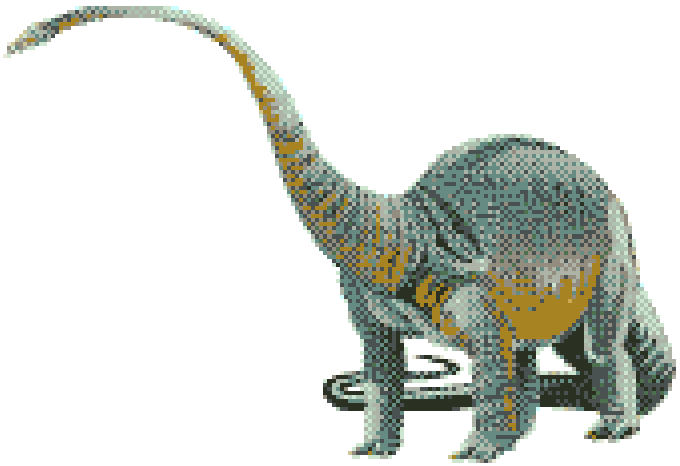
作業系統(Operating Systems)

Course 5: Thread (執行緒)



授課教師:陳士杰

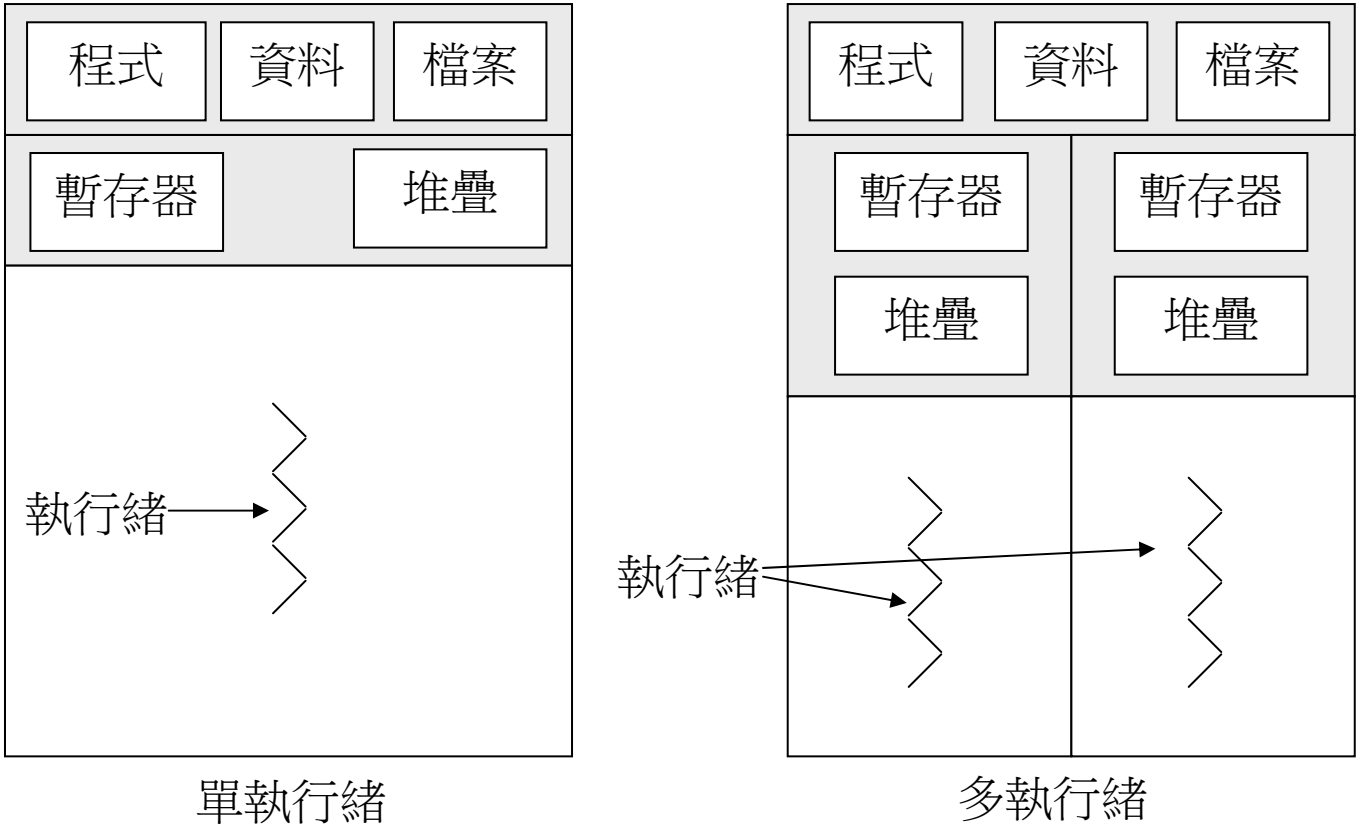
國立聯合大學　資訊管理學系

■本章重點

- Thread定義
- 與Process的比較
- User Level v.s. Kernel Level Thread
- Models種類
- Thread Pool

