CH8、Virtual Memory

虛擬記憶體

目錄：

Virtual Memory

一、主要目的(優點)：允許 Process 在超過 Physical Memory 可用空間大小情況下，Process 仍能執行。是 OS 的責任(負擔)，Programmer 無啥負擔

二、附帶好處

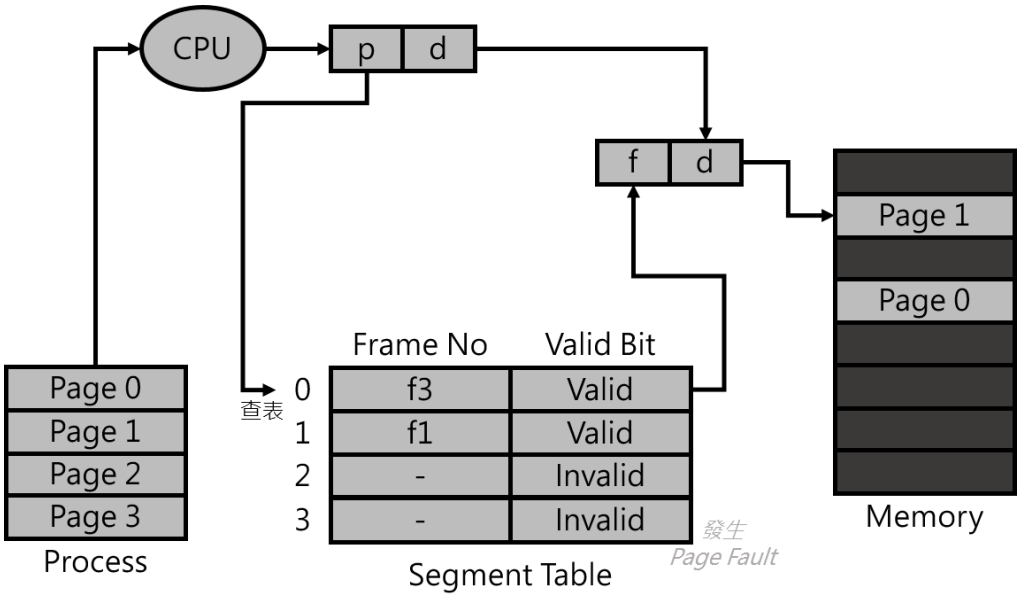
1. Memory 利用度較高
2. 盡可能地提高 Multiprogramming Degree、增加 CPU 利用度*(Note：Thrashing 除外)*
3. IO Transfer Time 較小*(Note：IO 次數 Total Time 增加)*
4. Programmer 只需專心寫好程式即可，無需煩惱程式過大、無法執行的問題，這是 OS 的責任。所以 Programmer 也不需要過時的 Overlay 技術

實現 Virtual Memory 的技術之一：Demand Paging 需求分頁

一、是架構在 Page Memory Management 基礎上

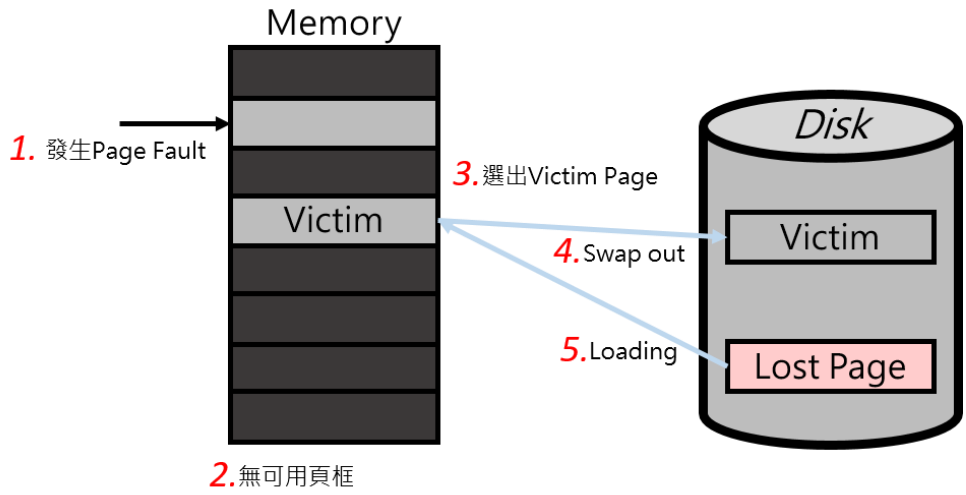
差別：在於採用"Lazy Swapper"的觀念，即 Process 執行之初，無需事先載入全部的 Pages，而是只載入部分的 Pages(甚至不載入任何 Page、Pure Demand Paging)，Process 即可執行。若 Process 執行時，它所需要的 Pages 皆在 Memory 中，則 Process 本身一切正常地執行。若 Process 執行時，企圖存取不在 Memory 中的 Pages，則稱為發生 Page Fault，OS 必須處理。將 Process 所需的"Lost" Page 從 Disk 載入到 Memory 中，Process 才可執行

二、在分頁表中，需引進一個欄位：Valid/Invalid Bit，用以區分此 Page 是否在 Memory 中。Valid 表示有在 Memory 中；Invalid 則表示沒有在 Memory 中。*(Note：此 Bit 是由 OS 設定/更改、MMU 只負責 Reference 讀取)*



