AOS HW1 M083040017 蘇品瑜

# test reference string

1. Random

import random

len\_rs = 100000 # reference string’s string

rs\_tc1 = [] # reference string

dw\_set = [] # disk write set

for i in range(len\_rs):

    rs\_tc1.append(random.randint(1,500))

    dw\_set.append(random.randint(0,1)) # 0: no change,1: dirty

1. Locality

import random

len\_rs = 100000 # reference string’s string

rs\_tc1 = [] # reference string

dw\_set = [] # disk write set

while len(rs\_tc1) < len\_rs:

    begin = random.randint(1,500)

    maxApend = 500 - begin  # 還可放幾個的扣打

    if maxApend >= 2:

        add = random.randint(1,maxApend)

        if len(rs\_tc1) >= len\_rs :

            break

        rs\_tc1.append(begin)

        dw\_set.append(random.randint(0,1))

        for i in range(add):

            begin = begin + 1

            if len(rs\_tc1) >= len\_rs:

                break

            rs\_tc1.append(begin)

            dw\_set.append(random.randint(0,1))

    elif maxApend==2:

        if len(rs\_tc1) >= len\_rs:

            break

        rs\_tc1.append(499)

        dw\_set.append(random.randint(0,1))

        if len(rs\_tc1) >= len\_rs:

            break

        rs\_tc1.append(500)

        dw\_set.append(random.randint(0,1))

    else:

        rs\_tc1.append(500)

        dw\_set.append(random.randint(0,1))

1. Custom

Why?

搭配 Custom Algorithm 特性，還有和FIFO特性唱反調(後進先出)。

Custom Algorithm特性:

先考慮 reference 在 frame 的累積出現次數，但一旦被替換，則重新計算。

如果一樣，再考慮 誰最近沒被使用。

最後 進行 replacement。

import random

rs\_tc1 = [] # reference string

len\_rs = 100000 # reference string’s string

dw\_set = [] # disk write set

for i in range(len\_rs):

    dw\_set.append(random.randint(0,1))

sum = 0 # 第幾個 reference

start = 50 # 一次 (塞(50-1) + 50++)

while sum < len\_rs:

    for i in range(1,50):

        rs\_tc1.append(i)

        sum = sum + 1

    rs\_tc1.append(start)

    start = start + 1

    sum = sum + 1

# Algorithm x reference string = 36 figures & behaviors

1. FIFO

先進來的page，在發生replacement時，先被替換出去。

If page not in memory , page fault ++.

If replacement occurs(swap out), interrupt++.

If victim page’s dirty bit is 1, disk write++.

import queue

for i in range(10):

    frame\_number = input("Number of frame: ")

    myqueue = queue.Queue(int(frame\_number)) # 設定 frame 數量

    page\_fault\_sum = 0 # 計算 page fault 數量

    interrupt = 0 # 計算 interrupt 數量

    disk\_write = 0 # 計算 disk write 數量

    cnt = 0 # reference string 到第幾個

# 先塞滿 memory

    while myqueue.full()!=True :

        myqueue.put(random\_rs[cnt]) # 將 reference 放入 memory

        cnt = cnt + 1

        page\_fault\_sum = page\_fault\_sum + 1

    print('page\_fault\_sum:',page\_fault\_sum) # print 目前 page fault 數量

    # 塞滿了，要塞下一個 先看 page 是否在 queue 裡

    # 有 則 hit , 沒有 則 miss

    while cnt != len(random\_rs) :

        if random\_rs[cnt] in myqueue.queue:

            pass # 不做任何事

        else:

            interrupt = interrupt + 1 # swap out occurs

            page\_fault\_sum = page\_fault\_sum + 1

            if dw\_set[random\_rs.index(myqueue.get())] == 1 : # modify

                disk\_write = disk\_write + 1

            myqueue.put(random\_rs[cnt]) # 放入新進的 reference

        cnt = cnt + 1

    print('page\_fault\_sum: ',page\_fault\_sum)

    print('interrupt: ',interrupt)

    print('disk write: ',disk\_write)

