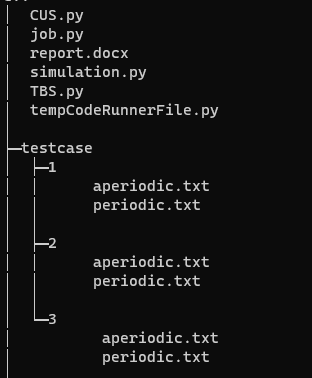
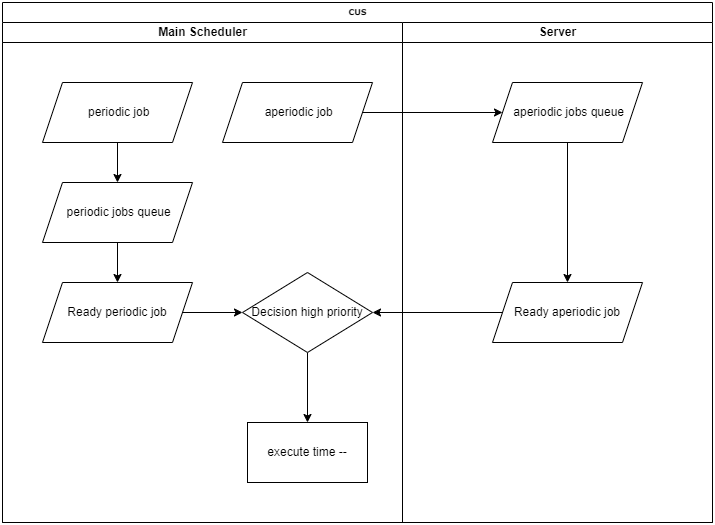
**Real-Time System hw2**

廖怡誠

這次的作業是使用CUS、TBS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**

在架構的設計上，我主要分成main scheduler與server兩個部分，前者負責處理週期性的工作，後者則處理非週期性的工作，main scheduler需要先從週期性工作的queue決定最高優先度的週期性工作，再與server決定出的非週期性工作進行比較，若週期性工作優先度較高則該工作執行時間減一；反之，若非週期性工作優先度較高則該工作執行時間減一。兩者最大的不同在於，週期性工作是動態優先權，每次都需要重新決定優先度較高的工作，而非週期性工作則是固定優先權，採用FIFO規則。

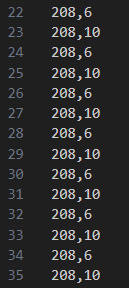
**圖1.樹狀架構圖 圖二.程式流程圖**

* 1. **程式流程圖**

圖2.是這次的程式流程圖，在每一次的時間點都需要先檢查週期性的工作是否有miss deadline，在依據Absolute deadline來排序找出最高優先度的工作，接著Server有根據RULE找出目前的Ready job，兩者以Absolute deadline決定優先權，並遞減該工作的剩餘執行時間，在每一次的循環中，週期性工作需要從queue中重新決定一個工作出來，而server則是會將目前的ready job消耗完畢才會再從非週期性工作中在找下一個使用。

1. **遇到問題**
   1. **非週期性工作同時間抵達的執行優先度**

在本次專題的複雜testcase中有出現非週期性工作同時抵達的情況(圖三)，此時需要FIFO規則判定屬於同時進入，教授的說法為按照提供的順序執行，而實際情況是優先執行較短執行時間的工作，我比較支持後者的作法，假設非週期性工作能夠執行的時間是固定的，那必然是較短執行時間的工作能夠完成的越多，能夠提高的效能也越多，但我認為事務上還是要考量長短執行時間的工作miss deadline個別所帶來的影響來判定。



