作業說明

• 請在下列的40個問題中，挑選並完成其中20個題目。

• 其中每一章節需要完成至少2/5的題目(4, 2, 4, 3題)。

• 其餘七題可以自行挑選要完成哪個章節的那些題目。

• 每一題請標明章節-題號(例如6-3, 8-4)並按照章節-題號順序作答。

• 請上傳完成的作業pdf檔案，可以使用手寫或打字。

• 可以參考投影片內容回答，但請不要直接複製投影片內容。

• 繳交截止時間2023/5/18 09:10。

Chapter 6

1. ■請舉例說明什麼是競爭情況(Race condition)。

A: 在shared memory communication中，若未對共享變數的存取提供任和互斥存取控制之Synchronization機制，則會造成共享變數的最終結果值，會因為Process間的交錯執行，有不同的結果值，這種data inconsistency的情況稱為Race condition。

1. ■請解釋何為臨界區間(Critical Section)。

A: Critical Section(CS)是解決Race condition problem的方法之一。

Critical Section是對”共享變數”之存取進行管制。當Process取得共享變數存取權利時，在尚未完成的期間內，仍和其他Process都無法存取共享變數，即使取得CPU也一樣。

Process可以分成2個部分，Critical Section與Remainder Section。CS的部分是Process中對共享變數進行存取敘述之集合，而RS則是除了CS之外的部分，在CS的前後，都需要由Programmer設計(加入)額外的控制程式碼，稱作Entry Section與Exit Section。而CS Design其實是在設計Entry\Exit Section Code。

優點:適用Multiprocessor System

缺點:設計較複雜、較不適用Uniprocessor System

Critical Section Design需要滿足的3個性質:

1. Mutual Exclusion:在任何時間點，最多只允許一個Process進入他自己的CS，不可有多個Process分別進入各自的CS。
2. Progress:
3. 不想進入CS的Process (或在RS內活動的Process)不能阻礙其他Process進入CS(或不參與進入CS之決策)
4. 從那些想進入CS的Process中，決定誰可以進入CS的決策時間是有限的(不可無窮)，即No Deadlock(不可Waiting forever，不能大家皆無法進入CS)
5. Bounded Waiting:從某Process提出申請到核准進入CS的等待時間(次數)是有限的。即若有n個Process想進入CS，則這一個Process至多等待(n-1)次就可進入CS(No Starvation，需公平)。
6. ■在某些情況下，可以將關閉系統的中斷(interrupt)以解決臨界區間問題， 請問這個解法可能有什麼問題？

A: 此方法即為Disable Interrupt，做法是Process在對共享變數存取之前，先Disable Interrupt(關閉中斷)，等到完成共享變數存取後，才Enable Interrupt(開啟中斷)，可保護Process存取共享變數期間，CPU不會被Preemptive。

而此做法的優點有:

1. Simple ,Easy to implement
2. 適用於Uniprocessor System(單一CPU)

缺點:

1. 不適用於Multiprocessors System，因為Disable 一顆CPU的 Interrupt，是無法防止Race condition(其他CPU上執行的Process，仍有存取共享變數的可能)，必須Disable 「全部」CPUs的 Interrupt，才可以防止Race condition，但此舉會大幅降低performance(因為無法平行處理)
2. 風險高，必須信任User process在Disable Interrupt後，短時間內會Enable Interrupt，否則kernel可能會拿不到CPU，產生極大風險。
3. ■請描述兩個程序的Peterson’s Solution解法。

A:

1. 共享變數宣告:

1.int turn; //= i or j

2.Boolean flag[i…j]; // 初值皆為false

1. Code:

[部分]

while(1){

flag[i]=true; //表有意願

turn=j; //權仗給對方

while(flag[j] && turn==j);

//對方有意願且權仗在對方，則卡住

/\*C.S.\*/

flag[i]= false; //表無意願

/\*R.S\*/.

