OS Homework3

資訊三乙 11027209 巫年巨

開發環境 : python , 版本 3.10.7

實作方法與流程

這次有作業要實做 5 個 Page Replacement 與 1 個呼叫全部的方法，由於有大量的資料需要紀錄，故使用多維 list 把全部的資料(記錄起來，除此之外多設了一個一維的 list 叫 page_frame 來記錄現在 page_frame 的狀況，而處理有 LFU 跟 MFU 的狀況時設了一個 dict 字典來記錄某個字被 reference 等的次數，每個 placement 方法大致上都會跑 page_reference_string 長度大小的迴圈，依三個狀況去做判斷，分別是在該字不在 page_frame 且 page_frame 沒滿或是有滿，或是該字在 page_frame 的三個情況，第一個情況和第三個情況在每個 placement 的演算法都類似，一的話都是增加 page_fault 次數並記錄 F 和把該字放入 page_frame 就好了而三的話就是紀錄空字串就好了，每個演算法的不同主要是處理該字不在 page_frame 裡且 page_frame 是滿的狀況，但是共通點都是會記錄 F 和 page_fault 和 page_replace 會加 1。

那麼首先就來講第一個方法 FIFO，這個就用 queue 的概念把最一開始加入的給 pop 掉，在 append 新的字進 page_frame 就好了。

第二個方法是 LRU 的話，我的方法是創一個 index_list 大小同 page_frame 且都設成 false，然後依照現在的字(如果現在找 page_reference_string 是 i 是 5 的話，就從 4 往前找到 i = 0)為止的字，如果字出現在 page_frame 裡的話，就把 page_frame 字同 index 的地方設為 True，就這樣直到只剩一個 false 的時候，就知道那個字是最久的 • 所以要被替換掉，所以把 page_frame 的那個 false 的 index 位置給 pop 掉後再 append 新的字進來就好了。

第三個 LFU+FIFO 的話，跟上者一樣創一個 index_list，然後放的是無限大，用意是為了找出最小的字出來，首先把 page_frame 的字放到字典中看次數，去更新 index_list 的內容，並且去記錄最小的次數和他的 index，做完後要多做一個判斷就是最小的次數是不是有多個跟他一樣的，有的話就做 FIFO 抓最小的 index 的給 pop 掉，沒有的話就是把那個最小的給 pop 掉在 append 新的字就好，要記得在 pop 掉前要先把被 pop 掉的字，在字典的次數做 reset 就好，而之後紀錄新的字不再字典的話就紀錄並且次數為 1，在的話就次數加 1 就可以結束了。

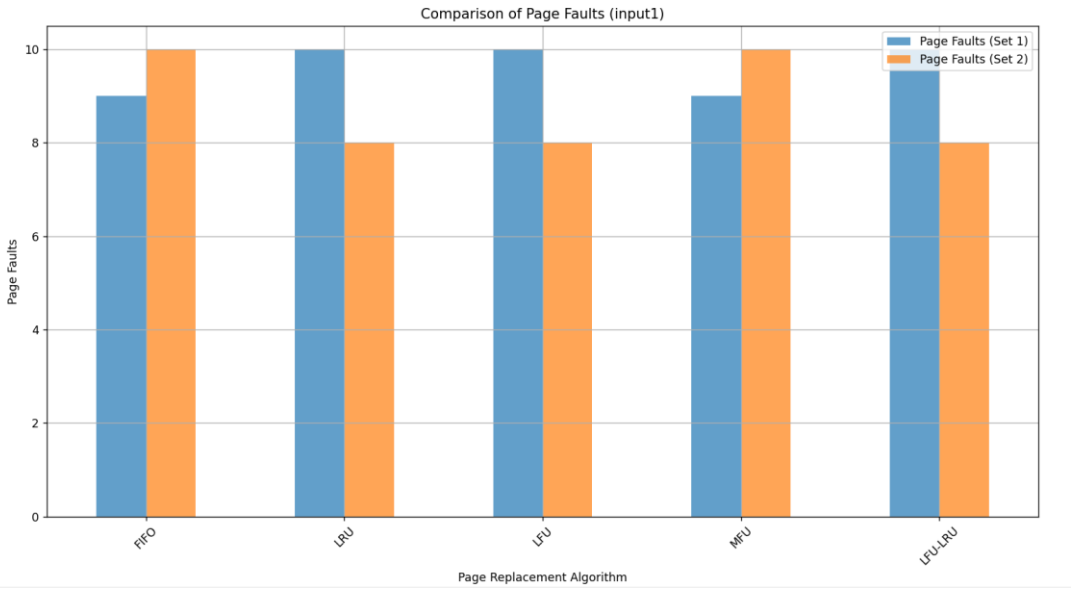
第四個 MFU+FIFO，大致上跟第三個相同，只要把無限大變成負無限大，然後找小的變成找大的，流程上差不了多少就結束了。

