11027149 資訊三甲 游婕歆

1. 開發環境

C++

2. 實作方法

FIFO :

用 vector 實作，當有新的進來，就判斷 page 這個 vector 的大小有沒有<= Page Frame 的個數，如果等於就判斷這個字元是否已經存在在 page 這個 vector 中，不存在就選最早進來的那個字元犧牲他，把目前的加在 page 的最前面。

LRU :

用 vector 實作，當有新的進來，就判斷 page 這個 vector 的大小有沒有<= Page Frame 的個數，如果等於就判斷這個字元是否已經存在在 page 這個 vector 中，不存在就選在 page 中最早進來的那個字元犧牲他，把目前的加在 page 的最前面。如果已經存在就要更新這個字元在 page 的位置，把目前的加在 page 的最前面。

LFU_FIFO :

用 vector 實作，當有新的進來，就判斷 page 這個 vector 的大小有沒有<= Page Frame 的個數，如果等於就判斷這個字元是否已經存在在 page 這個 vector 中，不存在就選在 page 中最早進來的+count 最小的那個字元犧牲他，再把目前的加在 page 的最前面，如果已經存在就要更新這個字元 count 的數字要++。

MFU_FIFO :

用 vector 實作，當有新的進來，就判斷 page 這個 vector 的大小有沒有<= Page Frame 的個數，如果等於就判斷這個字元是否已經存在在 page 這個 vector 中，不存在就選在 page 中最早進來的+count 最大的那個字元犧牲他，再把目前的加在 page 的最前面，如果已經存在就要更新這個字元 count 的數字要++。

LFU_LRU :

用 vector 實作，當有新的進來，就判斷 page 這個 vector 的大小有沒有<= Page Frame 的個數，如果等於就判斷這個字元是否已經存在在 page 這個 vector 中，不存在就選在 page 中最早進來的+count 最小的那個字元犧牲他，再把目前的加在 page 的最前面，如果已經存在就要更新這個字元 count 的數字要++並更新這個字元在 page 的位置，把目前的加在 page 的最前面。

3. 分析不同方法之間的比較

以 input2 的結果來看

```
--------------FIFO-----------------------
Page Fault = 15  Page Replaces = 12  Page Frames = 3

--------------LRU-----------------------
Page Fault = 12  Page Replaces = 9  Page Frames = 3

--------------Least Frequently Used Page Replacement----------------------
Page Fault = 13  Page Replaces = 10  Page Frames = 3

--------------Most Frequently Used Page Replacement ----------------------
Page Fault = 15  Page Replaces = 12  Page Frames = 3

--------------Least Frequently Used LRU Page Replacement----------------------
Page Fault = 11  Page Replaces = 8  Page Frames = 3
```

Page fault

FIFO

將最早進入內存的頁面置換出去，因此
它的頁面錯誤次數可能較高。 如果一個程序的訪存模式具有局部性， FIFO 可能會經常置換出常用的頁面，導致頁面錯誤頻繁發生。

LRU

根據頁面最近被使用的時間來進行置換，較早未使用的頁面會被優先置換出去。相對於 FIFO LRU 通常能夠更好地適應程序的局部性，因此頁面錯誤次數可能會較低。

LFU

根據頁面被訪問的頻率來進行置換，使用次數最少的頁面會被優先置換出去。LFU 適合於訪存模式變化頻繁的場景，但如果某些頁面在一開始被大量訪問，然後很長時間沒有訪問，可能會導致頁面錯誤次數較高。

MFU

被頻繁使用的頁面可能是程序的重要數據或者常用代碼，因此將這些頁面保留在內存中，可以減少頁面錯誤的次數。與 LFU 相比， MFU
更加關注 頁面的使用頻率而不是使用次數。
在某些模式下， MFU 算法可能表現得比 LFU 更好，特別是對於長時間使用相同頁面的情況。然而，如果某些頁面在一開始被大量訪問，然後不再被訪問，MFU 可能會將這些頁面保留在內存中，導致頁面錯誤次數較高。

Page replace

FIFO

將最早進入內存的頁面置換出去。當內存容量不足時， FIFO 算法會頻繁地進行頁面置換，因此可能導致較高的頁面置換次數。

LRU

根據頁面最近被使用的時間進行置換。相對於 FIFO LRU 算法通常能夠更好地

利用程序的局部性，因此可能導致較低的頁面置換次數。
LFU
根據頁面被訪問的頻率進行置換。 LFU 算法在一段時間內選擇使用次數最少的頁面進行 置換，因此頁面置換次數可能較低。在某些訪存模式下，
MFU
可能表現得比其他算法更好，特別是在長時間使用相同頁面的情況下。因為 MFU 關注頁面的使用頻率而不是使用次數，所以即使一個頁面只被訪問了一次，但如果訪問頻率非常高， MFU 算法可能會保留該頁面。

4. 結果與討論

拿 input1_method1 來實驗我們可以計算頁面置換次數來觀察 Belady's Anomaly 。

Belady's Anomaly:增加分配給進程的頁框數量反而會導致更多的 Page Faults。

Frame = 3
1 1 F
2 21 F
3 321 F
4 432 F
1 143 F
2 214 F
5 521 F
1 521
2 521
3 352 F
4 435 F
5 435
Page Fault = 9 頁面錯誤總數：9 次

Frame = 4
1 1 F
2 21 F
3 321 F
4 4321 F
1 4321
2 4321
5 5432 F
1 1543 F

2 2154 F
3 3215 F
4 4321 F
5 5432 F
Page Fault = 10

當 frame 數量從 3 增加到 4 時，Page Fault 的數量從 9 次增加到 10 次，這就是 Belady's Anomaly 的一個例子。

Belady's Anomaly 並不總是在所有情況下出現，它是一種可能出現的現象。在某些情況下，增加物理內存確實可以減少頁面置換次數，但在其他情況下，增加物理內存可能不會帶來性能改善。因此，在選擇和評估頁面置換算法時，需要綜合考慮訪存模式、內存大小以及特定應用的需求。