作業系統-書面報告

資訊三甲 11127138 林雨臻

指導老師：鍾武君

一、 開發環境：

1. **硬體設備**：：
   - 處理器 (CPU)：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
   - 記憶體 (RAM)：16 GB
   - 作業系統：Windows
2. **開發工具**：
   - 編譯器：g++ (版本 12.2.0)
   - 程式編輯器：vscode (Visual Studio Code)

二、 實驗方法和流程

1. FIFO_Algorithm:
   - 說明：當 page frame 已滿且需要替換時，移除佇列中的第一個進入的 page。
   - 資料結構：
     a. queue<int> pageQueue; // page frame
     b. unordered_map<int, bool> Page_Records; // page -> isExist
   - 實作方法：
     a. 如果 page 不在 frame 中：
        i. pageFaults++
        ii. 如果 frame 已滿，移除 pageQueue 的第一個 page，並在 Page_Records 中刪除此 page。
        iii. 移除選定 page，重設其資料，並 pageReplaces++。
        iv. 將新 page 加入 pageQueue 並在 Page_Records 中記錄。
     b. 如果頁面已存在於 frame 中：
        i. 保持原先排序。
2. LRU_Algorithm:
   - 說明：當 page frame 已滿且需要替換時，移除佇列中最久未使用的 page。
   - 資料結構：
     a. unordered_map<int, int> pagePosition; // page -> timestamp
   - 實作方法：
     a. 如果 page 不在 frame 中：
        i. pageFaults++
        ii. 如果 frame 滿了，使用 find_oldest() 找出最近最久未使用的 page 並將其移除。
        iii. 在 frame 中插入新 page 並記錄當前時間戳。
     b. 如果頁面已存在於 frame 中：

i. 更新此 page 的時間戳。
ii. 依照最近使用的時間排序。

3. LFU_and_FIFO_Algorithm:
   - 說明：當 page frame 已滿且需要替換時，移除佇列中 frequency 最少的，若 frequency 相等，則移除最先進入佇列的 page，並將 frequency 歸零等待下次進入重新計算。
   - 資料結構：
     a. unordered_map<int, page_record> pageRecords; // page -> (in_page_frame_flag, count, timestamp)
     b. set<pair<int, pair<int, int>>, Compare> pageFrames_data; // <page, <count, timestamp>>
   - 實作方法：
     a. 如果 page 不在 frame 中：
        i. pageFaults++
        ii. 如果 frame 滿了，使用 find_least_frequent() 找出使用次數最少且最早進入 frame 的 page。
        iii. 從 pageFrames_data 中移除選中的 page，重設其資料，並 pageReplaces++。
        iv. 將新的 page 加入 frame，更新其使用次數為 1 並記錄當前時間戳。
     b. 如果頁面已存在於 frame 中：
        i. 更新此 page 的時間戳。
        ii. 在 pageFrames_data 中移除舊資料，插入更新後的 page 資料。

4. MFU_and_FIFO_Algorithm:
   - 說明：當 page frame 已滿且需要替換時，移除佇列中 frequency 最多的，若 frequency 相等，則移除最先進入佇列的 page，並將 frequency 歸零等待下次進入重新計算。
   - 資料結構：
     a. unordered_map<int, page_record> pageRecords; // page -> (in_page_frame_flag, count, timestamp)
     b. set<pair<int, pair<int, int>>, Compare> pageFrames_data; // <page, <count, timestamp>>
   - 實作方法：
     a. 如果 page 不在 frame 中：
        i. pageFaults++
        ii. 如果 frame 已滿，使用 find_most_frequent() 尋找使用次數最多的 page。
        iii. 從 pageFrames_data 中移除選中的 page，重設其資料、pageReplaces++。
        iv. 將新 page 加入 frame，更新其資料。
     b. 如果頁面已存在於 frame 中：
        i. 更新 page 的使用次數。
        ii. 在 pageFrames_data 中移除舊資料，插入更新後的 page 資料

5. LFU_and_LRU_Algorithm:
   - 說明：當 page frame 已滿且需要替換時，移除佇列中 frequency 最少的，若 frequency 相等，移除佇列中最久未使用的 page。

- 資料結構：
  a. unordered_map<int, page_record> pageRecords; // page -> (in_page_frame_flag, count, timestamp)
  b. set<pair<int, pair<int, int>>, Compare> pageFrames_data; // <page, <count, timestamp>>
- 實作方法：
  a. 如果 page 不在 frame 中：
     i. 增加 page 錯誤計數 (pageFaults++)
     ii. 如果 frame 已滿，呼叫 find_least_frequent_and_recently() 尋找最少使用且最久未使用的 page。
     iii. 移除選定 page，重設其資料，並 pageReplaces++。
     iv. 將新 page 加入 frame，更新其資料。
  b. 如果頁面已存在於框中：
     i. 更新 page 的使用次數和時間戳。
     ii. 為了使輸出順序與老師一致，在 pageFrames_data 中移除舊 page，插入更新後的 page。

三、 探討結果和原因

1. **畢雷笛反例(Belady's Anomaly)：**

定義：某些頁面替換算法中，當 page frames 增加時，page fault 反而增多的現象。通常發生在 FIFO（First In First Out）替換方法中，因為它只考慮 page 進入的先後順序，忽略了 page 使用的頻率或時間。