第五個 LFU+LRU，其實也就是把二跟三抓一點抓一點部分近來，抓的分別是一開始用 LRU 的 index_list 找次數最小的部分，如果次數最小的有兩個或以上，

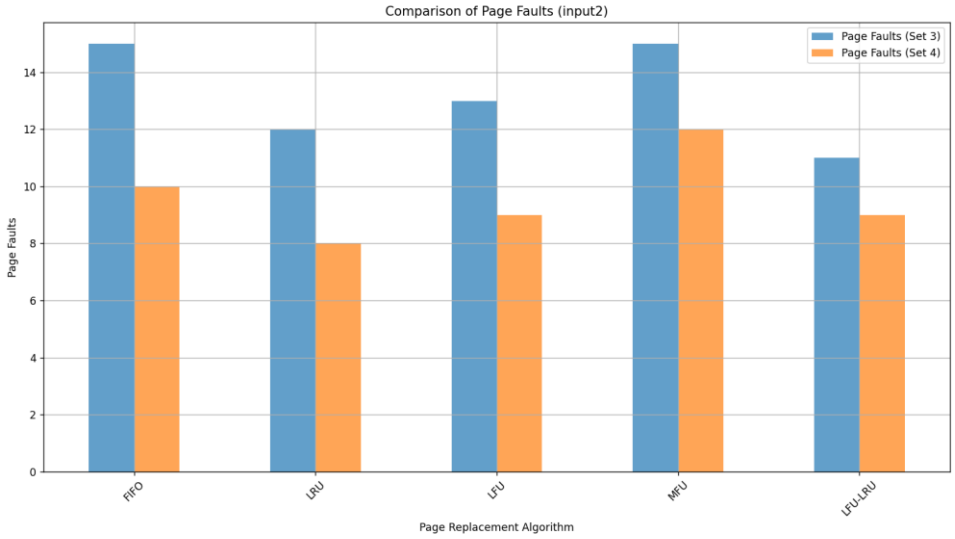
就變成另一個 LFU 的那個另一個 index_list 找剩下的那個 false 的部分，找完後的那個 index，pop 掉後就剩下更新字典之後就結束了。

不同方法的比較

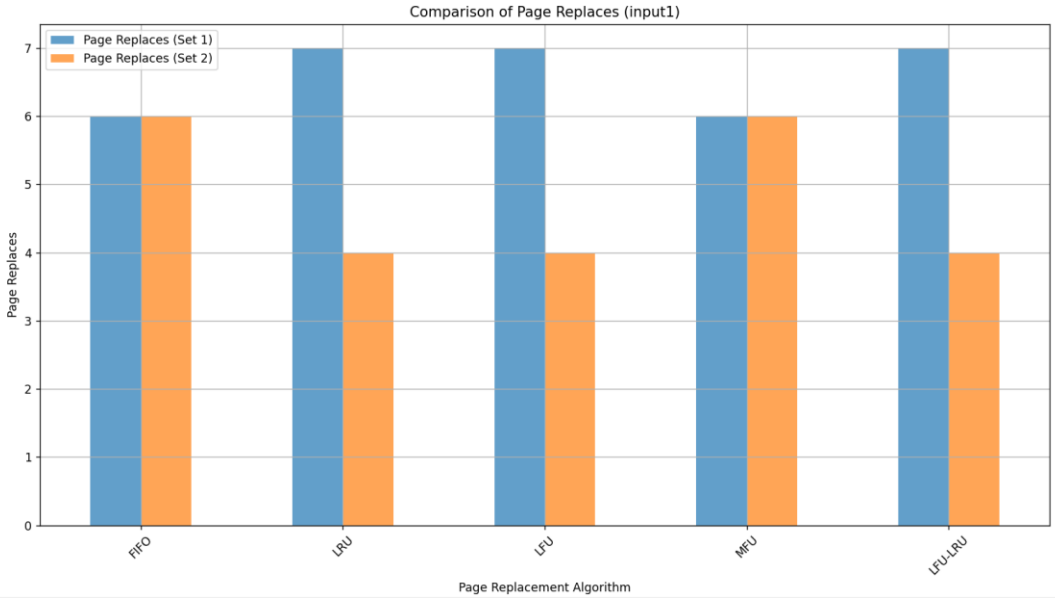
以下是 Page_Fault 的比較，分別是 input1 的資料藍色長條為放入 page_frame 大小為 3，而橘色是大小為 4 的狀況，正常來講 Page_Fault 次數隨著 page_frame 越大應該越小才對，但是可以從圖中可看出畢雷迪反例在 FIFO 和 MFU 的時候，而其他的狀況大致上可看出 LRU 和 LFU 在處理 page_fault 有著較好的表現