Page Fault 之處理步驟

1. MMU 會發出一個"Address Error" Interrupt 通知 OS
2. OS 收到中斷後，必須暫停，目前 Process 之執行且保存其 Status Info
3. OS 檢查 Process 之存取位址是否合法。若非法，則終止 Process；若合法，則 OS 判定是由 Page Fault 所引起
4. OS 先到 Memory 中檢查有無 Free Frame，若沒有，則必須執行"Page Replacement"工作以空出一個 Free Frame
5. OS 再到 Disk 中找出 Lost Page 所在位置，啟動 IO 運作，將 Lost Page 載入到 Free Frame 中
6. OS 修改 Page Table 記錄此 Page 的 Frame Number、以及將 Invalid 值改為 Valid 值
7. OS 恢復中斷之前 Process 的執行



Note：
(1)p8-5 簡單一些
(2)p8-7 更多一些步驟：6.…OS 可能將 CPU 分給(也可以不分，由 OS 佔用)…

Effective Memory Access Time 計算 in Virtual Memory

一、公式：

(1-p)*Memory Access Time + p*(Page Fault Process Time + *MAT(太小可略)*)
Where p is Page Fault Ratio

二、例題

例 1-1：

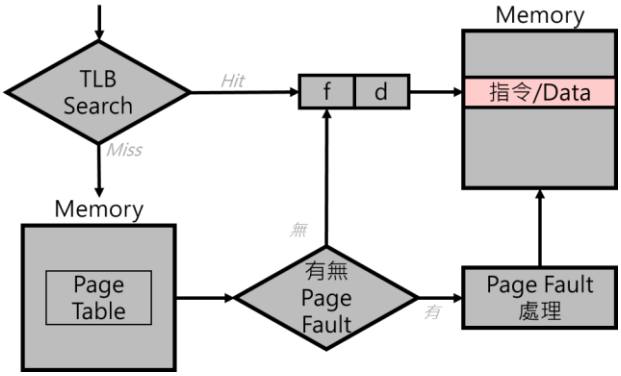
Memory Access Time = 200ns
Page Fault Process Time = 5ms
若 Page Fault Ratio = 10%
求 Effective Memory Access Time？

*(1-0.1)*200ns + 0.1*5ms = 180ns + 0.1*5*10⁶ns = 500180ns*

例 1-2：
若希望 Effective Memory Access Time 不超過 2ms，則 Page Fault Ratio 應<= ？

*(1-p)*200ns + p*5ms <= 2ms*
200-200p + 5000000p <= 2000000
4999800p <= 1999800
⇨ *p <= 19998/49998 = 2/5*

例 2：[補](Stalling)



令 p 代表 TLB Hit Ratio、q 代表 Page Fault Ratio
Effective Memory Access Time
= p*(TLB Time + Memory Access Time) + (1-p)*(TLB Time + Memory Access Time (查 Page Table) + Memory Access Time(存取指令) + q*Page Fault Process Time)

小結：
欲降低 Effective Memory Access Time，提升 Virtual Memory 效益，關鍵作法：降低 Page Fault Ratio

影響 Page Fault 的因素
1. Page Replacement 法則之選擇
2. Frame 數分配多寡之影響
3. Page size 之影響
4. Program Structure 之影響

Page Replacement(頁面替換)
一、Def：當 Page Fault 發生，且 Memory 中無 Free Frame 時，OS 必須執行此工作，即選擇一個 Victim Page (or the Replaced Page)，將它 Swap out 到 Disk 保存，以空出一個
會額外多出一個 Disk IO 運作，因此 Page Fault Process Time 更久

二、如何降低 Swap out 此一額外 IO 次數

作法：在 Page Table 再多引進一欄位：Modification Bit(或 Dirty Bit)，用以表示此 Page 上次載入後到現在，內容是否被修改過：0 表無修改、1 表有修改

OS 可檢查 Victim Page 的 Modification Bit 值，若為 0，則無需 Swap out，因此可降低 IO 次數；反之，則需要 Swap out

Note：此 Bit 由 MMU 設定/修改，OS 只作 Reference(反之是 Valid Bit 由 OS 設定，MMU Reference)

三、計算題

例 1：p8-5.12

Page Fault Process Time, if 有可用頁框或 the replaced page isn't modified：8ms
Page Fault Process Time if victim page is modified：20ms
Memeory Access Time：100ns
Victim Page is modified 機率：70%
求 Page Fault Ratio<= ?, if Effective Memory Access Time ?

*(1-p)*100ns + p * Page Fault Page Tabe(0.3*8 + 0.7*20) =*
*(1-p)*100ns + p*16.4ms <= 200ns*
*(1-p)*100 + p**16400000 <= 200*
*P*16399900 <= 100 => p<= 1/163999*

例 2：

1 次 IO Time：10ms
Page Fault Ratio：10%
Victim Page is modified Ratio：60%
Memory Access Time：200ns
求 Effctive Memory Access Time ?

*0.9*200ns + 0.1*(0.4*10ms + 0.6*20ms) = 1.60018ms*

四、Replacement Policy

1. Local Replacement Policy：OS 只能從發生 Page Fault 的 Process 之 Pages 中，去挑選 Victim Page，不可以從其他 Process 之 Pages 挑 Victim Page
   缺點：Memory 利用度較差
   優點：限縮 Thrashing 的範圍
2. Globlal Replacement Policy：OS 可從其他 Process 挑 Victim Page
   缺點：Thrashing 的範圍可能擴大到其他 Process
   優點：Memory 利用度較好

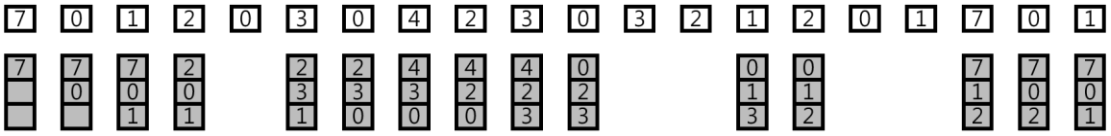
Page Replacment 法則

1. FIFO
2. OPT
3. LRU
4. LRU 近似
   (1) Additional Reference Bits Usage
   (2) Second Chance
   (3) Enhanced Second Chance
5. LFU 與 MFU
6. Page Buffering Algorithm(機制)