}

[部分]

while(1){

flag[j]=true;

turn=i;

while(flag[i] && turn==i);

/\*C.S.\*/

flag[j]= false;

/\*R.S.\*/

}

1. ■請解釋何為記憶體屏障(Memory barrier)

A: 在Multiprocessors System中，Memory model可分為兩種:

1. Strongly ordered:當其中一個Processor對Memory中的資料作修改時，其他Processors可以立即看見此修改。
2. Weakly ordered: 當其中一個Processor對Memory中的資料作修改時，其他Processors可能不會立即看見此修改。

而記憶體屏障(Memory barrier)是一個指令，用於確保對記憶體中的任何修改都能傳播（可見）給所有其他Processors。Memory barrier可以處理在Weakly ordered可能出現的資料不一致狀況。當執行Memory barrier指令時，系統會確保在執行後續的read/write操作之前，所有對記憶體的read/write操作都已完成。即使指令因系統最佳化等原因而導致亂序，Memory barrier仍然能夠確保操作完成並傳播給所有其他Processors。

1. ■請解釋test and set()指令為何可以作為entry section使用。

A: Boolean Test\_and\_Set(Boolean \*target) 是一種CPU指令(硬體指令)，其指令功能為return target舊值，並將target設為true，且保證此指令是atomically execution(不可分割的執行)。

而這種特性很適合作為entry section的使用:在共享變數宣告時將target初始化為False，當一個Process想進入CS時，先執行Test-and-Set指令，檢查target變數的值並將其設置為true，如果target舊值為true表CS內有Process在活動(擋住目前Process)，False則無(可順利進入CS)，而當Process要離開CS時，需要執行target=False(表示CS內無Process，放行擋住的Process)。

1. ■在Bounded-waiting with compare-and-swap的解法中，請解釋為何可以滿足Bounded-waiting的條件。

A: compare\_and\_swap(CAS)的定義如下:

int compare\_and\_swap(int \*value, int expected, int new\_value){

int temp =\* value;

if (\*value == expected)

\*value = new\_value;

return temp

}

CAS指令功能如下:

一律回傳舊value值，且如果value == expected成立，則value設成new\_value。並保證此指令是atomically execution(不可分割的執行)。

Bounded-waiting with compare-and-swap解法如下:

1. 共享變數宣告:int lock=0;Bloon waiting[0…n-1];初值皆為false
2. Code:

while(true){

waiting[i] = true;

key=1;

while(waiting[i] && key==1)

key= compare\_and\_swap(&lock,0,1);

waiting[i] = false;

/\* critical section\*/

j=(i+1)%n;

while((j!=i) && ! waiting[j])

j=(i+1)%n;

if(j==1)

lock=0;

else

waiting[j]=false;

/\* remainder section\*/

}

此Code有滿足CS Design 中的Bounded-waiting。假設到這n個Process皆想進入CS，即waiting[0…(n-1)]皆True令是第一個執行CAS而率先進入CS者，其他都在等待中當離開CS後，必會將之waiting[1]設false，讓進入(lock仍為1)，依此類推，會依序(FCFS)進入CS，所以至多等n-1次後即可進入CS (No starvation)。

1. ■請解釋何為“busy waiting”。

A: “busy waiting”的定義如下:當有Process在CS時，其他的Process嘗試進入CS會被持續的卡在迴圈(loop)內，也稱Spin Lock。

1. ■請解釋使用“busy waiting”的好處與壞處。

A: “busy waiting”的好處(優點):若Process卡在loop的時間很短(i.e.小於2個Context Switch Time)則busy waiting有利。

“busy waiting”的壞處(缺點):等待中的Process會與其他Processes競爭CPU，將搶到的CPU Time用在毫無實際進展的迴圈測試上，因此若Process要等待長時間才能跳出迴圈，則此舉非常浪費CPU Time。

1. ■請解釋何為Monitor?

