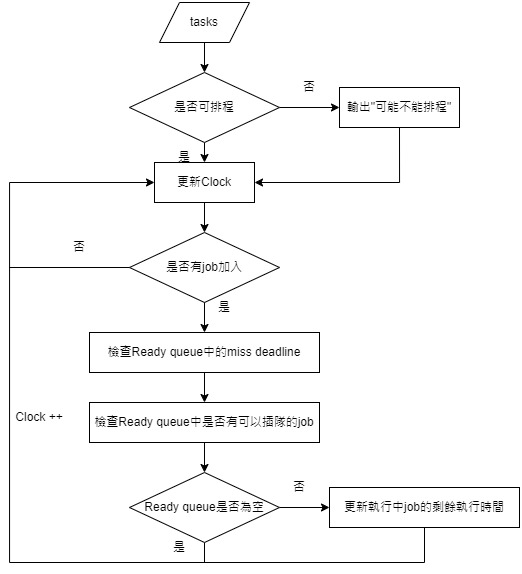
這次的作業是使用RM、EDF、strictLST排程機制，模擬實際的工作排程，以下將分成架構設計、遇到問題與心得三個部分。

1. **架構設計**

圖1.為這次作業的樹狀架構圖，為了模擬實際的timer運行，首先我設計了simulation.py，撰寫這份程式碼的用意是模擬實際clock的變化與task傳入到Ready queue的過程。我將task.txt的所有task儲存起來，並以計數器的方式模擬timer，在對應的時間點，將task傳入到Ready queue當中等待執行。

接著我將RM、EDF、strictSLT三個排程機制相同的架構寫成一份shedule.py，並以繼承的方式延伸成三種不同的排程機制的程式碼，這樣的設計方式可以有效的減少程式碼重複性過高的問題，只要針對對應的優先度判斷即可。

而在Ready queue的設計上，需要使用記憶體不連續的資料結構使用，在Python中我使用List來實現此概念。

**圖1.樹狀架構圖 圖二.程式流程圖**

1. **程式流程圖**

圖2.是這次的程式流程圖，在task要開始排程之前，需要檢查是否可排程，確認後才會正式進入的模擬排程的架構中。首先，根據對應的時間將對應的job放入Ready queue中(0)，並且每次都需要檢查是否有job已經miss deadline需要將其移除避免出現問題(1)，接著確認是否有出現優先度較高的job需要插隊(2)，最後，如果有正在執行的job，則需要更新其剩餘執行時間(3)，重複進行(0)、(1)、(2)、(3)四個操作直到設置的時間點(最大phase time加最小公倍週期)。

1. **遇到問題**
   1. **Context-Switch效能考量**

在原先教授還沒提出「當兩個job的優先度相同需要以TID較小的優先執行」這個規則之前，我認為在這個階段的規則需要考慮到context-switch問題，如同strictSLT，不斷的交換job，需要將被插隊job的狀態儲存到memory之中，雖然與執行時間相比，消耗的時間很短，但在實際的應用必須要追求效率的極致，因此我認為在優先度相同時，應該以此類方式設計。

此外，我將兩種寫法進行比較，兩者的排程順序略有不同，差別只在於原先要交換的兩個job順序不同，期間內的miss deadline的數量與沒有job的時間點數量仍會相同，這顯示已context-switch作為規則效能會更好。

* 1. **RM的schedulability test**

作業有要求RM與EDF在進行排程之前，需要先進行schedulability test。而作業的testcase在RM的schedulability test都會是”可能無法排程”，我認為這樣的結果蠻異常的，但經過檢查也沒有發現問題，畫出甘特圖也能夠畫出沒有miss deadline的順序，推測可能的原因是schedulability test較為嚴謹，但實際情況不會太常出現miss deadline的問題。

1. **心得**

在撰寫這次作業的過程中，使我重新複習了排程機制的架構與流程，在書上用task(phase, period, relative deadline, exec time)畫甘特圖跟實際寫程式模擬的感覺完全不同。在寫程式的過程中，需要考量到程式的架構與函式設計的方便性，例如，在將job放入到ready queue後，應該要先檢查ready queue裡面的job是否有miss deadline的問題出現，而最初我在設計的時候，沒有考慮到前後相依性的問題，就把檢查的函式放在確認是否有job要插隊的後面，這樣的流程會導致程式找到slack為負的job要換掉現在正在執行的job，這個操作是不被允許的。又或是許多名詞的定義是甚麼意思，例如，當初我在設計job的release time時， 我誤以為release time是每次job開始執行的時間，因為講義的敘述是說變成可執行與排程的，這讓我理解錯意思。因此我很慶幸有這次的作業可以讓我在考前重新練習排程機制，讓我對名詞定義與運行流程又更熟悉。

然而，在這次的作業中，我認為遇到最大的麻煩是題目的描述有些許不清楚與錯誤，讓我在寫的時候，需要花很多的時間思考或與老師確認，像是當優先度一樣時該如何處理，我最初是考慮context-switch來設計，因此如果優先度相同，我會先讓目前的job執行完再判斷下一個，但老師需要的是TID較小的優先度較高，這個通知在截止日當天才公告，個人有點過於倉促，又或是輸出格式的規定也是沒有定義清楚的地方。