Operating System Project 3

Page Replacement

資訊三乙 10727211 林彥輝

一、開發環境:

作業系統: Windows 10 Enterprise x64

使用語言: Python 3.8

測試環境:

- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (12 CPUs),~2.6GHz
- RAM: Micron DDR4 2666/32GB*1
- SDD: 1TB PCIe SSD
- OS: Windows 10 Enterprise x64
- IDE: PyCharm Community Edition 2020.2.2

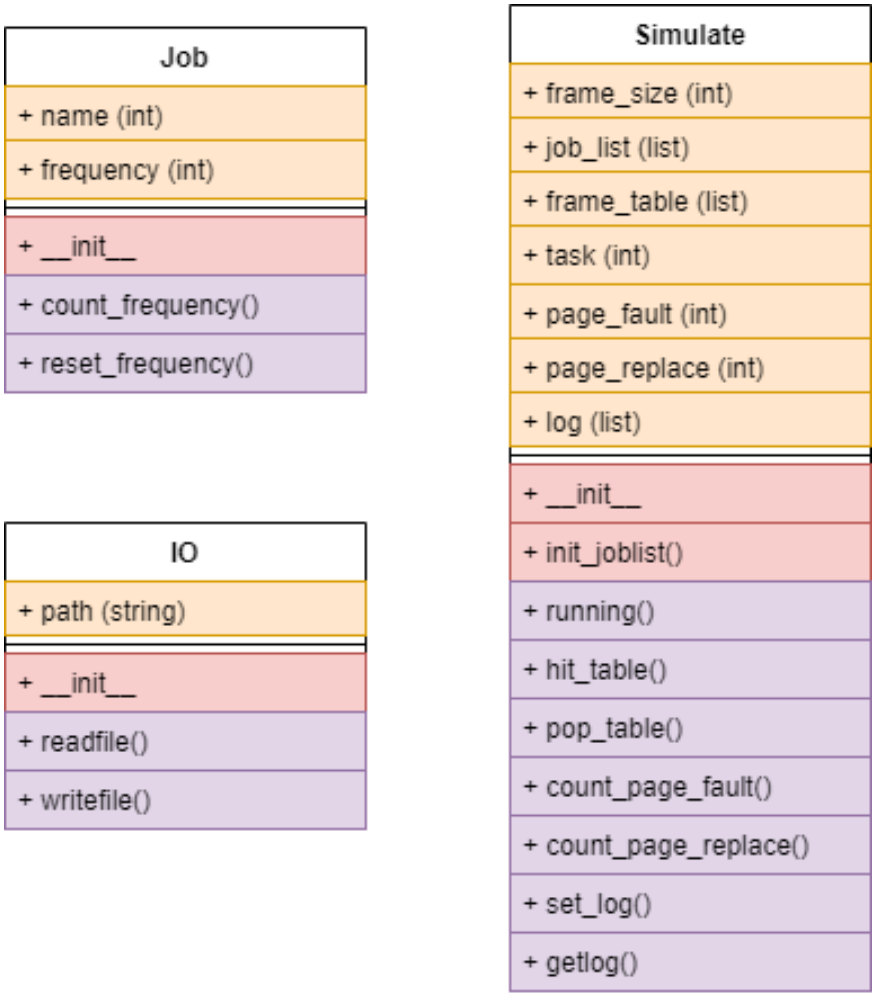
二、實作功能

本次 Project 中實作以下 CPU Scheduling 之功能：

1. First Come First Serve (FCFS)
2. Least Recently Used (LRU)
3. Least Frequently Used (LFU)+(FIFO)
4. Most Frequently Used (MFU)+(FIFO)
5. LFU + LRU
6. MFU + LRU

三、實作流程

實作模擬 Page Replacement 之 Data Structure 如下：



※正式名稱與變數名稱格式為：變數名稱(正式名稱)。

※以下解說將統一使用正式名稱說明。

Class Job：

模擬一個 Page 的所有資訊，包括 name(Page reference)、frequency (Counter)、，這些欄位提供 Object 在排程計算時有良好的評斷比較。

Class Simulate：

設計策略為 Simulate()為一種方法的模擬，故在 main()中需要呼叫六次 Simulate()，而使用參數(對應到 Simulate.task)分別代表不同的方法，完成這次的功能。

提供一系列變數：frame_size(page frame 數)、job_list(input 之 page reference 存放處)、task(對應 Project 的 6 個功能)、frame_table(page frame)、page_fault(page fault 數)、

page_replace(page replaces 數)、log(存放單個方法模擬之個時段 page reference、page frame、page fault，供輸出用)。