>fifo\_Random\_rs.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



>fifo\_Locality\_rs.py

一張含有 文字 的圖片

自動產生的描述

一張含有 文字, 地圖 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述



|  |  |  |  |
| --- | --- | --- | --- |
|  | Page Fault | Interrupt | disk write |
| 10 | 99903 | 99893 | 46755 |
| 20 | 99710 | 99690 | 46674 |
| 30 | 99321 | 99291 | 46485 |
| 40 | 98642 | 98602 | 46146 |
| 50 | 98123 | 98073 | 45905 |
| 60 | 97481 | 97421 | 45600 |
| 70 | 97083 | 97013 | 45416 |
| 80 | 96204 | 96124 | 44996 |
| 90 | 95128 | 95038 | 44500 |
| 100 | 94406 | 94306 | 44180 |

>fifo\_custom.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



1. Optimal

假設可以預知未來的reference，當發生替換時，則選擇未來不會用或是較晚才會用到的page，作為victim page。

for i in range(10):

    fn\_tc1 = input("輸入frame數: ")

    fn\_tc1 = int(fn\_tc1)

    hits\_tc1 = 0

    misses\_tc1 = 0

    frame = []

    frame\_record = [] # 紀錄 未來出現情況

    # rs 每往前一步 距離-- , 只更新新進來的page 距離(不需重新計算)

    cnt   = 0 # 第幾個 frame

    index = 0 # rs\_tc1's index

    page\_fault = 0

    interrupt  = 0

    dw = 0

    # 塞滿 memory

    while cnt <  fn\_tc1 :

        frame.append(rs\_tc1[index])

        frame\_record.append(int(0)) # 使得 與 frame 數量一致，default 放0

        cnt = cnt + 1

        page\_fault = page\_fault + 1

# 統計 距離下次 reference 出現，如果沒有設 100000+1

        for i in range(len(frame)):

            if frame[i] in rs\_tc1[index+1:len(rs\_tc1)]:

                frame\_record[i] = rs\_tc1[index+1:len(rs\_tc1)].index(frame[i]) - index

            else:

                frame\_record[i] = (100000+1)

        index = index + 1

    print(page\_fault) # 顯示目前 page fault 數

    while index <= (len(rs\_tc1) - 1):

        flag = 0

        if rs\_tc1[index] in frame:

            pass # do nothing

        else:

            # 挑一個 距離最遠的  # 記得 挑完，全部距離--

            victim\_page\_index = frame\_record.index(max(frame\_record))

            if dw\_set[rs\_tc1.index(frame[victim\_page\_index])] == 1:

                dw = dw + 1

# 全部距離 --

            for k in range(fn\_tc1):

                frame\_record[k] = frame\_record[k] - 1

# 取代

            frame[victim\_page\_index] = rs\_tc1[index]

            # 計算新進來 的 page 的距離

# 因為 已取代，所以 frame[victim\_page\_index] 代表 新進frame

            if frame[victim\_page\_index] in rs\_tc1[index+1:len(rs\_tc1)]:

                frame\_record[victim\_page\_index] = rs\_tc1[index+1:len(rs\_tc1)].index(frame[victim\_page\_index]) - index

            else:

                frame\_record[victim\_page\_index] = (100000+1)

            page\_fault = page\_fault + 1

            interrupt = interrupt + 1

        index = index + 1

    print('Final Page Fault: ',page\_fault)

    print('interrupt: ',interrupt)

    print('disk write: ',dw)

>optimal\_rrs.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



>optimal\_lrs.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面, 天空 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



>optimal\_custom.py

一張含有 文字 的圖片

自動產生的描述

一張含有 文字, 地圖 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



1. ESC (circular)

Second chance 進化版。

會有個 pointer ，輪流指frame，被指到的且reference bit 為0，則替換。

Ex : frame 數為3 ，指的順序為 0,1,2,0,1,2…以此類推

第一次進去memory時，reference bit 設0，如果連續出現，reference bit設1。

如果是1，得到一次 pass 權(可不被替換掉)。

如果 pointer 指到 ，且 reference bit 為 1，pass，但reference 被設為0。

多了一個 modify bit 要記錄。

Random 產生 reference 對應的 dirty bit，If is 1, modify. If is 0, no change.

