OS_HW3 結果報告資訊三乙 10727224 李敏嘉

一、 開發環境及開發平台

(語言: C++)

IDE: DEV C++

Windows 10

二、實作方法與流程

1. First In First Out (FIFO)

- 功能:

  先進先出置換法在page frame都填滿時，將最早進入page frame(也就是在主記憶體內最久的page)置換出。

- 方法流程:

  1. 將Page Reference讀入vector中，再依序將一個個page存入page frame。
  2. 當需要做頁置換時，則判斷該page(正要加入的page)是否已存在page frame中。若是，則不用改page frame；

  若否，則刪除最早加入在page frame的page，並新增page進page frame中，發生頁置換的次數+1。

  3. 在加入page frame前，若此page先前並未在page frame中，表示發生page fault，紀錄此(page fault)狀態，並將發生page fault的次數+1。
  4. 將此page的page number，目前的page frame，及page fault狀態存入output(vector)中。
  5. 繼續看下一個page，直到所有page都讀完。

2. Least Recently Used (LRU)

- 功能:

  最近罕用頁置換法在page frame滿時，將過去最久不被使用到的頁置換出。

- 方法流程:

  1. 將Page Reference讀入vector中，再依序將一個個page存入page frame。
  2. 若page frame尚未滿(不需作頁置換時)，則依序加入page frame中。若已在page frame中的page又被reference到時，就會將其頁從page frame (vector)中原本的位置中取出，並重新加入page frame (vector)中(表示更新時間標記)。
  3. 當需要做頁置換時，則判斷該page(正要加入的page)是否已存在page frame中。若是，則將其頁從page frame (vector)中原本的位置中取出，並重新加入page frame (vector)中(表示更新時間標記)。

  若否，則刪除過去最久沒有被reference到的頁。並新增新的page進page frame中，發生頁置換的次數+1。

  4. 在加入page frame前，若此page先前並未在page frame中，表示發生page fault，紀錄此(page fault)狀態，並將發生page fault的次數+1。

5. 將此page的page number，目前的page frame，及page fault狀態存入output(vector)中。

6 . 繼續看下一個page，直到所有page都讀完。

3. Least Frequently Used (LFU) + FIFO

- 功能:

  最不常使用頁置換法+先進先出置換法:

  每一個頁框都有一個對應的counter，起始值為0，當某個頁框被參考到的話，counter就會+1。需要做頁置換時，則選counter值最小(且待在主記憶體中最久)的頁框作頁置換。

- 方法流程:

  1. 將Page Reference讀入vector中，再依序將一個個page存入page frame。
  2. 若是某個頁框被參考到，則其對應的counter+1。
  3. 需要作頁置換時，則判斷該page(正要加入的page)是否已存在page frame中。

  若是，則不用改page frame；

  若否，則刪除counter最小(且最早加入在page frame)的page frame，並新增page進page frame中，發生頁置換的次數+1。

  4. 在加入page frame前，若此page先前並未在page frame中，表示發生page fault，紀錄此(page fault)狀態，並將發生page fault的次數+1。
  5. 將此page的page number，目前的page frame，及page fault狀態存入output(vector)中。
  6 . 繼續看下一個page，直到所有page都讀完。

4. Most Frequently Used (MFU) + FIFO

- 功能:

  最常使用頁置換法+先進先出置換法:

  每一個頁框都有一個對應的counter，起始值為0，當某個頁框被參考到的話，counter就會+1。需要做頁置換時，則選counter值最大(且待在主記憶體中最久)的頁框作頁置換。

- 方法流程:

  1. 將Page Reference讀入vector中，再依序將一個個page存入page frame。
  2. 若是某個頁框被參考到，則其對應的counter+1。
  3. 需要作頁置換時，則判斷該page(正要加入的page)是否已存在page frame中。

  若是，則不用改page frame；

  若否，則刪除counter最大(且最早加入在page frame)的page frame，並新增page進page frame中，發生頁置換的次數+1。

  4. 在加入page frame前，若此page先前並未在page frame中，表示發生page fault，紀錄此(page fault)狀態，並將發生page fault的次數+1。
  5. 將此page的page number，目前的page frame，及page fault狀態存入

output(vector)中。

6. 繼續看下一個page，直到所有page都讀完。

5. LFU + LRU

- 功能:

  最不常使用頁置換法+最近罕用頁置換法:

  每一個頁框都有一個對應的counter，起始值為0，當某個頁框被參考到的話，counter就會+1。需要做頁置換時，則選counter值最小(且最近罕用)的頁框作頁置換。

- 方法流程:

  1. 將Page Reference讀入vector中，再依序將一個個page存入page frame。
  2. 若是某個頁框被參考到，則其對應的counter+1。
  3. 需要作頁置換時，則判斷該page(正要加入的page)是否已存在page frame中。