FIFO 法則

一、Def：最早載入(Loading Time 最小)的 Page，選為 Victim Page

二、例：給予 3 個 Frames. Initially, they are all empty (或 Pure Demand Paging)，有下列的 Page Reference String，求 Page Fault 次數：
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



共 15 次，15/20 = 75%

或者另類考法：Page size = 100，下列存取位址：731, 008, 117, 258, 039, 331, 047…，要先去除 100(退兩位)；或者：Logical address = 12 bits, Page Number 佔 3 bits 存取位址如下：7AF, 8BD, 259, 047, EAF, D72…，要先轉成二進位碼後取最左邊 3 bits

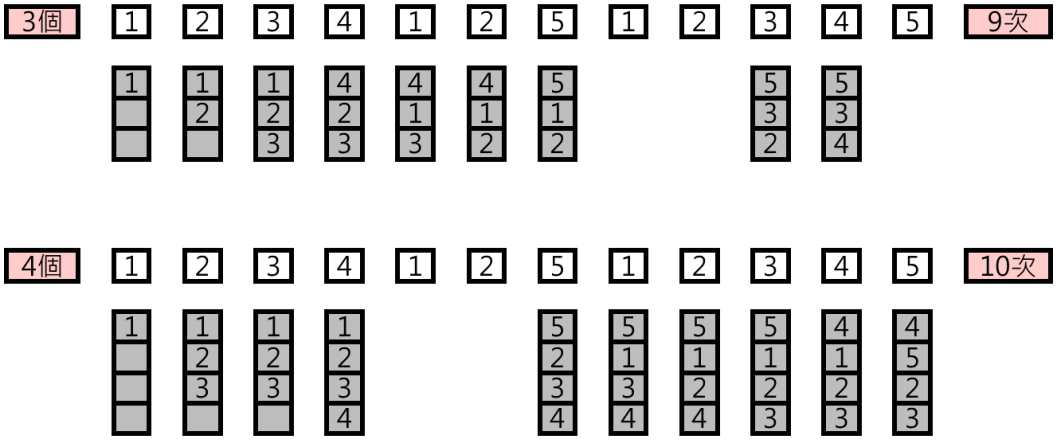
三、分析

1. Simple, easy implementation
2. 效能不是很好，Page Fault Ratio 相當高
   (Note：Page Replacement 法則中，只有最佳，沒有最差)
3. 可能有 Belady Anomaly(異常現象)

Belady Anomaly

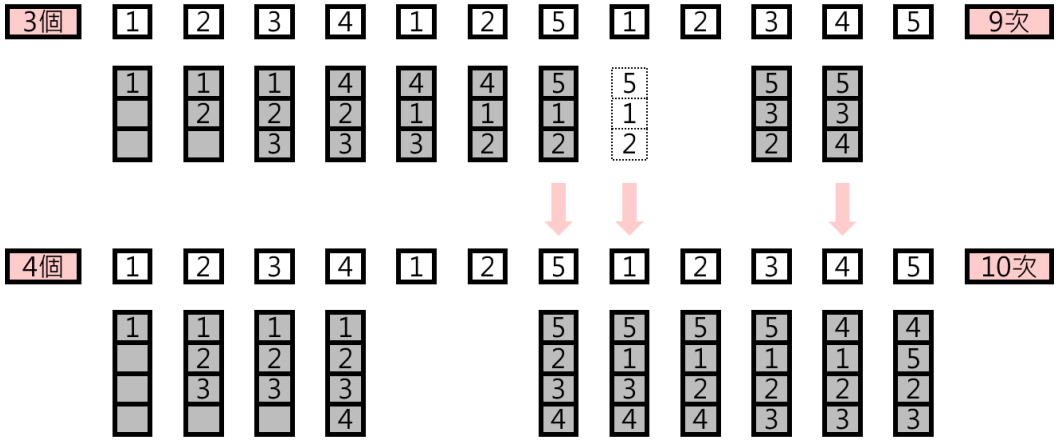
一、Def：Process 分配到的頁框數增加，其 Page Fault Ratio 卻不降反升之異常現象

二、例：1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
三個頁框時，只會發生 9 次 Page Fault
四個頁框時，會發生 10 次 Page Fault
故有 Belady Anomaly



三、Stack Property

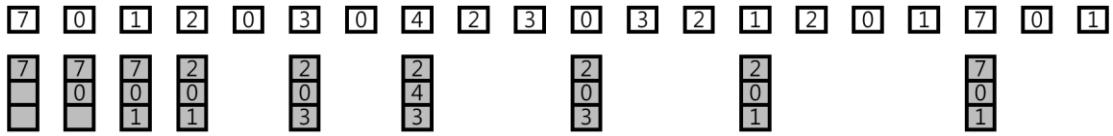
1. Def：n 個 Frames 所包含的 Pages Set，是(n+1)個 Frames 所包含的 Page set 之子集合(subset)，此性質稱之
2. 若具有 Stack Property，保證不會有 Belady Anomaly
3. 只有 OPT 與 LRU 法則，具有 Stack Property，因此它們不會發生 Belady Anomaly