A: Monitor是一個用來解決同步問題的高階結構，是一種ADT(Abstract Data Type)，Monitor之定義主要有3個組成:

1. 共享變數宣告區
2. 一個local functions(or procedures)的集合，供外界呼叫使用
3. Initialization(初始區)

Monitor內之共享資料只能被Monitor內的functions直接存取，外界不可直接存取，只能透過呼叫Monitor提供之functions來間接使用(類似private)。

特性: Monitor本身保障mutual exclusion，即在任何時間點最多只允許1個Process呼叫Monitor內的某個function執行，不會有多個Process同時呼叫Monitor內的functions執行(保障共享資料不會Race Condition)，所以Monitor相較於Semaphore更讓Programmer易於使用。

Condition Type : Condition變數型別是用在Monitor鐘，提供給Programmer解決同步問題使用，令x是Condition Type變數，在x上提供2個操作:

1. x.wait:執行此運作的Process會被Blocked，且置入x變數所屬的Waiting Queue中(預設是FCFS Queue)。
2. x.signal:如果先前有Process在Waiting Queue中，則此操作會從此Waiting Queue中移出一個Process並恢復(resume)其執行，否則無任何作用。

Condition Wait:通常Condition變數所附屬的Waiting Queue預設是FCFS Queue，但有時候我們需要的是Priority Queue。令x是Condition Type變數，x.wait(c)操作就可以將執行此操作的Process置入x變數所屬的Priority Queue中，其中c代表其優先權Level。

Chapter 7

1. ■在Bounded Buffer Problem中，有三個Semaphore，分別是mutex，full和empty， 請解釋mutex的用途。

A: Semaphore mutex的用途是用來對Buffer 變數做互斥控制，防止Race condition。

1. ■在reader-writer problem中，有一般整數變數read count，請解釋該變數 的用途。

A: 在reader-writer problem中，一般整數變數read count是用來統計目前的reader個數，且因可能有多個reader，所以該變數需要Semaphore mutex來做互斥控制。當read count=1時，代表目前此reader process是第一個reader，所以要執行wait(rw\_mutex)，確認有無writer正在寫入，若有則需等待寫入工作完成，若無則可以順利執行讀檔工作(非同時間的第一個reader就不需執行wait(rw\_mutex)，因為也有其他reader在讀，不會Race condition)。當read count=0時，代表沒有reader，執行Signal(rw\_mutex)，將檔案存取權釋放，有writer需要執行寫入工作就可以順利執行。

1. ■請描述哲學家餐桌(Dining-Philosophers)的問題設定。

A: 問題描述:5位哲學家吃中式晚餐，兩兩之間放一根筷子(chopstick)，其中哲學家的狀態分成:

1. Thinking:思考中，不想吃飯
2. Hungry:想吃飯，但必須可以取得左右2根筷子才能吃飯，否則wait
3. Eating:取得左右2根筷子，吃飯中

其中需對筷子(chopstick)做互斥存取控制。

1. ■在最初的嘗試中，使用了wait(chopstick[i])和wait(chopstick[(i+1)%5]依 序檢查能否拿起左右的筷子，這種解法可能會有什麼問題？為什麼？

A: 可能會發生Deadlock，若每位哲學家依序取得左邊的筷子，則接下來每位皆無法取得右邊的筷子形成Deadlock。

1. ■在Dinig Philosopher的解法中，test函式在if條件時，會呼叫slef[i].signal， 為什麼需要這個操作？

A: 在哲學家晚餐問題的解法中，slef[i].signal操作是在test函式中。

當第i個哲學家想吃飯時，執行pickup(i)，先將第i個哲學家的狀態改為Hungry(表達想吃飯)，接著執行test(i)，檢查第i個哲學家的狀態是否想吃飯且是否能夠取得左右兩支筷子(能否順利吃飯的條件)，如果條件全為True，則將狀態改為Eating(吃飯中)且呼叫self[i].signal(先前沒有執行self[i].wait，不做事)，若否，則不做事。然後在pickup(i)內接著判斷執行test(i)後，第i個哲學家的狀態有無被修改成Eating，若為Eating則不會被卡住，可以順利吃飯，若不為Eating則會被self[i].wait卡住(因為左or右有人正在吃飯)。