[程式運作]

從 index 0開始，找 reference bit 為 0 且 dirty bit 為 0 的情況，優先取代。

有找到則 reference string 換下一位。

沒有找到 則找 reference bit 為 0 且 dirty bit 為 1 的，做取代，經過但reference bit 的page為 1 則 reset為 0。

for k in range(10):

    frame\_number=input("輸入frame數量")

    frame\_number = int(frame\_number)

    index = 0

    page\_fault = 0

    interrupt = 0

    dw = 0

    pointer = 0

    # Roud 1 - fill in

    for i in range(frame\_number):

        if modify[index] == 1:

            Frame[i] = {"value":rs[index],"r":0,"m":1}

        else:

            Frame[i] = {"value":rs[index],"r":0,"m":0}

        page\_fault = page\_fault + 1

        index = index + 1

    print("fill in ",page\_fault)

    # one cycle back to position 0 : point to 0

    # pointer = 0

    # Round 2 - judge whether page fault

    while index < len(rs):

        frame\_status = [] # 紀錄 該frame存放的value

        status       = [] # reference bit: r, dirty bit: m

        for i in range(frame\_number):

            # 印出目前狀況

            #print(Frame[i]["value"]," ",Frame[i]["r"]," ",Frame[i]["m"])

            frame\_status.append(Frame[i]["value"])

            status.append([Frame[i]["r"],Frame[i]["m"]])

        # check whether in frame

        exist = 0 # default is not found

        for i in range(frame\_number):

            if rs[index] == Frame[i]["value"]:

                Frame[i]["r"] = 1

                exist = 1 # found

                index = index + 1

                break # break search

        # page fault 發生 shift to next page

        # or page survive shift to next page

        if exist == 0 :

            while True: # cycle frame list

                #print('pointer: ',pointer)

                out\_of\_Loop1 = 0

                # run Loop 1

                while True:

                    if Frame[pointer]["r"] == 0 and Frame[pointer]["m"] == 0:

                        Frame[pointer]["value"] = rs[index]

                        Frame[pointer]["m"] = modify[index]

                        Frame[pointer]["r"] = 0

                        page\_fault = page\_fault + 1

                        interrupt = interrupt + 1

                        index = index + 1

                        out\_of\_Loop1 = 1

                        pointer = (pointer%4) + 1

                        break # Loop1

                    else:

                        pointer = pointer + 1

                    if pointer >= (frame\_number-1):

                        pointer = (pointer%4) + 1

                        break # Loop1

                if out\_of\_Loop1 == 1 :

                    break    # jump out While True == 不用再找下去

                # run Loop 2

                out\_of\_Loop2 = 0

                if out\_of\_Loop1 == 0:

                    while True:

                        if Frame[pointer]["r"] == 0 and Frame[pointer]["m"] == 1:

                            Frame[pointer]["value"] = rs[index]

                            page\_fault = page\_fault + 1

                            interrupt = interrupt + 1

                            dw = dw + 1

                            Frame[pointer]["m"] = modify[index]

                            Frame[pointer]["r"] = 0

                            out\_of\_Loop2 = 1

                            index = index + 1

                            pointer = (pointer%4) + 1

                            break # Loop 2

                        else:

                            Frame[pointer]["r"] = 0

                            pointer = pointer + 1

                        if pointer >= (frame\_number-1) :

                            pointer = (pointer%4) + 1

                            break

                if out\_of\_Loop2 == 1 :

                    break   # jump out While True == 不用再找下去

    print("total page fault ",page\_fault)

    print("interrupt ",interrupt)

    print("dw ",dw)

>esc\_random.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



>esc\_Locality\_rs.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



>esc\_custom.py

一張含有 文字 的圖片

自動產生的描述

一張含有 文字, 地圖 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述



1. Custom Algorithm : LRU\_Pro

第一優先 被替換掉: 在 frame 累積出現次數最少。

當 累積次數最小是一樣的話，進入第二優先。

第二優先 被替換掉: 最近沒被使用到(很久沒被用到)。

for z in range(10):

    fn = input("輸入frame數: ")

    fn = int(fn)

    page\_fault = 0

    dw = 0

    interrupt = 0

    frame = [] # 記憶體 狀況

    frame\_record = [] # 紀錄出現次數

    index = 0 # 到哪個 reference

    for i in range(fn):

        frame.append(rs[index])

        frame\_record.append(1) # 從1開始，計算次數

        index = index + 1

        page\_fault = page\_fault + 1

    print(page\_fault) # 顯示目前page fault

    while index < len(rs):

        if rs[index] in frame:

# 次數 + 1

            frame\_record[frame.index(rs[index])] = frame\_record[frame.index(rs[index])] + 1

# 變更 累積次數

            tmp1=frame\_record[frame.index(rs[index])]

            del frame\_record[frame.index(rs[index])]

            frame\_record.append(tmp1)

# 變更 記憶體 狀況

# 新進 塞最後

            tmp2 = frame[frame.index(rs[index])]

            del frame[frame.index(rs[index])]

            frame.append(tmp2)

        else:

            min\_value = min(frame\_record) # 找出次數最少的數量

 # 找離目前reference page 最近的 reference page

# 如果 dirty bit 為 1 , 則 disk write++

            for k in range(index-1,0,-1):

                if rs[k] == frame[frame\_record.index(min\_value)]:

                    if dw\_set[k] == 1 :

                        dw = dw + 1

                    break

            # 替換，並新增

            frame.remove(frame[frame\_record.index(min\_value)])

            frame.append(rs[index])

            del frame\_record[frame\_record.index(min\_value)]

            frame\_record.append(1) # 該 frame 次數 重設為1

            interrupt = interrupt + 1

            page\_fault = page\_fault + 1

        index = index + 1

    print(page\_fault)

    print(interrupt)

    print(dw)

>LRU\_Pro\_Random.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



>LRU\_Pro\_Locality.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面 的圖片

自動產生的描述



>LRU\_pro\_custom.py

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

一張含有 螢幕擷取畫面, 文字 的圖片

自動產生的描述

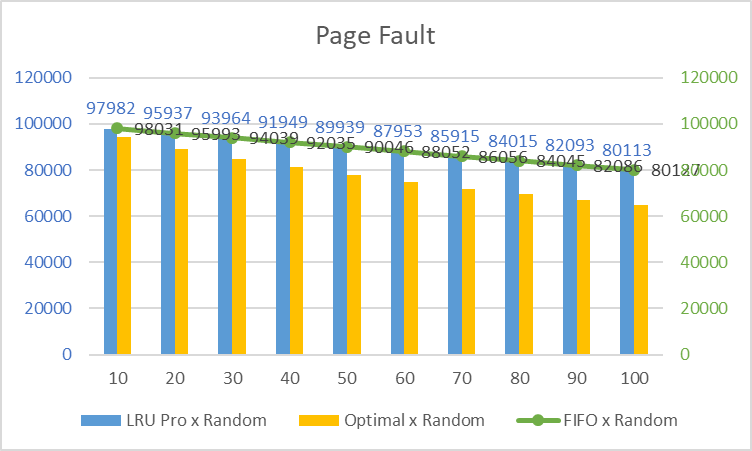
一張含有 螢幕擷取畫面 的圖片

自動產生的描述



FIFO vs LRU Pro (自訂演算法) vs Optimal : Random

[Page Fault分析] LRU Pro (自訂演算法) 略勝於 FIFO。



[Interrupt 分析] LRU Pro (自訂演算法) 險勝於 FIFO。



[disk write 分析]



FIFO vs LRU Pro (自訂演算法) vs Optimal :Locality

[分析Page Fault] LRU Pro (自訂演算法) 隨著 Frame 數 ，拉開與 FIFO 的差距。



[Interrupt 分析] LRU Pro (自訂演算法) 隨著 Frame 數 ，拉開與 FIFO 的差距。

現象和 Page Fault 差不多。



[disk write 分析]



FIFO vs LRU Pro (自訂演算法) vs Optimal :Custom

[分析Page Fault] 專為 LRU Pro 設計 ，Optimal == LRU Pro

[說明] Custom 是專為 LRU page 設計的，雖然LRU Pro 不能像 Optimal 預知未來，但以優先1的篩選方式(次數最多不換)，讓將要替換的最後一個page，次數最少，所以在執行時間上，LRU Pro 會比 Optimal 還要來的快，同時達到 Optimal 的效果，在 Custom Reference String 的Case 上。



[Interrupt 分析]



[disk write 分析]

