OS Project3 說明文件

資訊三乙 10927256 姜美羚

一 、開發環境:

作業系統 : win10

IDE : Visual Studio Code

開發語言: python

二 、實作方法&流程:

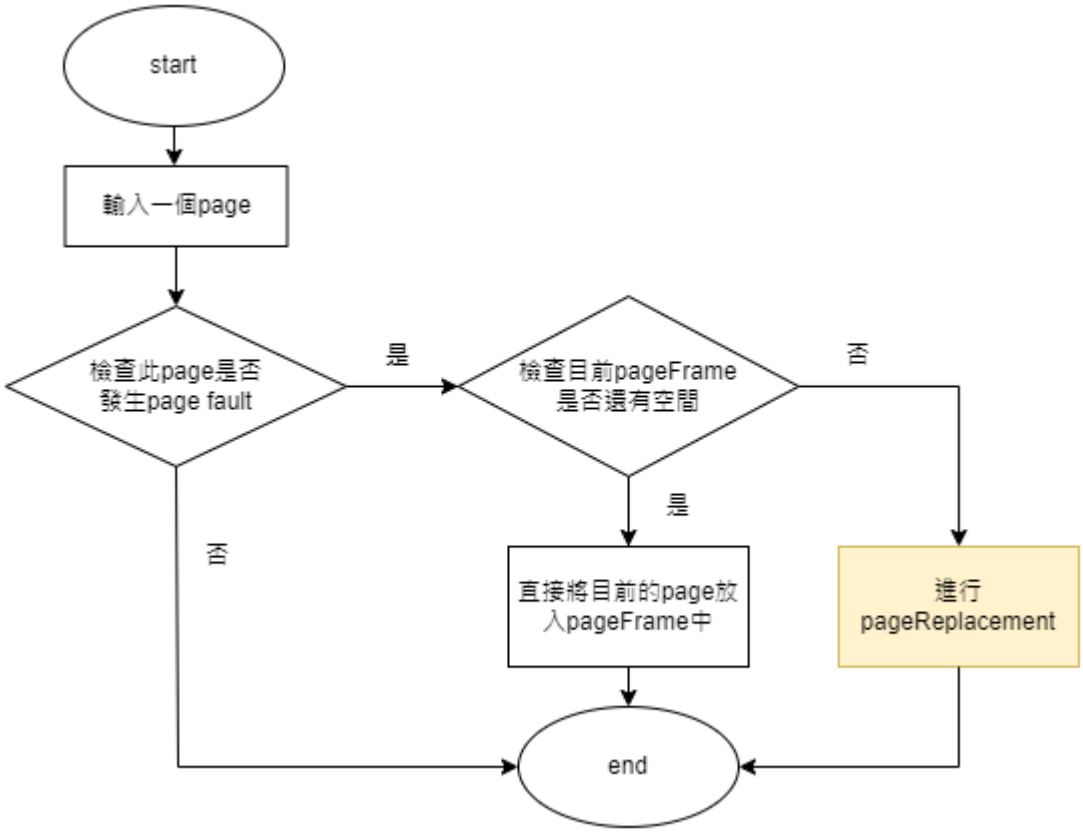
- 資料結構:

  PFrameList => 二維陣列，用於儲存各方法，各階段的 pageFrame 狀況。

  pFaultList => 二維陣列，用於儲存各方法，各階段的 pageFault 發生狀況。

  output => 二維陣列，用於儲存各方法的 pageFault 發生次數、pageReplace 發生次數。

- 處理 page 的流程如下:



本次 project 的實作流程大致上都相同(如上圖)，只有在處理 pageReplacement 的方法上有差異。

- 對於處理 pageReplacement，各個方法的詳細說明:

方法 1(FIFO):