當第i個哲學家吃完飯時，執行putdown(i)，將第i個哲學家的狀態改為Thinking(不想吃飯)，並給左與右邊的哲學家測試機會，執行test((i+4)%5)與test((i+1)%5)，若左邊哲學家想吃飯且能夠取得他左右邊兩支筷子(代表當初左或右有人在吃飯，被self[(i+1)%5].wait卡住)，則讓左邊哲學家的狀態改成Eating且執行self[(i+1)%5].signal解除該哲學家等待的信號，使其可以吃飯(若左邊哲學家未通過test，則換右邊哲學家test)。

有此可知，需要slef[i].signal操作的原因是當第i個哲學家狀態為Hungry想吃飯，但未通過test，被self[i].wait卡住，代表左或右有人正在吃飯，當那個正在吃飯的人吃完飯，會給他的鄰居機會test，若通過test則會呼叫self[i].signal解救他，使他可以吃飯。

Chapter 8

1. ■有4個條件同時成立時，會造成Deadlock的發生，請列出這四個條件的名稱。

A: Deadlock成立的4個必要條件有:

1. Mutual Exclusion(互斥存取)
2. Hold & Wait(持有並等待)
3. No Preemption(不可搶先)
4. Circular Waiting(循環等待)
5. ■請解釋Deadlock發生條件中的Hold and wait是什麼意思。

A: Deadlock中的Hold and wait指的是Process持有部分資源，並且又在等待其他Process所持有的資源。

1. ■請解釋Deadlock發生條件中的No preemption是什麼意思。

A: Process不可任意剝奪其他Process所持有的資源，必須等待對方釋放資源後，才有機會取得。

1. ■可以利用resource-allocation graph檢查系統的狀態有無deadlock，請問判斷的標準是什麼？

A: 我們可以用RAG觀察並得到以下結論:

1. No Cycle則No Deadlock
2. 有Cycle不一定Deadlock
3. 除非每一類型資源皆是single-instance(單一數量)，則有Cycle 有Deadlock
4. ■請解釋Deadlock Prevention和Deadlock Avoidance的差別。

A: Deadlock Prevention(預防)的做法是要確保Deadlock成立的4個必要條件中的其一必不發生。

而Deadlock Avoidance(避免)的做法是當Process提出某些資源申請，OS會執行銀行家演算法(Banker’s Algorithm)內含Safety Algorithm，檢查若核准申請後系統是否處於Safe State，是則核准Process的申請，否則(unsafe)拒絕Process的申請，Process的申請需等待一陣子後再重提申請。

1. ■請解釋何為Safe State？

A: 可存在(找到)1組”Safe sequence”，使OS可以依此Process order分配其所需要的資源，讓所有Process皆可完工，稱為Safe State(否則為unsafe state)

1. ■在銀行家演算法(Banker’s Algorithm)中，使用的變數有Available/Max/Allocation/Need， 請分別解釋四者的用途。

A: 假設Process個數為n，資源種類數量為m:

1. Available:長度為m的向量，表示系統中目前各種資源的可用數量(即資源總量-Allocation)
2. Max:為n\*m matrix，表示每個Process最多需要的資源數量才可完工
3. Allocation: n\*m matrix，表示每個Process目前持有的各種資源數量(已被配置給Process的資源數量)
4. Need: n\*m matrix，表示每個Process還需要多少資源數量才能完工。(Need[i,j]=Max[i,j]- Available[i,j])
5. ■請解釋Safety Algorithm的運作原理。

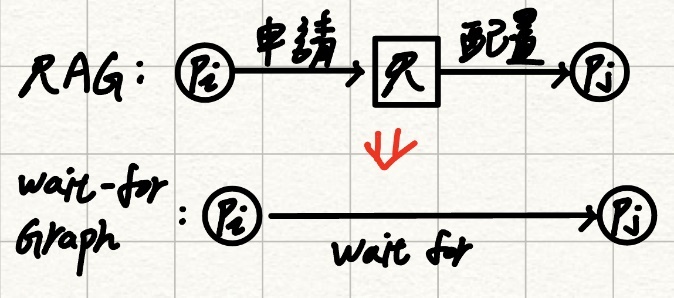
A: Safety Algorithm:

1. 先宣告2個向量:Work=Available，長度m，表示系統中目前各資源的可用數量、Finish[i]=false for i=0,…,n-1，Finish[i]表示第i個Process是否完工
2. 找到某個i使滿足Finish[i]=false且Work，若不存在此i，則跳到第4步
3. Work= Work+，Finish[i]=true，回到第2步
4. 如果對於所有的i，Finish[i]=true，則為Safe State(否則為unsafe state)

Safety Algorithm的運作原理在於:若有Process需要的資源數小於未分配的資源數，則將資源給該Process，使其完工並釋放資源，增加未分配資源數，若依照此作法可讓所有Process完工，則為Safe State(可找到Safe sequence)。若到了第4步有尚未完工(Finish中存在false)，則代表當初是在第2步，Process的需求量都大於未分配的資源數，無法完工(unsafe state)。

1. ■請解釋Resource-Allocation graph和wait-for graph的差異。

A: wait-for graph:從RAG簡化而得，若每種Resources皆為Single-Instance，可以簡化RAG，圖中的頂點種類只有Process(無Resources點)，邊只有”waiting-for” edge，若在wait-for graph中有Cycle，代表有Deadlock。



1. ■請解釋如何使用Safety Algorithm檢查有無Deadlock。

A:承第8題ans，若Safety Algorithm回傳Safe State，即可找到Safe sequence，代表OS可以依此Safe sequence順序分配資源給Process，保證無Deadlock。若是Unsafe State，表示有機會有Deadlock，實際上不一定發生Deadlock(Deadlock是Unsafe State的一個子集)。

Chapter 9

1. ■請分別解釋base register和limit register的用途。

A: base register中紀錄的是Process在記憶體中的起始位置，相對位址加上base register中的值即是實體位址，在MMU中base register則稱為relocation register。而limit register則是記錄Process的大小，用來判斷存取記憶體位址的範圍大小是否合法。

1. ■請解釋Logical address和Physical address的差別。

A: Logical address是由CPU生成的，而Physical address則是實際去實體記憶體(RAM)存取的位置。而Logical address轉為Physical address需要MMU介入。

1. ■請解釋何為動態連結(Dynamic linking)。

A: 在Execution Time，若Module被呼叫到，才將之載入，並與其他Modules進行Linking修正(外部符號之解決)，適用在Library Linking(ex Dynamic Linking. Library,DLL)。

優點:節省不必要之Linking time

缺點: Process執行時間較久

1. ■請解釋何為連續記憶體配置(Contiguous Allocation)

A: 即OS在為Process配置記憶體空間時，必須配給Process一個連續的Free Memory Space，而每個Process所占的Memory Space稱為一個Partition。

1. ■請解釋記憶體管理中的洞(hole)的定義。

A: 在連續記憶體配置下，Memory中會有一些Free Memory Space(or Block)稱為Hole。通常OS會用Link-List觀念管理Holes，稱為AV-List(Available-List)。

1. ■在選擇合適的記憶體位置配置一個程序的資料時，採用best-fit的好處是 什麼？

A: best-fit的作法是:

當Process大小為n時，check AV-List中所有的Holes，找出一個Hole，滿足其Hole sizen且Hole size-n的差值最小，將此區塊配置給Process。

採用best-fit的好處是空間利用度會是最佳的

1. ■請解釋何為外部碎裂(external fragmentation)

A: 在Contiguous Allocation要求下，AV-List中任何一個Hole size均小於Process size，但這些Hole size總和卻大於等於Process size。因為這些Hole並不連續，因此無法配置給Process，造成空間閒置不用，Memory利用度低。