**圖三.非週期性工作同時抵達**

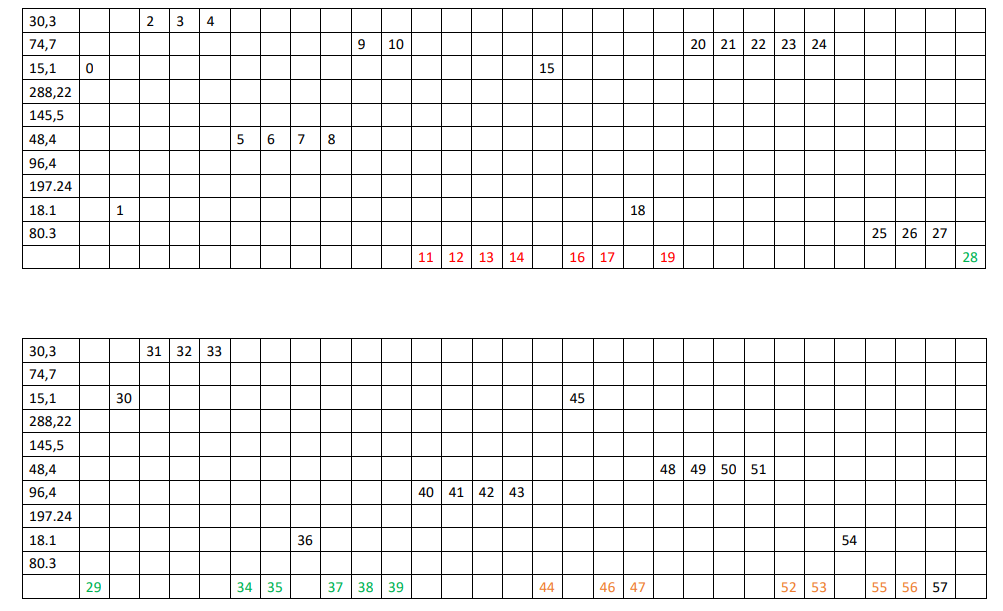
* 1. **非週期性工作miss deadline問題**

本次專題沒有探討到非週期性工作miss deadline的問題，雖說非週期性工作對整體效能影響不大，但也不是說不會影響，以CUS來舉例，投影片並沒有提及clock到deadline時，若ready job還沒執行完，該放棄此job還是延長執行時間，實際情況應為放棄此job，但也有人針對這個問題使用動態電壓調整改善系統的能源效率和反應時間或是Work-demand-based Slack-Stealing scheme，當非週期性工作大於server能夠附載的量時，會利用空閒時間提早完成非週期性任務，來降低反應時間。不過，經過檢查之後，測資並沒有出現非週期性工作miss deadline的問題，且倘若真的出現miss deadline的工作，server的deadline應該先進行補充還是正常等到Rule達成在補充也是一個問題點。

1. **心得**

這次的專題實作起來相較於第一份專題簡單，沒有複雜的邏輯，只需要將程式分成週期性工作與非週期性工作兩部分的程式，週期性工作的概念就是EDF，而非週期性工作則只要按照規則一步一步實作出來即可，在我處理非週期性工作的server程式碼中，我仿照投影片教的consumption與replenishment實作，consumption負責遞減server的budget，replenishment則負責找出目前的ready job以及更新server的deadline，只要根據兩者的功能放入到對應的段落中即可，像是CUS的replenishment要放在有新的非週期工作進入queue或是clock已經到server的deadline；而TBS則是要放在server的budget消耗完的部分。最初，我的寫法是將規則拆開，先判斷非週期性工作的queue是否為空，若為空，則為R2a或R3b，不為空則為R2b或R3a，但這樣的分法會無法區分出R2b和R3a，因為R2b也有clock等於deadline的情況，於是最後我直接以規則分為R2與R3兩類。

我認為這份專題最麻煩的地方在於檢查錯誤，因為需要跑1000個時間點，如果在比對後，發現不同的部分在時間點500，就不知道如何檢查，總不可能畫圖畫到時間點500，幸運的是在TBS的0.8 testcase中，錯誤的地方在時間點57，於是我從時間點0開始畫到時間點57(圖四)才發現是server的deadline出錯。



**圖四.畫圖驗證程式結果**