如果目前載入的 page 發生 pageFault，且 pageFrame 滿時，則將 pageQ 中 time stamp 最小的 page(亦即 pageFrame 中的第一個)從 pageFrame 中 pop 掉，然後再將目前載入的 page 新增至 pageFrame 的最尾端，並將 pageReplace 發生次數+1。若 pageFrame 未滿，則直接將目前載入的 page 新增至 pageFrame 的最尾端。

方法 2(LRU):

如果目前載入的 page 發生 pageFault，且 pageFrame 滿時，則將 pageQ 中 time stamp 最小的 page(亦即 pageFrame 中的第一個)從 pageFrame 中 pop 掉，然後再將目前載入的 page 新增至 pageFrame 的最尾端，並將 pageReplace 發生次數+1。若 pageFrame 未滿時，則直接將目前載入的 page 新增至 pageFrame 的最尾端。如果目前載入的 page 未發生 pageFault，則從 pageFrame 中找到該 page，並更新該 page 的 time stamp (亦即將此 page 移到 pageFrame 中的最一個)。

方法 3(LFU+FIFO):

LFU，此方法是優先比較 counter 值，當目前載入的 page 發生 pageFault 且 pageFrame 滿時，選擇 counter 值最小的 page 當作犧牲者優先置換，也就是最少被 reference 到的 page。而當遇到 pageFrame 中有 counter 值相同的情況，此時會依照 FIFO 的規則，選擇 time stamp 最小的 page(亦即 pageFrame 中的第一個)當作犧牲者優先置換。若 pageFrame 未滿，則直接將目前載入的 page 新增至 pageFrame 的最尾端。如果目前載入的 page 未發生 pageFault，則從 pageFrame 中找到該 page，並更新該 page 的 counter 值，表示此 page 有被 reference 到。

方法 4(MFU+FIFO):

MFU，此方法是優先比較 counter 值，當目前載入的 page 發生 pageFault 且 pageFrame 滿時，選擇 counter 值最大的 page 當作犧牲者優先置換，也就是最常被 reference 到的 page。而當遇到 pageFrame 中有 counter 值相同的情況，此時會依照 FIFO 的規則，選擇 time stamp 最小的 page(亦即 pageFrame 中的第一個)當作犧牲者優先置換。若 pageFrame 未滿，則直接將目前載入的 page 新增至 pageFrame 的最尾端。如果目前載入的 page 未發生 pageFault，則從 pageFrame 中找到該 page，並更新該 page 的 counter 值，表示此 page 有被 reference 到。

方法 5(LFU+LRU):

LFU，此方法是優先比較 counter 值，當目前載入的 page 發生 pageFault 且 pageFrame 滿時，選擇 counter 值最小的 page 當作犧牲者優先置換，也就是最少被 reference 到的 page。而當遇到 pageFrame 中有 counter 值相同的情況，此時會依照

FIFO 的規則，選擇 time stamp 最小的 page(亦即 pageFrame 中的第一個)當作犧牲者優先置換。若 pageFrame 未滿，則直接將目前載入的 page 新增至 pageFrame 的最尾端。如果目前載入的 page 未發生 pageFault，則從 pageFrame 中找到該 page，並更新該 page 的 counter 值及該 page 的 time stamp (亦即將此 page 移到 pageFrame 中的最一個)。

三、不同方法的比較 ：

Reference string:

Input1 : 123412512345

Input2 : 70120304230321201701

Page frame size 均為 3

	FIFO	LRU	LFU+FIFO	MFU+FIFO	LFU+LRU
Input1	9	10	10	9	10
Input2	15	12	13	15	11

表 1、 pageFault 次數比較表

	FIFO	LRU	LFU+FIFO	MFU+FIFO	LFU+LRU
Input1	6	7	7	6	7
Input2	12	9	10	12	8

表 2、 pageReplace 次數比較表

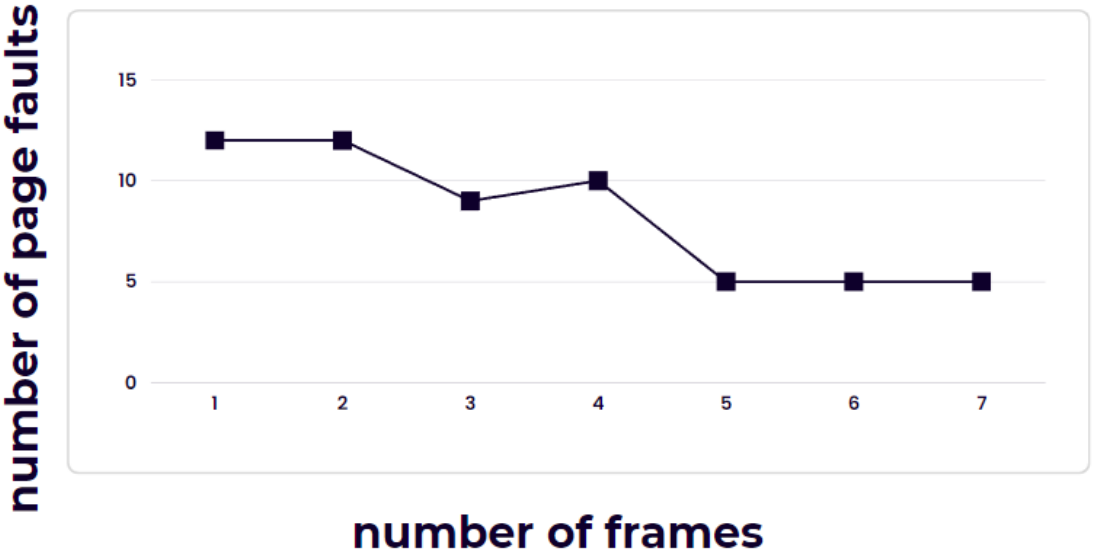
從上述兩個表中可以明顯看到，在同樣的 Reference string 時，得出的結果 pageFault 的次數一定大於 pageReplacement 的次數。因為發生 pageFault 時不一定會發生 pageReplacement。因為在最一開始時，pageFrame 為空，代表 pageFrame 中不存在目前要 reference 的 page，所以會發生 pageFault，但此時 pageFrame 裡一定有剩餘的空間，因此不會發生 pageReplacement。

四、結果與討論 ：

畢雷笛反例(Belady's Anomaly):

通常在理想情況下，增加 frame 的數目，會減少發生 pageFault 的次數，但在少數情況下，增加 frame 的數目，反而會增加發生 pageFault 和 pageReplace 的次數。而這種少數特例的情況稱為「畢雷笛反例」。在 FIFO 置換法中，在極少數的 Reference string 會發生畢雷迪反例，如上課講義中的例子:

Reference string: 123412512345



從上面的表格可以看出，在 pageFrame size 從 3 增加到 4 時，pageFault 的發生次數不減反增，和理想情況下所預期的不同。因為 FIFO 不是一種基於 Stack based 的演算法，缺乏 pageReplacement 的優先順序，導致畢雷笛反例發生。但畢雷笛反例發生只是極少數的特例。

而畢雷笛反例除了在 FIFO 置換法中出現，也會在其他置換法中出現，像是 第二次機會算法(Second chance algorithm) 、隨機頁面置換算法(Random page replacement algorithm)

五、結論:

在寫說明文件時，也花了一點時間在找 Belady's Anomaly 的例子，想知道除了上課講義中所舉的例子以外，是否還有其他的 Reference string 也會發生同樣的情況，結果找了好久，也試了很久，但都沒有找出一個結果。於是最後在寫文件說明 Belady's Anomaly 的例子時，依舊採用了老師在上課中所的範例。但這個經驗也證明了，Belady's Anomaly 的 case 真的是在極少數的情況下才會發生。

六、參考資料:

https://www.geeksforgeeks.org/beladys-anomaly-in-page-replacement-algorithms/