OS\_HW3程式說明文件

1. 開發環境
2. 處理器：Intel(R) Core(TM) i7-10750H
3. 記憶體：8.00 GB
4. 系統類型：64 位元作業系統，x64 型處理器
5. Windows 規格：Windows 10 家用版
6. IDE：Visual Studio Code
7. 使用語言：C/C++
8. 實作功能

讀取檔案內Page Frame的個數，以及各個Page Reference的次序，依這些資訊模擬各種指定的Page Replacement方法，須輸出每次Page Reference時，每一個Page Frame記錄的內容，並計算每種方法之Page Fault次數以及Page Replace次數。

1. 資料結構
2. FIFO() :

\* vector< char > : buffer // 記錄Page Frame的內容

\* int : pageFault // 記錄Page Fault發生的次數，初始值為0。

\* bool : isFault // 記錄Page Fault是否發生，若是，於輸出檔案寫入F。

\* bool : stored // 記錄輪到的次序是否有在Page Frame中，若否，則發生Page Fault。

1. LRU() :

\* vector< char > : buffer //記錄Page Frame的內容

\* int : pageFault // 記錄Page Fault發生的次數，初始值為0。

\* bool : isFault // 記錄Page Fault是否發生，若是，於輸出檔案寫入F。

\* bool : stored // 記錄輪到的次序是否有在Page Frame中，若否，則發生Page Fault。

\* int : storedIndex // 記錄輪到的次序在Page Frame中的位置

1. LFU\_FIFO() :

\* vector< char > : buffer // 記錄Page Frame的內容

\* int : pageFault // 記錄Page Fault發生的次數，初始值為0。

\* bool : isFault // 記錄Page Fault是否發生，若是，於輸出檔案寫入F。

\* bool : stored // 記錄輪到的次序是否有在Page Frame中，若否，則發生Page Fault。

\* int[100] : timeStamp // 記錄每個次序進入Page Frame的時間，初始值為0。

\* int[100] : counter // 記錄每個次序進入Page Frame的次數，初始值為0。

\* bool[100] : inBuffer // 記錄每個次序是否進入過Page Frame

\* bool : full // 記錄Page Frame是否已滿

\* int : minIndex // 在Page Replacement時，可能要被交換次序的index。

\* int : minKey // 在Page Replacement時，可能要被交換次序的內容。

\* int : minCounter // 在Page Replacement時，可能要被交換次序的次數。

\* int : minTimestamp // 在Page Replacement時，可能要被交換次序的timeStamp。

1. MFU\_FIFO() :

\* vector< char > : buffer // 記錄Page Frame的內容

\* int : pageFault // 記錄Page Fault發生的次數，初始值為0。

\* bool : isFault // 記錄Page Fault是否發生，若是，於輸出檔案寫入F。

\* bool : stored // 記錄輪到的次序是否有在Page Frame中，若否，則發生Page Fault。

\* int[100] : timeStamp // 記錄每個次序進入Page Frame的時間，初始值為0。

\* int[100] : counter // 記錄每個次序進入Page Frame的次數，初始值為0。

\* bool[100] : inBuffer // 記錄每個次序是否進入過Page Frame

\* bool : full // 記錄Page Frame是否已滿

\* int : maxIndex // 在Page Replacement時，可能要被交換次序的index。

\* int : maxKey // 在Page Replacement時，可能要被交換次序的內容。

\* int : miaxCounter // 在Page Replacement時，可能要被交換次序的次數。

\* int : maxTimestamp // 在Page Replacement時，可能要被交換次序的timeStamp。

1. LFU\_LRU() :

資料結構跟LFU\_FIFO()相同。

1. MFU\_FIFO() :

資料結構跟MFU\_FIFO()相同。

1. 運作流程
2. 請使用者輸入檔案名稱，進行開檔，讀入Page Frame個數及次序，並依序呼叫指定方法的function。
3. FIFO :

首先將buffer內用’-’填滿，目的是之後判斷到是’-’字元，就表示還沒滿，可以直接放入，其次先判斷輪到的次序有沒有重複在頁框中，若沒有，代表發生Page Fault，替換時將最早進來的次序移除，最後計算總共發生Page Fault的次數以及Page Replace的次數。

1. LRU :

首先將buffer內用’-’填滿，目的是之後判斷到是’-’字元，就表示還沒滿，可以直接放入，其次先判斷輪到的次序有沒有重複在頁框中，若沒有，代表發生Page Fault，替換時先找出過去最久不被使用到的頁作移除，並將剩下次序依序往前提，再把vector的第0個設成目前的次序，完成置換的動作，最後計算總共發生Page Fault的次數以及Page Replace的次數。

1. LFU\_FIFO :

首先將buffer內用’-’填滿，目的是之後判斷到是’-’字元，就表示還沒滿，可以直接放入，其次先判斷輪到的次序有沒有重複在頁框中，若沒有，代表發生Page Fault，替換的話要先找出誰的counter值最小，代表最不常使用，因為我有一個記錄counter的陣列，就利用一個迴圈找出，那如果counter相同，就比較誰的timeStamp最小，而且這個timeStamp是設定過一次就不會再更改的，因為要符合FIFO的精神，最後計算總共發生Page Fault的次數以及Page Replace的次數。

1. MFU\_FIFO :

運作流程與LFU\_FIFO大略相同，差別在於是要先找出誰的counter值最大，代表最常使用，可以先請他離開了，而當counter相同時，也是比較誰的timeStamp最小，代表最先來過，最後計算總共發生Page Fault的次數以及Page Replace的次數。

1. LFU\_LRU :

這部分也是以LFU\_FIFO作延伸，先找出誰的counter值最小，代表最不常使用，如果counter相同，就再找出過去最久不被使用到作替換，最後計算總共發生Page Fault的次數以及Page Replace的次數。

1. MFU\_LRU :

這個function我是先複製MFU\_FIFO作雛形，再去更改當counter相同時要作的處理，因為是使用LRU，所以要找出過去最久不被使用到作替換，這個利用我的timeStamp陣列可以完成，最後計算總共發生Page Fault的次數以及Page Replace的次數。

1. 方法比較
2. input1 : n = 12, Page Frame = 3
3. input2 : n = 20, Page Frame = 3
4. 結果與討論
5. 在input1中比較Page Fault的次數，FIFO比LRU更多一點，而LFU跟MFU沒有太大差異，甚至MFU的次數跟FIFO相同，另外LFU+FIFO跟LFU+LRU結果是相同的，MFU+FIFO跟MFU+LRU也看不出差異，整體來說，可能是資料數不足，看起來較無感。
6. 在input2的結果中可以比較明顯的感受到差異，FIFO的Page Fault次數就比LRU多，LFU跟MFU也都比LRU多，LFU跟自己比較的情況下，FIFO演算法顯得比較笨重，造成次數較LRU高，MFU也是相同情形，整體而言有測出差異性，個人覺得LRU可能是最佳的演算法，不管是程式的計算量或是執行效率上，都有不錯的成效。
7. FIFO是將存在於實體記憶體頁框中最久的分頁給取代掉，實作起來最為容易。
8. LRU是將存在於頁框中最久沒用到的分頁給取代掉，實作起來比FIFO稍微困難，需儲存每個在頁框內的分頁使用後閒置的時間。
9. LFU是讓次數最少的那一頁被替換掉，因為要比較哪一個頁的次數最少，多了一次迴圈找尋的動作，所以實作上個人覺得比LRU更難一些。
10. MFU可以說就是LFU的分身，基本上換湯不換藥。