OPT(Optional，最佳)法則

一、Def：選擇將來(未來)長期會使用到的 Page 為 Victim Page

二、例：9 次 Page Fault



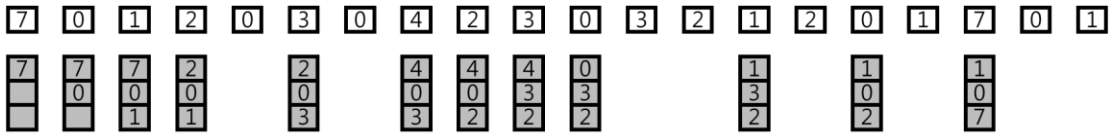
三、分析：

1. Page Fault Ratio 最低，因此叫最佳
2. 不會有 Belady Anomaly
3. 無法被實作(因為 needs the future knowloedge)，通常作為理論上研究比較對象之用

LRU(Least Recently Used)法則

一、Def：選擇過去不常使用的 Page 作為 Victim Page；即相當於是 OPT reverse(依歷史資訊作決定的 OPT)，即挑 Least Reference Time 最小的 Page

二、例：9 次 Page Fault



三、分析

1. Page Fault Ratio 可以接受
2. 不會發生 Belady Anomaly
3. LRU 的製作需要大量硬體支援，因此成本很高，所以才有 LRU 近似法則

四、製作

[法一]：Counter 法

Def：每發生一次 Page Reference，Counter 值會加 1，且此值會 Copy 到該存取 Page 的 the Last Refernce Time 欄位/Register，將來 OS 要選 LRU Page 時，就挑 the Last Refernce Time 最小的 Page

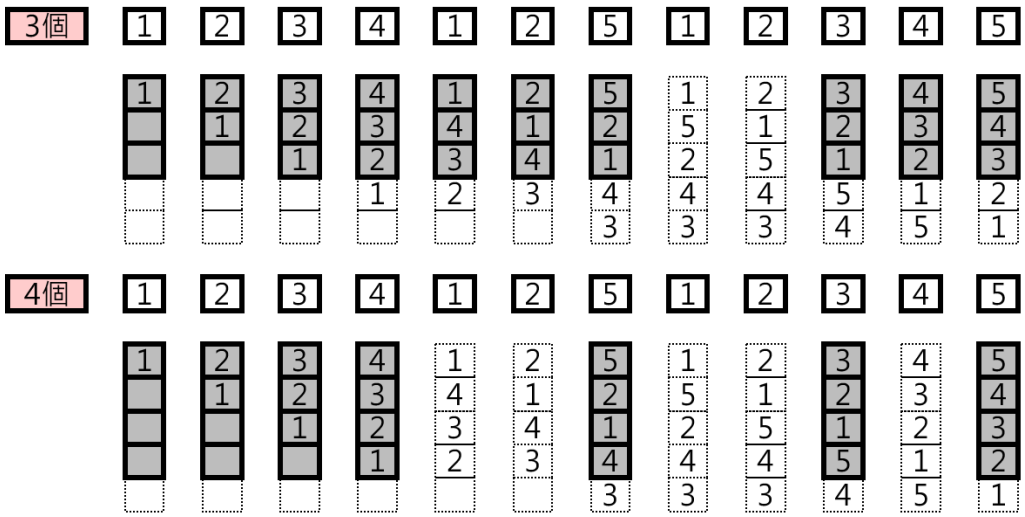
[法二]：Stack 法

Def：最後一次存取之 PAGE 必置於 Stack Top 端

Stack 之 Bottom(底端)，即 LRU Page

Stack 大小 = Frame 數目

例：1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



3 個頁框 = 10 次；*Note：FIFO 中，3 個頁框只要 9 次(故只有最佳、沒有最差)*

LRU 近似法則

一、主要是以”Reference Bit”(參考位元)為基礎：

0 表此 Page 未被參考過、1 表此 Page 曾被參考過

二、方式

[法一]：Additional Reference Bits Usage

作法：每個 Page 有一個欄位(or Register)，例如：8 bits，當發生 Memory Access，該被 Access Page 之 Reference Bit 會設為”1”；系統每隔一段時間會將各 Page 的 Register 值右移一位(空出最高位元；最右位元捨去)，並將各 Page 之 Reference Bit 值 Copy 到 Register 之最高位元，且 reset 各 Page 之 Reference Bit 為 0。將來要挑 Victim Page 時，就挑 Register 值最小之 Page。若多個 Pages 具相同值，則以 FIFO 為準

例：1, 2, 1, 1, 1, 1, (Reset) 3, 1, 3, 3, 3, 3, (Reset) 5

第一次區間的 Reset：

第二次區間的 Reset：

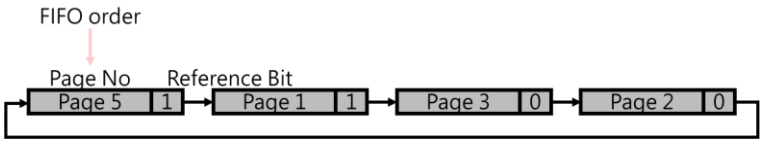
比較 Regiser 大小，選擇最小者為 Victim Page，故選擇 Page 2 的 2，來置換成 5

Second Chance(二次機會)法則

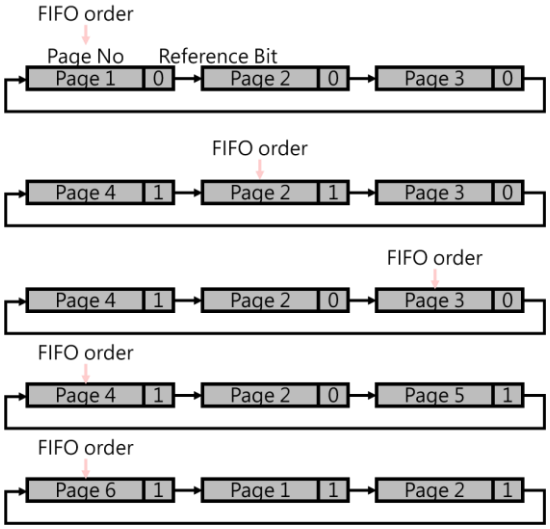
一、Def：以 FIFO 為基礎，搭配 Reference Bit 使用，挑 Victim Page 之步驟：

1. 先以 FIFO 挑出一個 Page
2. 檢查此 Page 的 Reference Bit 值
   (1) 若為 0，則它即是 Victim Page
   (2) 若為 1，則跳至 3.
3. 給它機會(不挑它為 Victim Page)
4. Reset 它的 Reference Bit 值為 0
5. 它的載入時間改為現在的時間
6. Goto 1.