1. ■請解釋何為記憶體的Compaction。

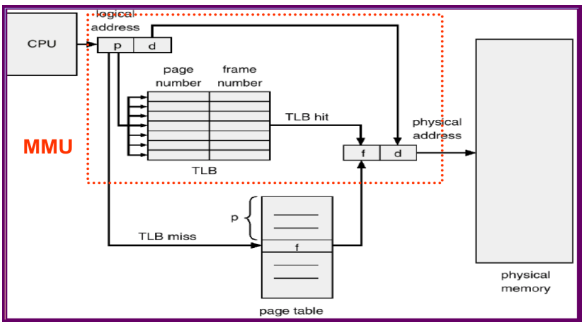
A: Compaction是解決外部碎裂方法中的其中一種，做法是移動執行中的Process，使原本非連續的Hole可以聚集形成一個夠大的連續Free Memory Space。Compaction的困難點在於不易製定最佳的Compaction策略、Process必須是Dynamic Binding才可在執行期間移動。

1. ■請解釋何為分頁表(Page Table)

A: Page Table是在Page Memory Management下，因為採取非連續性配置，OS會替每個Process建立Page Table，紀錄各個Process配置於哪個Frame之Frame number(Frame不一定要連續)。若Process size = n個Pages，則他的Page Table就會有n個entry。

1. ■請解釋Translation lookaside buffer的用途。

A: Translation lookaside buffer(TLB)，或稱Associative Registers，is a associative high-speed memory(類似Cache)，是用來保存Page Table中經常被存取之entry內容(page number與Frame number)。MMU會先到TLB查詢page number是否命中(hit)，若hit則直接取出Frame number，若miss則仍需到Memory中查詢Page Table。



1. ■請解釋Valid-invalid bit的用途。

A: 使用Valid-invalid bit是在Memory Protection中的其中一種作法。為了區分page內容是否可以被Process存取，可以在Page Table中多加一個Valid-invalid bit欄位，值為V或I。V表示Process可以存取該page，I則表示Process不能存取該page。此外，此bit會用在Virtual Memory中，表示page在(V)或不在(I) Memory中。

1. ■在分頁表的實作中，其中一種是使用階層式分頁(hierarchical paging)，請 解釋使用此方法的理由。

A: 階層式分頁(hierarchical paging)也稱MultiLevel paging、paging the page table、forward paging，此方法可以解決Page Table size太大的問題。

做法:不要一次將整個Page Table整個載入到Memory中，而是載入Process執行所需之Page Table內容即可，所以要對Page Table的組織架構調整，採多層式(階層式) Page Table結構。

以Two-Level paging為例:

Logical address格式:

|  |  |  |
| --- | --- | --- |
| Level-1(x bits) | Level-2(y bits) | d |

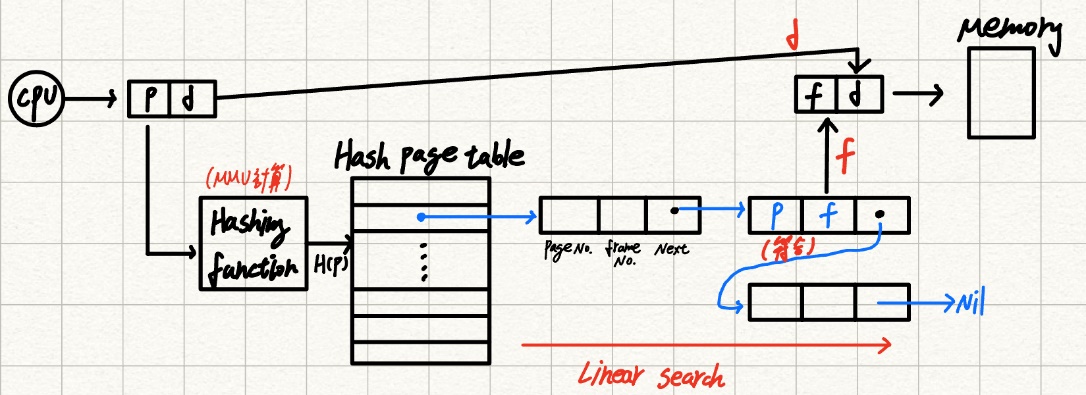
Level-1 Page Table中有個entry，每個entry紀錄某個Level-2 Page Table位址(指標)，共可記個Level-2 Page Table。而每個Level-2 Page Table中有個entry，每個entry紀錄Frame number。可以此類推更多層。