Thread (執行緒)

- **Def:**
  - 又稱為 **Lightweight Process (LWP; 輕量級行程)**，是OS分配CPU Time的對象單位。
  - 和Process相似，也是搶奪CPU的一個基本單位。
- 每個Thread擁有下列項目：
  - **Thread ID (執行緒的識別碼)**
  - **Thread State**
  - **Program counter (程式計數器)**
  - **Register set (暫存器組)**
  - **Stack (堆疊)**
- 同一個Task (Process)內的Threads彼此共享：
  - **Code Section (程式碼區域)**
  - **Data Section (資料區域)**
  - **O.S. Resources (作業系統資源)**
- 傳統的Process (Heavyweight Process; 重量級行程)就等同於一個Task (Process)內**只有單一個Thread**。



單執行緒
(即：Process)

多執行緒

- **Process和Thread只有內部OS較能看得出來，外觀看不出來!!**

Thread vs. Process

Thread	Process

Motivation (動機)

- 目前許多在桌上型電腦所執行的應用程式都是屬於多執行緒運作。
  - 網頁瀏覽器
    - Display images or text
    - Retrieve data from the network
  - 文書處理器
    - Display graphics
    - Read keystrokes
    - Check spelling and grammar

Benefits (利益)

- **Responsiveness (應答)**
  - 多執行緒在一個交談互動的應用中，允許**程式中的某一部份當被中斷或是執行得非常久**時，該程式仍然可以繼續執行。(**一個Process內只要還有一個Thread還在Run，則該Process還可再執行**)
- **Resource Sharing (資源分享)**
  - 執行緒間共用著它們所屬行程的記憶體和資源
- **Economy (經濟)**
  - 採用執行緒的話，因為**執行緒的產生**和從事**Context Switch**共用它們的所屬Process的記憶體和資源，所以在實行上比較經濟
- **Utilization of MP Architectures (使用多處理器架構)**
  - 如果是在多處理器架構下實施多執行緒，因為每一個執行緒可以並行地在不同的處理器上執行，因此多執行緒的利益可以大幅提升。
  - 在單一處理器的架構下，CPU讓每一個執行緒快速地切換移動，讓使用者誤以為是同時在進行。

Thread 種類

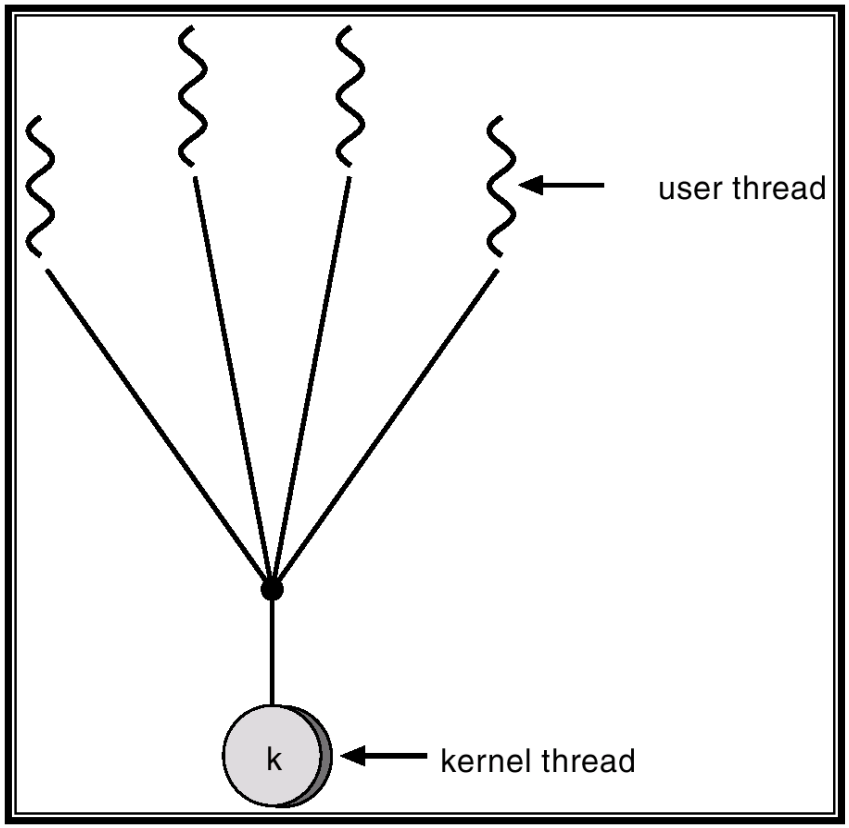
- 分類角度：**Thread Management** (e.g., Creation, Destroy, Context Switching, Scheduling, etc.) 是由誰去掌控？
- 執行緒可以依其執行時**所處模式**的不同，區分成兩個模式：
  - The user level -- **User Threads (使用者執行緒)**
    - 在User Mode下進行，O.S.不知道有這些Thread存在
    - 不需要O.S.介入管理
  - The kernel level -- **Kernel Threads (核心執行緒)**
    - 在Monitor Mode下進行，O.S.知道有這些Thread存在
    - 由O.S.介入管理

