作業系統
HW3-書面報告
班級:資訊三甲
學號:**11027142**
姓名:林蕙郁

- 開發環境
  利用C++程式碼+Visual Stdio code當作開發環境。

- 實作方法、資料結構、運算流程
  {
  (type) name
  (deque)page queue:page frame的內容。
  (vector<bool> pageFault:紀錄是否發生page fault。
  (vector<vector<int>>) frameState:紀錄每個page進入之後的page queue。
  (map<int, int>) pageFrequency:pageRef, Frequency。
  (int) pageFaultCount, pageReplacementCount:紀錄fault跟replacement次數。
  }

  page fault判斷:每當一個新的page reference進入的時候，去檢查page queue內是否出現過該page，如果沒出現過代表出現page fault，有的話則反之。

  FIFO:做page fault判斷。
  出現page fault:
  1.pageFault出現次數更新
  2.pageReplacement次數更新。
  3.紀錄出現pageFault(boolean值)
  4.判斷page queue是否等於page frame size，如果是的話要把page queue最早進入的page移出，不是的話則do nothing，最後要將該page加入到queue中。

  LRU:做page fault判斷，發生page fault的話要跟FIFO作相同方式的移除。
  未發生page fault的話要讓該page移出queue後再重新排隊。

  LFU__FIFO:做page fault判斷,
  發生page fault:
  1.更新page fault值
  2.判斷page queue大小是否等於page frame size。
  值相同(page queue.size() == page frame size):將lfuPage設為最早進入queue的那個，然後去判斷每個page的frenquency，如果有找到frequency更小的話就替換lfuPage之值，最終判斷完之後把lfuPage之frequency初始化成0，並且移出page queue，然後再把新進來的page加入至page queue中，並且更新pageReplacement值
  最後不管值相不同，都要把新進來的page加入page queue中並且將該page的frequency+1。
  未發生page fault:
  1.將新進的page加入進page queue
  2.將該page之frequency+1。

MFU__FIFO：和LFU__FIFO作法幾乎相同，只是把lfuPage改成mfuPage，並且找尋frequency最大的去做更換。
LFU__LRU：做法跟LFU__FIFO很像，只是在未發生page fault的時候，需要採取LRU的作法（將page從queue中移除後在加回queue中重新排隊），而不是像FIFO是 do nothing。

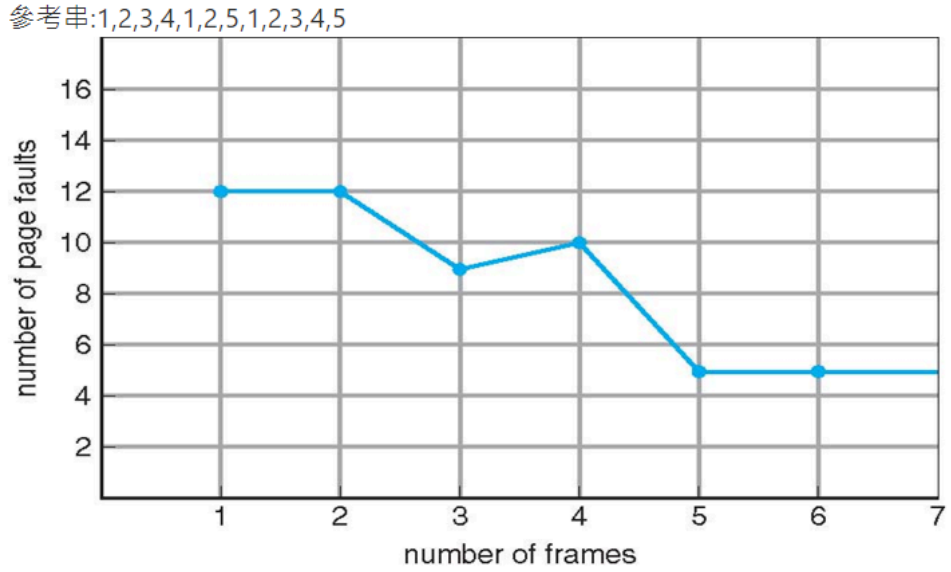
- 分析不同方法之間的比較
  根據input2.txt的結果，可以看出FIFO跟MFU的page fault次數以及page replacement次數是相同的。
  LFU+LRU的兩種次數都是最小的。
  而LRU是次數總和是處在中間。
  通常FIFO的次數最多是因為他很單純的依照FIFO的判斷方式去做判斷，這樣有可能沒辦法知道資料重要程度，因為有的資料可能很重要需要長時間待在記憶體內，但如果剛好page frame很小（假設是3），且剛好那段時間內所輪迴的page是固定的（假設是4個在輪迴），這樣就會一直發生page fault（因為不可能塞4個page進去）。
  而page replacement的數量剛好都是page fault-page frames（以input2數據所觀察）。
  LRU則是如果page已經出現在記憶體內的話就重新排隊，這樣在需要發生page fault的時候可以去移除最久未出現的page，因為有可能代表這個page比較重要，所以他需要做重新排隊的狀況去避免FIFO而從記憶體移除他。
  LFU&MFU：依照每個出現的頻率去做選擇，最少出現的可能代表他在那個當下已經沒那麼重要了所以可以把它換掉，而最常出現的可能是為了避免餓死情況，但也需要小心有可能是因為這個page很重要所以他才會一直待在裡面導致頻率最高。
  LFU+LRU：雙重方式可以使它有最大效益，可以避免FIFO而把可能是重要的page清出，也可以選擇最少出現的page清出。

- 結果與討論
  以FIFO為例子，像我剛所提及的3個空間塞4個page是不可能的，那我們就把page size往上提升是不是可以避免？由input1數據可知是不可以的，下表為輸出結果：

input1.txt/page frame	3	4
page fault	9	10
page replacement	6	6

  增加page frame的page fault不減反增，而這個就是著名的Belady's anomaly。
（續下頁）

再根據下表作探討:



可以得知，page frame是4的時候會不減反增，而5~7則是持平，這跟我們所預想的增加page frame就可以減少page fault有點出入，所以如果想要page fault要確切的可以降低的話，可能還是需要做出像LRU、LFU這種判斷方式才可以有效地降低page fault發生次數。
而根據input1，page frame改成4後，其他方式都有減少page fault跟page replacement的次數，所以利用完整的條件（例如計算頻率或是觀察最oldest的page）去做判斷是可以降低發生次數的。