  若是，則將其頁從page frame (vector)中原本的位置中取出，並重新加入page frame (vector)中(表示更新時間標記)。

  若否，則刪除counter最小(且最近罕用)的page frame，並新增page進page frame中，發生頁置換的次數+1。

  4. 在加入page frame前，若此page先前並未在page frame中，表示發生page fault，紀錄此(page fault)狀態，並將發生page fault的次數+1。
  5. 將此page的page number，目前的page frame，及page fault狀態存入output(vector)中。
  6. 繼續看下一個page，直到所有page都讀完。

6. MFU + LRU

- 功能:

  最常使用頁置換法+最近罕用頁置換法:

  每一個頁框都有一個對應的counter，起始值為0，當某個頁框被參考到的話，counter就會+1。需要做頁置換時，則選counter值最大(且最近罕用)的頁框作頁置換。

- 方法流程:

  1. 將Page Reference讀入vector中，再依序將一個個page存入page frame。
  2. 若是某個頁框被參考到，則其對應的counter+1。
  3. 需要作頁置換時，則判斷該page(正要加入的page)是否已存在page frame中。

  若是，則將其頁從page frame (vector)中原本的位置中取出，並重新加入page frame (vector)中(表示更新時間標記)。

  若否，則刪除counter最大(且最近罕用)的page frame，並新增page進page frame中，發生頁置換的次數+1。

  4. 在加入page frame前，若此page先前並未在page frame中，表示發生page fault，

紀錄此(page fault)狀態，並將發生page fault的次數+1。

5. 將此page的page number，目前的page frame，及page fault狀態存入output(vector)中。

6 . 繼續看下一個page，直到所有page都讀完。

資料結構 ：

使用vector，存(input) page reference 跟(output) 要印出的答案。

Vector-Output: 存每一page的處理後的資料，其中包含

1 . pageNum – 該page的編號，為integer

2. counter – 該page出現的次數(表示在記憶體中存多久)，為integer

3. pageFrame – 目前的頁框，pageFrame也是用vector存，裡面存的是在頁框中的page(表示正在記憶體中的page) (見第5點說明)

4. status – 以string來存，若發生page fault，則將此狀態記為”F”

5. pageFrame(每一頁的資料): 目前的頁框，除了存pageNum(integer)外，也存了counter(integer)，在需要紀錄page出現次數時，便更新counter。

使用vector原因:

因為考慮到不確定page有多少，被reference到的又有多少，所以選擇好操作又不用擔心大小的vector。

看了第一題FIFO原本有想過用Queue來做page frame，因為先進先出本來就是Queue的特性，第二題LRU也能用Queue來做，但是看了第三和四題LFU, MFU，在page frame中，若要一個一個比較其counter，用Queue就會特別麻煩(也不是最佳方式)，反而還是vector更好，所以最後整體就選擇vector來做page frame。

三、結果比較: (前兩個(input1&input2)page frame數都是3)

input1	FIFO	LRU	LFU+FIFO	MFU+FIFO	LFU+LRU	MFU+LRU
Page Fault 次數	9	10	10	9	10	9
Page Replace 次數	6	7	7	6	7	6
input2						
Page Fault 次數	15	12	13	15	11	12
Page Replace 次數	12	9	10	12	8	9
Input1 (page frame 數=4)						
Page Fault 次數	10	8	8	10	8	10
Page Replace 次數	6	4	4	6	4	6
Input2 (page frame 數=4)						
Page Fault 次數	10	8	9	12	9	9
Page Replace 次數	6	4	5	8	5	5

FIFO:

將待在主記憶體中最久的page置換，但是可能會將經常被reference的頁置換出，所以可能會造成page fault及頁置換次數提高。若將page frame數提高，不一定能改善page fault發生的狀況，甚至可能提高page fault次數，(畢雷笛反例)。

LRU:

若提高其page frame數，則可以減少發生page fault及頁置換的次數。LRU不會出現像是FIFO一樣的畢雷笛反例(Belady's Anomaly)。

LFU :

最不常使用頁置換法，此方法將計數器值較小的頁置換，但因為剛進入的頁，counter值都很小，所以取代時，可能會取代新進的頁，如此就會造成取代錯誤的情形發生。

搭配FIFO: 若有多個頁框的counter值(最小的)相同，則替換掉帶在記憶體中最久的page，也因為先比較counter值，所以跟單純的FIFO比起來，能減少page fault及頁置換的次數。

搭配LRU: 原本LRU的方式就跟FIFO的方式差不多，甚至能減少page fault發生次數，所以LFU搭配LRU也是取了各自的優點，進而降低page fault發生次數。

MFU:

最常使用頁置換法，此方法會將計數器最大的那個頁框作頁置換，原因是因為他待在記憶體很久，很可能已經執行完畢了，不過待很久的那一頁也可能是非常重要的頁，需要一直被reference到，所以如果換掉他，很可能之後又會被reference到，造成page fault，需要再置換的問題發生。

由input1和input2的例子來看，其實這並不是一個很好的方法，跟LFU比起來，MFU似乎更容易發生page fault。而且當頁框增加時，也可能會發生畢雷笛反例。

四、未完成的功能:

無