提供一次排程所需的所有計算 Function，

- running() ： 為 Simulate 的核心 Function，先使用 hit_table() 查看使否該 Page reference 在 page frame 中存在，若不存在(Not hit)，則進行 pop_table()進行 replacement。
  - hit_table()呼叫方式

```
            if task is 1 or task is 3 or task is 4:
                have_hit = self.hit_table(instr='FIFO', name=i.name)
            elif task is 2 or task is 5 or task is 6:
                have_hit = self.hit_table(instr='LRU', name=i.name)
            else:
                print('error')
```

  - pop_table()呼叫方式

```
            if have_hit is False:
                if task is 1 or task is 2:
                    self.pop_table(instr='formal')
                elif task is 3 or task is 5:
                    self.pop_table(instr='LFU')
                elif task is 4 or task is 6:
                    self.pop_table(instr='MFU')
                else:
                    print('error')
```

- hit_table() ： 根據參數指定的不同指令(FIFO、LRU)，兩者指令接先尋找 page frame 中是否有欲尋找的 page reference，差異 LRU 會更新 page frame 的順序，若在 page frame 中找不到該 page reference，則更新 page fault 次數。
- pop_table():會先根據 page frame 是否已被填滿，若滿根據參數指定的不同指令(formal、LFU、MFU)，三者差異在於，formal 直接刪除 page frame 中尾端之 page，LFU、MFU 則會使用迴圈找出 page frame 中之 frequency 數最多或最少者，根據方法之條件做刪除。
- count_page_fault():將 page_fault 次數加一。
- count_page_replace():將 page_replace 次數加一。
- set_log():設定 log，內容即為單次虛擬之所有資訊。實作方式為碰到 log 為空時，先增加標頭，之後再增加 page reference、page frame、page fault 符號。

- get_log():目的取出 log，實作方式為將 log 加入最後 page fault、page replaces、page frame 次數再 return。
  - 單次 Simulate 結果之 log 內容範例。

```
-------------FIFO----------------------
['1', ['1'], 'F']
['2', ['2', '1'], 'F']
['3', ['3', '2', '1'], 'F']
['4', ['4', '3', '2'], 'F']
['1', ['1', '4', '3'], 'F']
['2', ['2', '1', '4'], 'F']
['5', ['5', '2', '1'], 'F']
['1', ['5', '2', '1'], 'T']
['2', ['5', '2', '1'], 'T']
['3', ['3', '5', '2'], 'F']
['4', ['4', '3', '5'], 'F']
['5', ['4', '3', '5'], 'T']
Page Fault = 9  Page Replaces = 6  Page Frames = 3
```

四、不同方法比較

- 以平均頻率為基準(相同 page 出現間隔 > page frame 數)
  - FIFO：**適合於平均出現頻率低者**

    註記：最簡單的實作方法，但也容易是表現最差的演算法。
  - LRU ：**適合於平均出現頻率高者**
- 以短期頻率為基準(出現 page 間隔 <= page frame 數)
  - LFU：**適合於短期出現頻率高者**

    註記：短期內重複出現機率高之 page，搭配 LRU，演算法表現最佳。
  - MFU：**適合於短期出現頻率低者**

    註記：page frame 中出現頻率高者可能是相當重要之資料，替換可能付出龐大的代價，如更多的 Page Fault，甚至溢位。

- Belady's Anomaly：Page Fault 一定不小於 page frame 內之 Page 數，因為每一 Page 在第一次擺進 page frame 一定會發生 Page Fault，因此調大 page frame 之尺寸，成效未必得到成長。(雖然在測資中並未觀測到此狀況)

$$\text{Page replaces} = \begin{cases} 0, & \text{if (Page Fault} - \text{Fage frames} < 0) \\ \textit{Page Fault} - \textit{Page Frames}, & \textit{else} \end{cases}$$

五、困難處與心得

整體來說，這次的作業比起前兩次作業來說困難度調降了許多，實作上只需一堂演算法的上課時間就能完成，因此剛好有個機會讓自己嘗試不同方式的 coding style，因為各種演算法過程中的計算過於相似，因此透過減少 Function 的方式，讓 Function 透過參數在關鍵時刻決定要執行何種指令，結果讓自己的程式碼短而精簡，是個蠻不錯的一次嘗試。

另外碰到的 Bug 在於計算 frequency 時，起初認為 frequency 的定義是一個 frame reference 在整體出現的次數或者在累積參考的次數，後來發現都不是，是擺進 page frame 後先做初始化，之後再做累積的動作。這是如果沒有透過實作不會發現的小細節。