二、例：四個頁框



三個頁框：1, 2, 3, 4, 2, 5, 2, 6, 1, 2



Note：當所有 Page 之 Reference Bit 皆相同，則退化成 FIFO，也叫作 Clock Alogrithm

Enhanced Second Chance(加強型二次機會)法則

Def：以<Reference Bit, Modification Bit>配對值，作為挑選 Victim Page 之依據，值最小之 Page 為 Victim Page，若有多個 Pag 具相同值，則以 FIFO 為準

	<Reference Bit, Modification Bit>
由小	<0, 0>
	<0, 1>
	<1, 0>
到大	<1, 1>

LFU 與 MFU 法則

一、Def：以 Page 的累計參考總次數作為挑選 Victim Page 之依據

1. LFU(Least Frequenctly Used)：次數最小的 Page 為 Victim Page
2. MFU(Most Frequenctly Used) ：次數最大的 Page 為 Victim Page

若有多個 Page 具相同值，也是 FIFO 為準

(略過計算)

二、分析：

1. Page Fault Ratio 相當高
2. 有 Belady Anomaly
3. 製作需相當大量的硬體支援，成本很高

例：如下表，則不同法則所選出的下一個 Victim Page 為何？

Page	Loading Time	The Last Reference Time	Reference Bit	Modification Bit	參考次數
P1	493	800	0	0	410
P2	172	700	1	1	235
P3	333	430	0	1	147
P4	584	621	1	0	875
P5	256	564	0	1	432

1. FIFO
2. LRU
3. Second Chance
4. Enhanced Second Chance
5. Least Frequently Used
6. Most Frequently Used

1. *P2*
2. *P3*
3. *P5*
4. *P1*
5. *P3*
6. *P4*

Page Buffering Alogrithm 機制

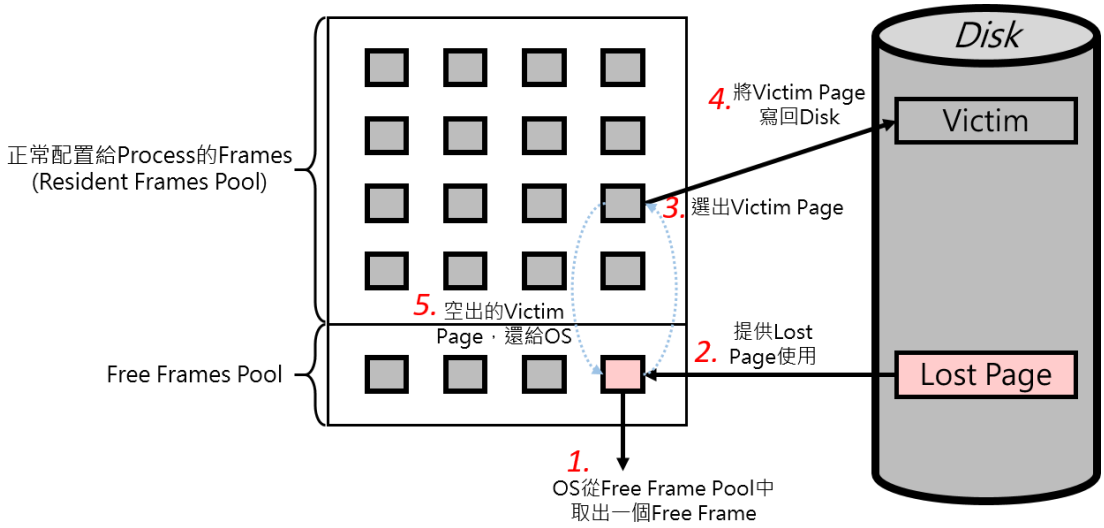
一、緣由：當挑出 Victim 後，且它被 Modified 過，則：

1. 將 Victim Page Swap out 到 Disk
2. 載入 Lost Page
3. Process resume execution

   Process 恢復執行之時間點拖太久，需要改善

二、方法：

[法一]：OS 會 keeps 一個 Free Frames Pool*(OS 的私房錢，不是配置給 Process 用途)*



OS 挑完 Victim Page(modified)：

1. OS 從 Free Frame Pool 中取出一個 Free Frame，供 Lost Page 載入
2. 載入完後，Process 即可恢復執行
   OS 可稍後將 Victim Page 寫回 Disk，空出之 Frame，再還給 OS，加入 Free Frame Pool 中

[法二]：OS 會 keep 一條"Modification List"，記錄所有被 Modified 過的 Page Information(即 Modification Bit = 1 之 Page)

OS 會等 Paging IO Device 有空時，將此 List 中，某個 Page 寫回 Disk，同時自此 List 中移除這些 Pages，然後 reset their Modification Bit 值為 0

如此可增加 Victim Page 是 unmodicfied 之機率，這樣 Process 有較高機會快速恢復執行

[法三]：是架構在[法一]基礎上，差別在於多作"針對 Free Frame Pool 中的每個 Frame，記錄放的是哪個 Process 的哪個 Page(<Process ID, Process Number>)"。這些 Page 內容必定是最新的