在 FIFO 中，增加 page frames 數量通常會預期減少 page faults，因為有更多的空間可以存放 page。但從下方兩張圖可以看出，使用 FIFO 時，當 page frames 從 3 增加到 4 ，page fault 反而從 9 增加到 10。

----------FIFO----------

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 Page Replaces = 6 Page Frames = 3

----------FIFO----------

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 Page Replaces = 6 Page Frames = 4

2. 比較五種方法間 **pageFaults、pageReplace 的差異** (page 輸入：123412512345)：

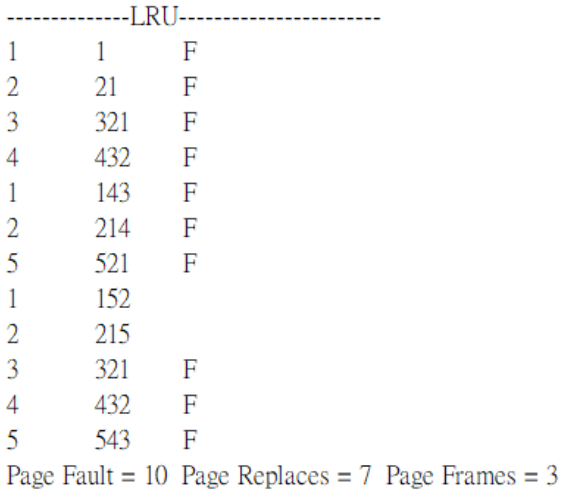
方法 **(PAGE FRAMES = 3)**	**PAGE FAULTS**	**PAGE REPLACES**
FIFO	9	6
LRU	10	7
LFU + FIFO	10	7
MFU + FIFO	9	6
LFU + LRU	10	7

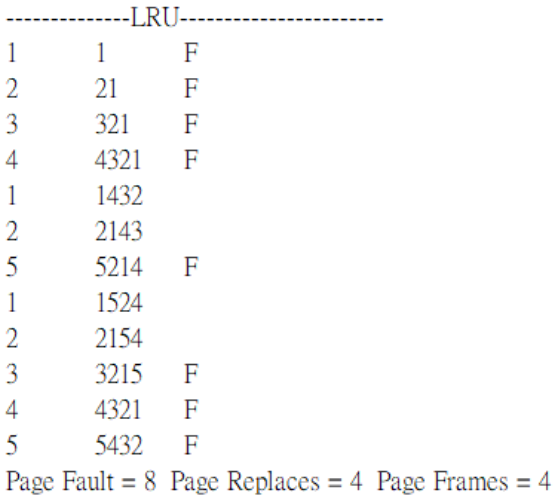
(表一)

方法 **(PAGE FRAMES = 4)**	**PAGE FAULTS**	**PAGE REPLACES**
FIFO	10	6
LRU	8	4
LFU + FIFO	8	4
MFU + FIFO	10	6
LFU + LRU	8	4

(表二)

I. FIFO：因為不考慮 page 的使用頻率與時間，pageFaults 和 pageReplaces 相對較高。
II. LRU：尋找最近沒使用的 page 進行替換，適合每種 page 與任務執行階段高度關聯的情況，如此才可降低替換的發生次數。
III. LFU + FIFO：結合使用頻率與先進先出，能有效處理常用 page，但在每個 page 的頻率變化大的情況下性能可能下降，因為仍會頻繁發生置換。
IV. MFU + FIFO：認為頻率高的 page 可能已經使用完畢，但我認為此法並不特別符合真實場景，就我的常理而言除非每種 page 都有限制相同的總數，否則我會認為目前出現次數最高的 page，未來的出現次數也會更高。
V. LFU + LRU：綜合考慮使用頻率與最近使用時間，最佳方案。





----------Least Frequently Used Page Replacement----------

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	321	F
4	421	F
5	521	F

Page Fault = 10 Page Replaces = 7 Page Frames = 3

----------Least Frequently Used Page Replacement----------

1	1	F
2	21	F
3	321	F
4	4321	F
1	4321	
2	4321	
5	5421	F
1	5421	
2	5421	
3	3521	F
4	4321	F
5	5421	F

Page Fault = 8 Page Replaces = 4 Page Frames = 4

----------Most Frequently Used Page Replacement ----------

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 Page Replaces = 6 Page Frames = 3

----------Most Frequently Used Page Replacement ----------

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 Page Replaces = 6 Page Frames = 4

----------Least Frequently Used LRU Page Replacement----------

1	1	F
2	21	F
3	321	F
4	432	F
1	143	F
2	214	F
5	521	F
1	152	
2	215	
3	321	F
4	421	F
5	521	F

Page Fault = 10 Page Replaces = 7 Page Frames = 3

----------Least Frequently Used LRU Page Replacement----------

1	1	F
2	21	F
3	321	F
4	4321	F
1	1432	
2	2143	
5	5214	F
1	1524	
2	2154	
3	3215	F
4	4321	F
5	5421	F

Page Fault = 8 Page Replaces = 4 Page Frames = 4