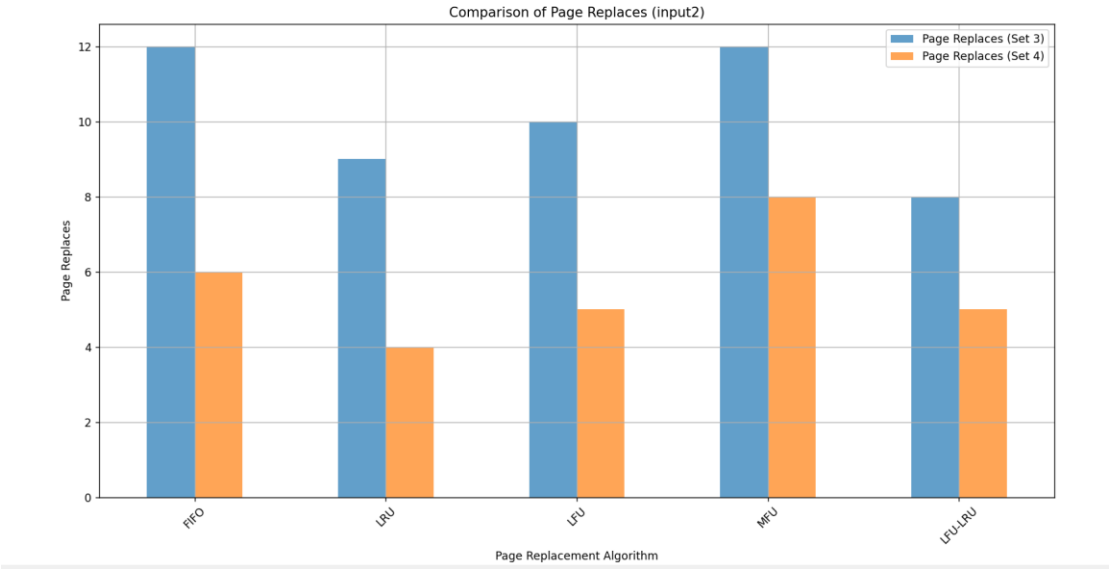
而在 input2 的時候也可看出，LRU 和 LFU 在處理 page_fault 有著較好的表現，FIFO 在所有表現中也未必最差，MFU 在 Page_frame 大小為 4 的時候，是比 FIFO 還要差的。LFU 大致上都在中規中局的位置，而 LFU+LRU 未必每次加起來就會有比較好的效果，用圖表可看出單純在 PAGE_FRAME 大小為 4 時比+了 LRU 的效果還好。



以下是 Page_Replace 的比較，分別是 input1 的資料藍色長條為放入 page_frame 大小為 3，而橘色是大小為 4 的狀況，Page_Replace 次數確實如 page_frame 大小便得越來越小，最糟也是相同，可看出在 input1 的狀況中，在藍條(3)的 FIFO 跟 MFU 的狀況是比其他 LRU 和 LFU 好的，但是橘條下(4)FIFO 和 MFU 大小上去後發生 replace 次數相同，而其他的 replace 的次數就下降至兩者之下了。



而在 input2 的時候也可看出，LRU 和 LFU 在處理 Page_Replace 大致上有著較好的表現，FIFO 和 MFU 的表現就差了點，Page_frame 大小為 4 的時候，LFU 大致上也還是在中規中局的位置，LRU 在 4 的時後效果最後，而 LFU+LRU 在 3 的時後效果最好，所以可歸納出每個方法對於 Page Replace 來說都有一定的優勢存在。



結果與討論

這次的任務中從效能上來看各種方法的話，我覺得比較來說綜合評分整體下來看選 LRU 是比較好的選擇，在大部分的情況下的 Page Fault 和 Page Replacement 較低，而應用上也屬於最佳化的演算法不會有畢雷迪反例。

至於畢雷迪反例的狀況的話，用 input1 來看，增加了 page_frame 大小但是 FIFO 和 MFU 的 page_fault 卻上升了，跟上課說的是一模一樣，可以發現在魔幻數字 123412512345 下觀察到這個狀況，所以知道了不是一直增加

page_frame 大小就可以使 page_fault 減少，還是會有特例出現的。

我比較訝異的是 MFU 的大致效能盡然會跟 FIFO 差不多，可是應用上我覺得還是會有用到 MFU 的地方，不然只有 LFU 的話感覺會有餓死或是某一個霸佔太久的狀況會發生，而 LRU 加上 LFU 聽起來把兩個優化的東西加起來要更快，但實際上卻只有少少的例子是 1+1=2 大於前面的 LRU 或 LFU，這次的總結是不是把任何最好的方法放在一起就是最好的方法，每個方法還是有它的優點存在的。