10520104 電資四 莊東翰

1.開發環境:
Windows10 64位元 家用版
開發平台:
DEV C++
TDM-GCC 4.9.2 64-bit
程式語言:C++

2.功能和流程:

FIFO	先進來的page優先被swap out
LRU	近期最少被使用的page優先被swap out
LFU	Counter最少的page優先被swap out
MFU	Counter最多的page優先被swap out

比較:
FIFO vs LRU:
這兩種方法只要page的排序方式是依照最近使用的順序排序，優先被swap out的page都是最底層的page。這兩個方法的差異主要是當某個page要被使用，且本來就存在memory時，FIFO不改變每個page在queue中的順序，而LRU必須把此page抽換至最頂層(最近使用)。

LFU vs MFU:
只要在queue中依照執行順序排序，從最上層開始往下層找，分別記錄MFU: counter最大的page 或 LFU: counter最小的page，有相同counter更底層的page就優先做swap out。

FIFO: 按照reference順序依序放入memory中，若滿了則以memory中最早來的page優先swap out。

LRU: 按照reference順序依序放入memory中，最近執行完的page要排在queue的最上面。Swap out時優先選最底部的page。

LFU+FIFO: 選擇swap out的page選擇順序:

1. counter最少的page
2. 若有多個一樣少的page則選最下面的。

MFU+FIFO:
選擇 swap out 的 page 選擇順序:

1. counter 最多的 page
2. 若有多個一樣多的 page 則選最下面的。

LFU+LRU:
最新被執行的 page 必須排到最上層。
選擇 swap out 的 page 選擇順序:

1. counter 最少的 page
2. 若有多個一樣少的 page 則選最下面的。

MFU+LRU:
最新被執行的 page 必須排到最上層。
選擇 swap out 的 page 選擇順序:

1. counter 最多的 page
2. 若有多個一樣多的 page 則選最下面的。

3. 資料結構:

```
struct CountType{
  string page;
  int freq;
};
```

每個 Page 都要記錄此 page 在 memory 中被使用的次數，如果此 page 被 swap out 則下次被 swap in 時要重新從次數 1 開始算。

```
struct PageType{
  int page_fault;
  int page_replace;
};
```

每種功能都需要紀錄 page fault, page replace 次數並回傳。

vector<CountType> memory
每種方法都需要有個 queue 紀錄執行順序。

4.結果與討論:

Page fault 的發生不見得是因為 Place replacement 發生(memory 沒滿時 page fault 不代表發生 Place replacement)。

Place replacement 的發生一定伴隨著 Page fault。

out_input1.txt:

reference string: 1234125123455

frame size: 3

(fault, replace)	FIFO	LRU
FIFO	(9,6)	X
LRU	X	(10,7)
LFU	(10,7)	(10,7)
MFU	(9,6)	(9,6)

造成兩種結果的差異主要是因為最後的 5，FIFO 剛好最後因為 5 還沒被剔除，所以可以省下一次 page replacement。MFU 會剔除 counter 數多的 page，剛好 5 這個 page 在先前的 counter 不是最高沒被剔除，且現在仍在 memory 中，省下一次 page replacement。