流程：

1. OS 選完 Victim Page(Modified)
2. OS 去 Free Frame Pool 中尋找有無 Lost Page 存在；若存在，則將此 Free Frame 加入 Resident Free Page 中，Process 即可恢復執行，連一次 IO 皆不用，OS 寫回 Victim Page 後，再加入/還給 Free Frame Page；若不存在，則依[法一]步驟處理

例 p8-90.89：An operating system uses a FIFO replacement algorithm for resident pages and a free-frame pool of recently used pages. Assume that the free-frame pool is managed using the least recently used replacement policy. Answer the following questions:

1. If a page fault occurs and if the page does not exist in the free-frame pool, how is free space generated for the newly requested page?
2. If a page fault occurs and if the page exists in the free-frame pool, how is the resident page set and the free-frame pool managed to make space for the requested page?
3. What does the system degenerate to if the number of resident pages is set to one?
4. What does the system degenerate to if the number of pages in the free-frame pool is zero?

1. *[法一]步驟*
2. *[法三]步驟*
3. *LRU*
4. *FIFO*

例：舉例說明：

1. LFU 之 Page Fault 次數少於 LRU：
2. LFU 之 Page Fault 次數多於 LRU：

1. *1, 1, 2, 3, 3, 4, 1*
   LFU：4 次；LRU：5 次

2. *1, 2, 2, 2, 3, 3, 3, 1, 4, 1*
   LFU：5 次；LRU：4 次

Frame 數目分配多寡之影響

一、一般而言(in general)：Process 分配到的頁框數增加，其 Page Frame Ratio 理應下降

二、OS 在分配 Process 頁框數時，必須滿足最少及最多之限制，這 2 個限制均由硬體決定、非 OS 決定

1. 最多數目限制：即是 Physical Memory size(頁框總數)
2. 最少數目限制：CPU 完成機器指令執行過程中，最多的可能 Memory Access 次數

   否則機器指令可能永遠無法完成

例：IF、 ID、 EXE、 MEM、 WB
1 次 1 次 1 次

1. 假設指令不跨頁面：
   運算元(Memory 變數)是採 Direct addressing mode，則最多需要__次 Memory Access？
   OS 至少要分給 Process__Frames？
   3 次、大於 3 個

2. 假設指令跨頁面：
   運算元(Memory 變數)是採 Direct addressing mode，則最多需要__次 Memory Access？
   OS 至少要分給 Process__Frames？
   6 次、大於 6 個

Thrashing 現象

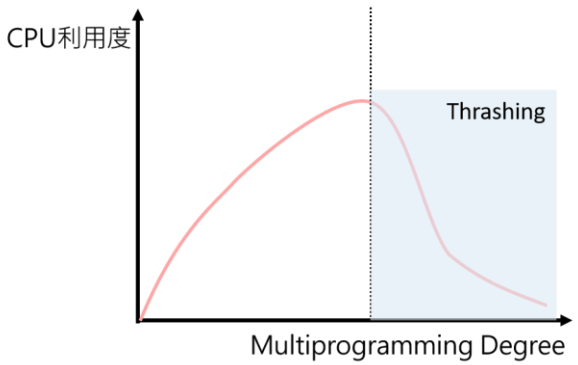
Def：若 Process 分配到的頁框數不足時，則此 Process 會經常 Page Fault，且 OS 要 Page Replacement。

若 OS 採用 Global Replacement Policy，所以 OS 可能挑其他 Process 的頁面，作為 Victim Page，而這也可能造成其他 Process 的 Page Fault，且也會去搶別的 Process 的 Page 使用

如此一來，大家都在 Page Fault，大家皆等待 Paging IO Device 之 IO 動作(swap out/in)完成，此時 CPU 利用度下降，系統會企圖調高 Multiprogramming Degree，引入更多 Pages 進入執行，但是 Memory 本來就不夠了，這些 Process 也立刻就發生 Page Fault，系統又再度調高 Multiprogramming Degree，如此循環下去，此時系統呈現：

1. CPU 利用度急速下降
2. Paging IO Device 異常忙碌
3. Process 花在 Page Fault Process Time 遠大於正常執行時間

此一現象稱為 Thrashing



解決/預防 Thrashing 之方法

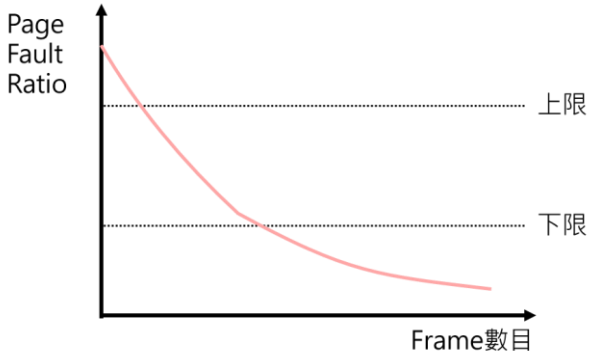
[法一]Thrashing 發生

Decrease Multiprogramming Degree

Ex：挑一些 Lower-Priority 或完成度低的 Process Swap out

[法二]：利用 Page Fault Frequency Control 機制，來防止/預防 Thrashing 發生

作法：OS 制定 Process Page Fault Ratio 之合理上限與下限值



OS 發現 Process 的 Page Fault Ratio：

1. 高於上限值：OS 應多分配額外的 Frame 給此 Process，以降低其 Ratio，以回到合理區間
2. 低於下限值：OS 應該自此 Process 取走多餘的 Frame，分給其他有需要的 Process

