# 作業系統-HW3

資工三乙 11027222 黃彥霖

開發環境: Windows11, Visual Studio Code

語言: C++

**任務簡介&實作方法與流程:**總共分為 6 種任務

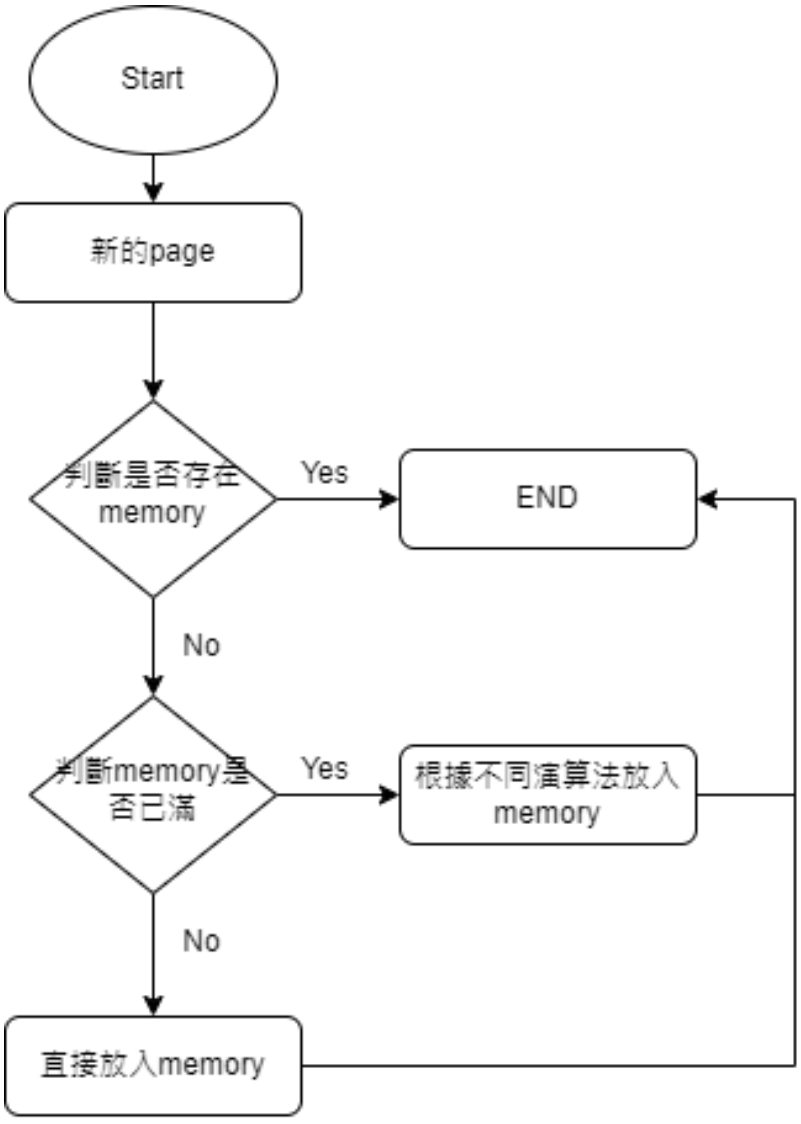
資料儲存方法:

使用 unordered_set 當作記憶體(Page frame)，會儲存當前的 pages。

使用 unordered_map 紀錄 pages 的訪問次數。

使用 queue 來記錄 pages 的進入順序。

實作流程:



Page Replacement 的不同方法:

每次執行都會記錄 Page Fault、 Page Replacement，並在最後連同 Page Frame 一起寫入檔案中。

1. FIFO (First In First Out):
   當發生 page replacement 的時候，會把最早待在 page frame 的 data 換掉，也就是佇列中 pop()的功能，再把新的 page 插入 page frame 裡面。犧牲者就是佇列當中存在最久的 page。

2. LRU (Least Recently Used):
   會換掉最久沒有使用到的 page，一樣是以佇列的方式進行，但和 FIFO 不太一樣的地方是會重新排隊，如果一個 page 再 page frame 裡面，那他會從佇列中間離開，並重新排隊，回到佇列的最後面，代表他最近被訪問過，這樣每次要選犧牲者時，佇列最前面的永遠都是最久沒被訪問的 page。
   。
3. LFU+FIFO (Least Frequently Used):
   此方法會用到 counter，一樣以佇列的方式去紀錄資料，但會順便紀錄在 page frame 中的 page 被訪問了幾次，選犧牲者的條件是把使用頻率最少的換掉，用佇列可以確保選到頻率最小和最早進來的 page，解決數到一樣頻率的情況發生。

4. MFU+FIFO (Most Frequently Used):
   此方法也會用到 counter，一樣以佇列的方式去放資料，但會順便紀錄在 page frame 中的 page 被訪問了幾次，選犧牲者的條件是把使用頻率最高的換掉，用佇列可以確保選到頻率最高和最早進來的 page，解決數到一樣頻率的情況發生。

5. LFU+LRU:
   此方法也會用到 counter，以 LFU 的基礎做延伸，一樣是選訪問頻率最小的，但加上被訪問後，加上 LRU 的特性，會重新去佇列排隊。選犧牲者的條件就是頻率使用最少和最長時間沒被使用。

6. ALL
   說明:把上述 1~5 的方法各做一次。

## 不同方法的比較:

根據範例 input1、input2 的資料，比較個方法的 Page Fault 和 Page Replacement 的次數。

(表 1) Page Fault:

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

(表 2) Page replacement:

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

從實驗結果來看，page fault 的次數都會比 page replacement 高，原因是因為要 load 進入 frame 的時候，算一次 page fault 但不會有 replacement 的情況出現，從兩個數據來看，page fault － page replacement = page frame。

從上面表格看到，LFU+LRU 平均發生 page fault 和 page replacement 的次數最少，但很難去分辨個方法的好壞，會根據輸入的情況而有所不同。並稍微討論一下各方法的優點和缺點。

## 討論各方法的優缺點:

➔ FIFO

優點:

1.容易實作和理解

缺點:

1. 不容易再頻繁切換 page 的時候有好的效能。
2. 會有 Belady’s Anomaly 的情況，當 page frame 上升反而 page fault 上升。

➔ LRU

優點:

1.最少用到的 page 進行替換，一般能有效減少頁面錯誤，相對貼近一般使用情況。

缺點:

1. 要花比 FIFO 多的效能，去維護佇列的順序和犧牲者 page 的篩選。

➔ LFU

優點:

1. 對於只用到的 page 出現一次，不會讓它長時間存在再 frame 裡面，有更多的操作空間。

缺點:

1. 額外的花費去維持紀錄使用頻率。
2. 可能會保留一些長期未被使用但頻率高的 page，而替換掉近期頻繁使用的 page，因為它沒辦法把使用頻率提高就被換掉。

➔ MFU

優點:

1. 有些 page 可能長時間待在 frame 裡已經執行完畢，可以被替換掉。

缺點:

1. 額外的花費去維持紀錄使用頻率。
2. 不符合常見使用模式，大多數情況下，頻繁使用的 page 更有可能被再次使用，MFU 可能會做出不合理的選擇。

➔ MFU

優點:

1. 結合了 LFU 和 LRU 的優點，能夠根據使用頻率和最近使用情況進行選擇，有相對較好的性能，並更能適應更多不同的情況。

缺點:

1.額外的花費去維持紀錄使用頻率，會用到非常多的紀錄，維持此方法要花費的成本比別人高很多。

2.實現相對複雜。

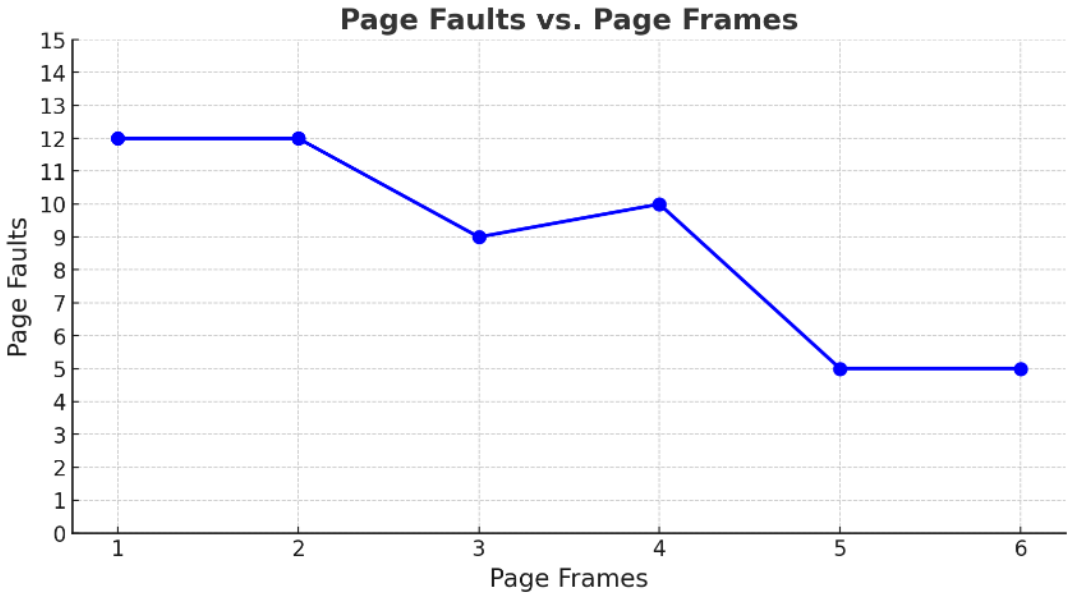
## Belady’s Anomaly 例子:

以 input1 的資料做 FIFO，找到 Belady’s Anomaly 的例子

Input: 123412512345

Page frame: 1~6

| Page frame | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Page fault | 12 | 12 | 9 | 10 | 5 | 5 |




這個例子是比較極少出現的，理論上 page frame 越大，發生 page fault 的次數就會越小。當 page frame 從 3 到 4 時，page fault 的次數不減反增，導致這個情況。

心得:

因為最佳置換方法是不可能達成的(無法預知未來)，只能從演算法達到最接近的方法，整題而言作業不難，也另外寫了一下程式去找 Belady’s Anomaly，算是輕鬆找到這個結果，不用再另外想測資，也讓我省下不少時間去慢慢改 page frame 找這個例子。