缺點:EMAT(Effective Memory Access Time)更久，因為需多次Memory Access

1. ■在分頁表的實作中，其中一種是使用雜湊分頁表(Hashed Page Table)，請 解釋使用此方法的理由。

A: 雜湊分頁表(Hashed Page Table)也是可以解決Page Table size太大的問題。作法:將Page Table以Hashed Table方式表現，且採chain方法處理overflow。Page Table與Hashed Table具有相同Hashing Address之page number與他們的Frame number，並保存在Link List中，Node Structure為:

|  |  |  |
| --- | --- | --- |
| Page No. | Frame No. | Next |

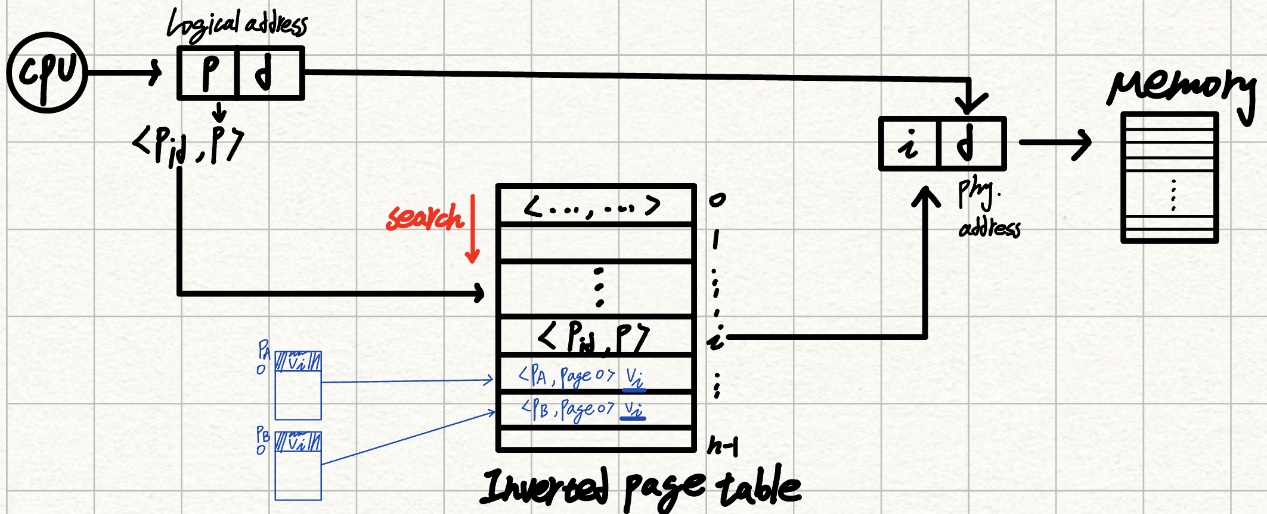


缺點:使用Linear Search在Link List中找符合的Page number較耗時

1. ■請解釋反向分頁表(Inverted Page Table)的運作原理。

A: 改以”Physical Memory”為記錄對象，而不是每個Process有各自的Page Table。

整個系統只有一份表格(Table)，若Physical Memory有n個Frames，則Inverted Page Table有n個entry，每個entry紀錄<Process ID,Page No.>代表此Frame被哪個Page占用。



例: Physical Memory=8GB，Page size=16KB，Page Table entry佔4 bytes，則Inverted Page Table size?

Physical Memory有8GB/16KB=個Frames，所以Inverted Page Table有個entry，共\*4 bytes=2MB

優點:有效縮減Page Table size

缺點:一一比對<Pid,P>，Search耗時、喪失(無法支援)原本Memory Sharing之效益

1. ■請解釋何為swap空間及使用swap的理由。

A: 當Memory空間不夠時，OS會挑選Process，將該Process轉移到swap空間(Backing store)，並加到ready queue中等待，釋放Memory空間給其他Process使用。優點:使用Memory空間較有彈性、提高可靠性。