因此 OS 能夠控制所有 Process 之 Page Fault Ratio 在合理區間，則理當不會發生 Thrashing

[法三]：運用 Working Set Model 技術

預估 Process 在不同執行時期所需之頁框數目，並依此分配足夠的 Frame 數目，以防止 Thrashing

Working Set Model 技術

一、是架構在 Locality Model 之理論基楚上

Def：Process 執行時，對於所存取的 Memory Area，並非是均勻的，而是有某種局部/集中區域存在之特性，一般分為 2 種 Locality

1. Temparal Locality (Time Locality)時間區域性
2. Spacial Locality 空間區域性

1. 時間區域性：
   Def：目前所存取的區域，過不久又會再度被存取(或此區域經常被存取)
   例： Loop 敘述：while, for, repeat…until
   Subroutine(副程式)：function, pure code
   Counter：i++, i--
   Stack：Top 元素
2. 空間區域性：
   Def：目前所存取之區域，其鄰近的區域也極有可能被存取
   例：Array, Sequential Code Execution, Common Data Area, Linear Search, Vector operater

二、只要 Program 中用到的指令，Data Structure Algorithm 符合 Locality Model，則 Good(因為 Page Fault Ratio 應會下降)；若違反，則 Bad

例(Bad Semple)：Hashing, Binary Search, Linked-List, goto/jump 指令, Indirect addressing mode，皆違反空間區域性

三、Working Set Model 相關術語

1. Working Set Model：記為 Δ，表示以 Δ 次 Page Reference 作為統計 Working Set 之依據
2. Working Set：在 Δ 次 Page 參考中，所參考到的不同 Pages 之集合
3. Working Set size(wss)：Working Set 之元素(Page)個數，代表 Process 此時所需之頁框數

例：Δ=10 次

t1 t2

2, 1, 5, 1, 6, 6, 7, 1, 1, 5, 4, …, 2, 3, 3, 3, 4, 4, 3, 3, 4, 3, 3, 1

則：

1. t1 時間點，Working Set={____}，wss=____
2. t2 時間點，Working Set={____}，wss=____

1. {1, 2, 5, 6, 7}, 5
2. {3, 4}, 2

四、OS 如何運用？

假設 n 為 Process 個數

wss_i = $Process_i$ 在此時期的 Working Set size

設 D= Sum wssi = 頁框的總需求量(Demand)

令 M=Physical Memory 頁框總數(size)

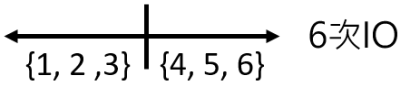
1. D <= M
   則 OS 會依 wssi 值分配給 Pi 足夠的 Frame 數，如此可防止 Thrashing
2. D > M
   則 OS 選擇一些 Process，將它們 Swap out，以降低 D 值到 D <= M 為止，OS 再分配

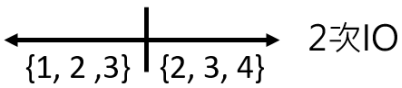
五、優點：

1. 可防止 Thrashing
2. 對 Prepaging 也有助益(事先猜測 Process 所需要哪些 Page，並事先載入，以避免初期之大量 Page Fault(if 猜錯))

缺點：

1. 不易制定精確的 Working Set
2. 若前後期差異很大，則 IO 次數上升





例 1(p8-58.39)：下列狀況 Increase Multiprogramming Degree 是否有助於提高 CPU 利用度？

1. CPU 利用度：3%, Disk 利用度：97%
2. CPU 利用度：87%, Disk 利用度：13%
3. CPU 利用度：13%, Disk 利用度：3%

只有 3

例 2(p8-63.49)：

CPU 利用度：20%

Paging Disk：97%　　=> 代表 Thrashing

下列哪些措施 will, is likely to, never improve CPU 利用度？

1. Increase Multiprogramming Degree
2. Decrease Multiprogramming Degree
3. Install Faster CPU
4. Install More Main Memory
5. Install Bigger Memory
6. Install Faster Disk
7. Local Replacement Policy Used
8. Prepaging Used
9. Page size 變大
10. Page size 變小

Will：2, 4；Is likely to：6, 7, 8, 9；Never：1, 3, 5, 10

Page size 之影響

若 Page size 愈小，則：

1. Page Fault Ratio 上升
2. Page Tabe size 變大
3. IO 次數(total time)變大
4. 內部碎裂：輕微
5. IO Transfer Time：變小
6. Locality：愈佳

趨勢：朝向『大的』Page size 在設計

Programming Struture 之影響

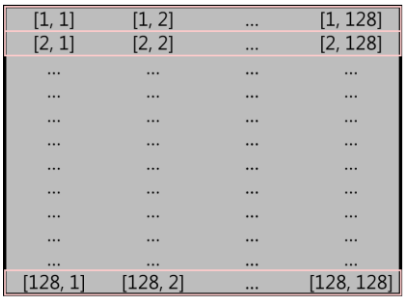
一、若 Programming 中使用的指令，Data Structure Algorithm 符合"Locality Model"，則為 Good(因為有助於降低 Page Fault Ratio)；反之若違反，則 Bad

二、程式中對於 Array 元素之處理順序，最好與 Array 元素在 Memory 中的儲存方式(Raw-Major 或 Column-Major)對應，有助於降低 Page Fault Ratio

例 1：A array[1: 128, 1:128] of int，每個 int 佔 1 Byte。A 以 Raw-Major 方式存於 Memory，Page size = 128 Byte，給 3 個 Frames 且程式已在 Memory 中，採 FIFO Replacement Policy，求下列 Code 之 Page Fault 次數