User Thread	Kernel Thread

Multithreading Models (多執行緒模式)

- Many-to-One (多對一模式)
- One-to-One (一對一模式)
- Many-to-Many (多對多模式)

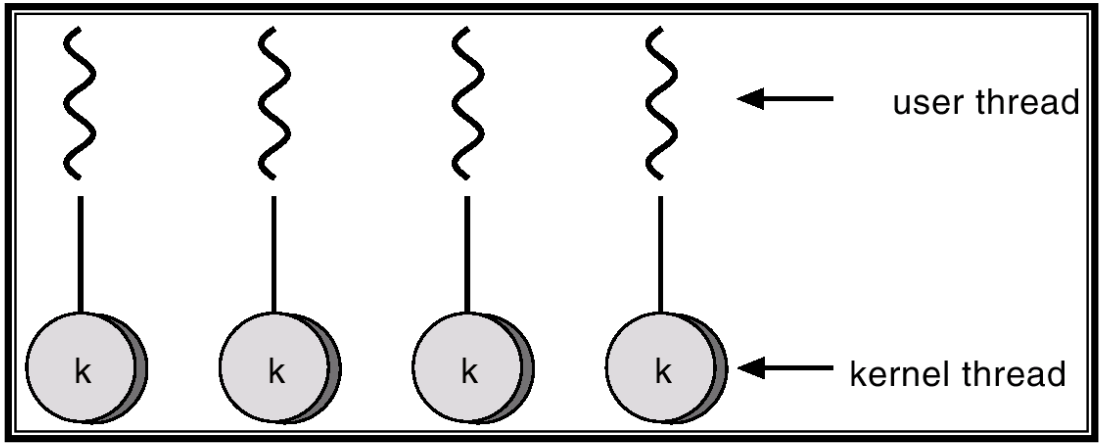
Many-to-One Model (多對一模式)



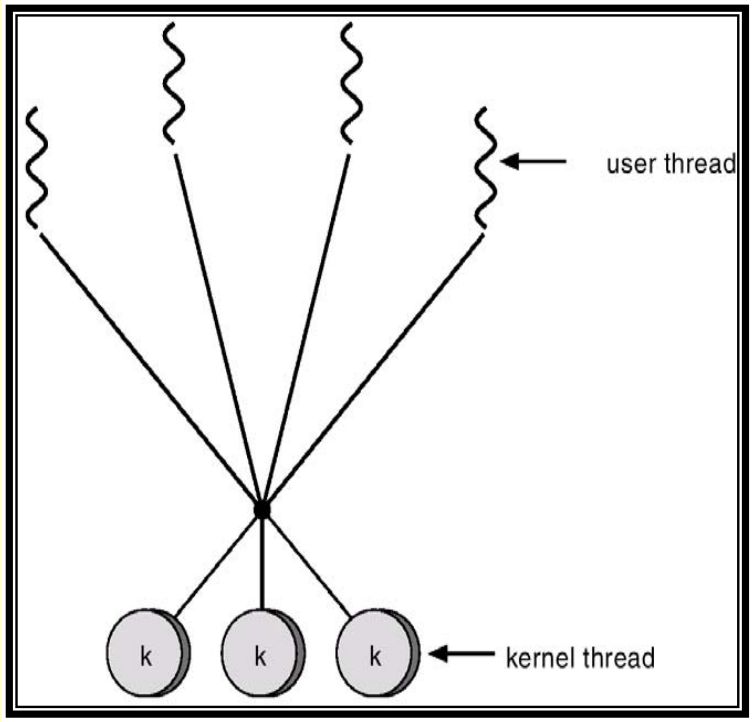
- 多對一模式是指**多個User Threads**對應到**一個Kernel Thread**。
- 執行緒主要的管理動作是在**使用者空間**執行，所以很有效率。
- 缺點是如果有任何一個User Level的Thread執行暫停的系統呼叫，將**造成整個行程暫停執行**。
- 雖然這個模式可以產生它所需要的Thread數量，但因為**只有一個Kernel Thread**可以存取Kernel，O.S.一次只能使用一個執行緒，且O.S.不知道有這些User Threads存在，無法將Block以外的Thread配給其它的處理器，所以**數個執行緒不能在多個處理器上並行**執行。