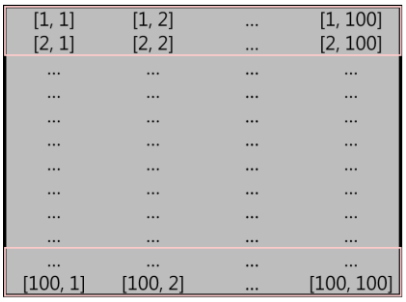
1.	for i=1 to 128 do
	　　for j=1 to 128 do
	　　　　A[i, j]=0;

2.	for j=1 to 128 do
	　　for i=1 to 128 do
	　　　　A[i, j]=0;



1. *每一列發生 1 次 Page Fault，因此共 128 列*1 次 = 128 次 Page Fault*
2. *每一行發生 128 次 Page Fault，因此共 128 行*128 次 = 128*128 次 Page Fault*

例 2：A array[1: 100, 1:100] of int，每個 int 佔 1 Byte。A 以 Raw-Major 方式存於 Memory，Page size = 200 Byte，給 3 個 Frames 且程式已在 Memory 中，採 FIFO Replacement Policy，求下列 Code 之 Page Fault 次數



1. *每 2 列發生 1 次 Page Fault，因此共 50 列*1 次 = 50 次 Page Fault*
2. *每 2 行發生 100 次 Page Fault，因此共 50 行*100 次 = 5000 次 Page Fault*

Copy-on-Write 技術

(此處主要是討論 3 種 fork())

一、傳統的 fork()*(CH4)*

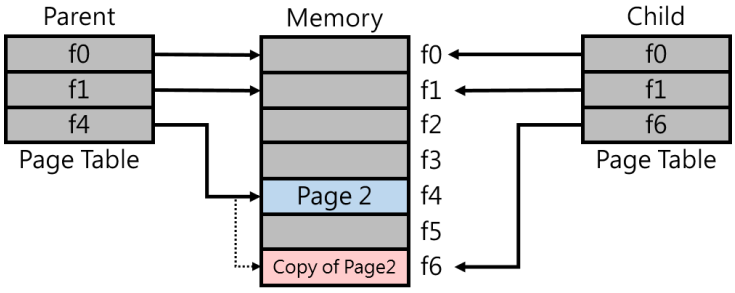
Def：Parent Process fork()建立出 Child Process，OS 會配置 New Frames 給 Child Process(即 Child 與 Parent 佔用不同的 Memory Space)，同時 OS 也要複製(copy) Parent Process 內容(Code Section 及 Data Section)給 Child Process initially

此舉會導致 2 個缺失

1. Memory(Frames)需求量大增
2. Slower Child Process Creation，且上述作法在"Child 生出後，立刻 execlp()，作其他工作"時，更加顯得無用(不必要)

二、fork() with "Copy-on-Write"技術

Def：當 Parent Process 生出 Child Process 之初，OS 會讓 Child 共享 Parent Process 之 Memory(Frames)空間，如此一來，無需配置給 Child Process new Frames，也不用 copy Parent 內容給 Child。因此可降低頁框需求量、及 Speed up Process Creation



但是任何一方改變了某 Page 內容，則另一方會受到影響(此為 risk)。因此引入"Copy-on-Write"技術

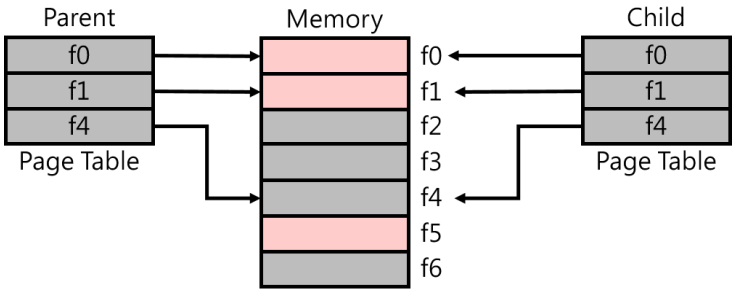
即：若 Child Process 想要更改某 Page 之內容(ex：Stack 內容)，則 OS 會配置一個 Memory Frame 給 Child，且 copy Page 內容到 Memory Frame 中，供 Child 使用/修改(且修改 Child 的 Page Table 指向 New Frame)，如此一來，則不會影響 Parent 那些有 Modified 可能的 Page，需標示 Copy-on-Write，而有些不會 Modified Page(ex：Read-only Code/Data)，即可共享

三、vfork()(Virtual Memory fork())

Def：Parent 生出 Child 之初，也是讓 Child 共享 Parent 相同的 Frame，但是它並未提供"Copy-on-Write"技術

所以任何一方改變了某 Page 內容，則另一方會受到影響，故務必小心使用

因此特別適用在當生出 Child 後，Child 立刻執行 execlp()、去作其他工作時，vfork()非常有效率



例：Command Interpreter(命令解譯器)製作(ex：UNIX shell)

TLB Reach

一、Def：經由 TLB Mapping 所能存取到的 Memory Area 大小，即 TLB entry 數目*Page size

例：TLB 有 8 個 entry，且 Page size=16KB

*TLB Reach = 8*16KB = 128KB*

Note：希望 TLB Reach 愈大愈好

二、如何加大 TLB Reach？

[法一]：提高 TLB entry 數目

優點： 1.TLB Reach 變大

2.連帶 TLB Hit Ratio 也較高

缺點： 1.成本高(貴)

2.有時 TLB entry 數仍不足以涵蓋 Process 之 Working Set

[法二]：加大 Page size

優點：加大 TLB Reach，成本可接受

缺點：內部碎裂會很嚴重

=>解決辦法：現代的硬體均提供一些大小不同的 Page(Multiple Page size)

例：提供 2 組 Page size(4KB、2MB)

TLB 記錄項目：

Page Number	Frame Number	Page(Frame) size

此外，管理以前是硬體(MMU)在管理，現在變由 OS 來管理