One-to-One Model (一對一模式)

- 每一個User Thread都對應到一個Kernel Thread。當一個使用者執行緒處於暫停或是等待的狀態時，其它的執行緒都還可以執行。
- 它比多對一模式提供了更多的並行功能，也允許多個Threads在Multiprocessor上並行執行。
- 產生一個User Thread時，需連帶產生一個Kernel Thread，而Kernel Thread會對程式的執行產生一些額外的負擔。∴此模式限制執行緒產生的個數。
- **Examples**
  - **Windows NT/2000**
  - **OS/2**



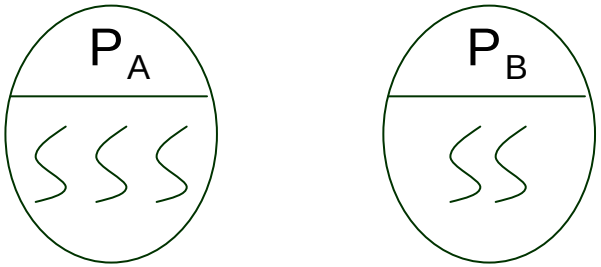
Many-to-Many Model (多對多模式)



- **多個User Threads**對應到**多個(個數少於或等於使用者執行緒個數)Kernel Threads**。
  - 多對一模式可以產生它所需要的執行緒數量，但因為核心一次只能使用一個執行緒，所以數個執行緒**不能真正地在多處理器的環境下並行執行**。
  - 一對一模式雖然可以提供較強的並行能力，但**不能產生太多的的執行緒**。
- **Programmer或O.S.**可以產生所需要的**Thread**數目，使其在多處理器上並行執行；另外，當一個**Thread**暫停執行時，**O.S.**可以安排另一個**Thread**接著執行。

❖範例❖

- 有兩個Process A與B



若OS採平均分配原則來分配CPU Time，則P_A與P_B各分多少%之CPU Time。1) User Thread, 2) Kernel Thread

Ans:

1）∵Kernel不知道有User Thread，只知道有P_A與P_B兩個Process。∴ P_A與P_B各分到50%的CPU Time。

2）∵有5條Thread欲分配，每條可分到20%的CPU Time。∴ P_A分到3×20% = 60%的CPU Time；P_B分到2×20% = 40%的CPU Time 。

Thread Pools (執行緒池)

- 每當O.S.收到一個要求時，就會產生一個個別的Thread來服務此要求。
- Multithread可以產生很多的Thread，但是可以產生多少呢?
  - 產生Thread需要花費時間
  - 如果無限制地生產，可能會耗盡系統的資源
- 有一個可能的解決方法就是Thread Pool的觀念。
- Def:
  - 一個Process開始執行時，產生了一些Thread，並將這些Thread放到一個Pool中以等待工作。
  - 當有一個工作要求產生時，就從這個Pool中喚醒一個Thread給要求者來執行其所需的工作。
  - 當工作完成後，Thread就回到Pool中等待其它的工作。
  - 如果一個工作要求產生，而Pool中沒有Thread可以執行時，這個工作就要等待直到有為止。

- **Thread Pool的好處是：**
  - 對於一個服務要求而言，使用**現存的Thread**比等待產生一個Thread較來得快。
  - 執行緒池限制了任何時間點上**Thread的個數**，不會讓系統的資源耗盡 (Pool中沒有空閒的Thread時，不會再產生新的Thread)。這對於**有資源上的限制**之系統非常重要。
- **Thread Pool中的Thread個數可以根據：**
  - CPU的個數
  - 實體記憶體大小
  - 預期客戶要求的個數