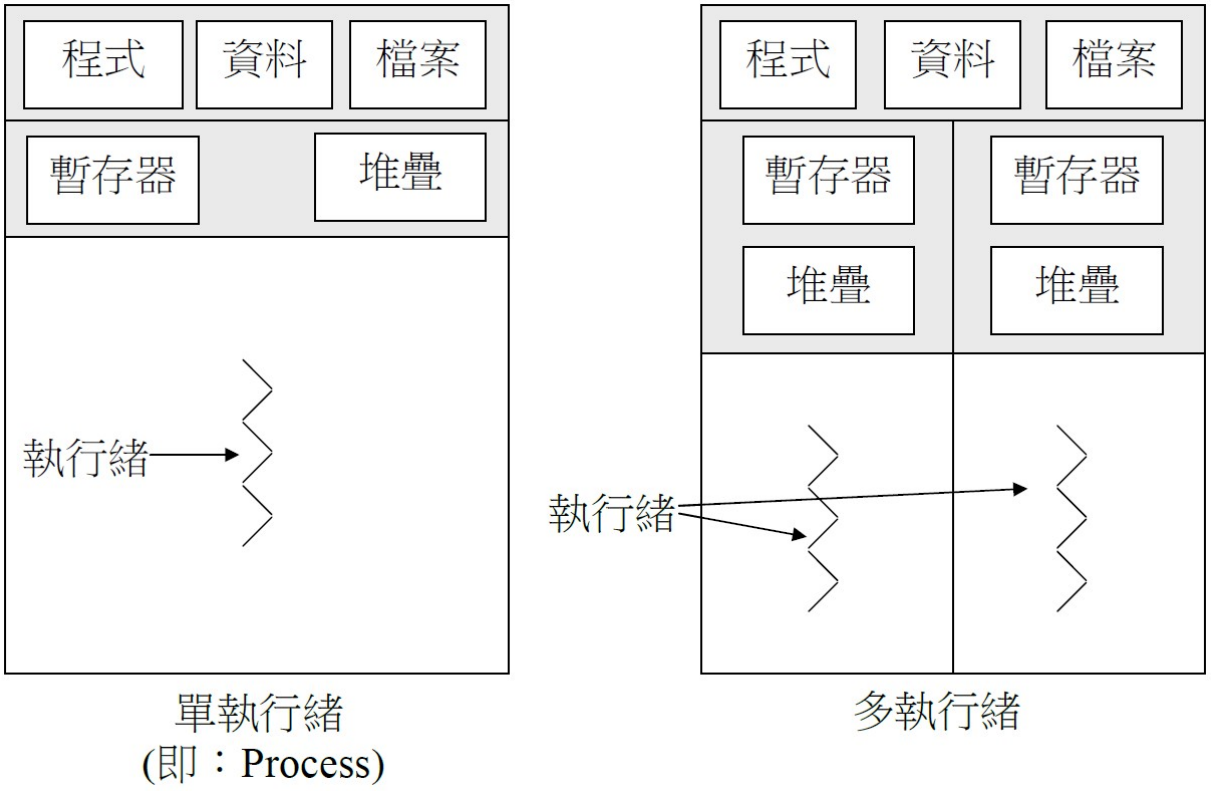
執行緒

Thread 執行緒

- **Lightweight Process**(LWP) 輕量級行程
- OS 分配 CPU Time 的對象
- 每個 Thread 擁有下列項目
  - Thread ID 執行緒識別碼
  - 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 可以共享
  - Code Section 程式碼區域
  - Data Section 資料區域
  - OS Resource 作業系統資源
- 傳統的 Process 等同於一個 Task 內**只有一個 Thread**



動機

- 當 Process 發出 Blocking System Call (e.g. I/O Request)
  - OS 會將 Process 移出 CPU 等待
- 如果是由 Process **其中一個 Thread** 發出 Blocking System Call
  - OS 會 Lock 發出 Blocking System Call 的 Thread
  - CPU 控制權交給 Process 中其他 Thread

Benefits 利益

Responsiveness 應答

- **允許程式中某個 Thread 被中斷或執行非常久**，該程式仍可以繼續執行
- **一個 Process 內只要還有一個 Thread 在執行，則該 Process 還可再執行**

Resource Sharing 資源共享

- 執行緒所屬行程的記憶體和資源共享

Economy 經濟

- Context Switch 共用所屬的 Process 記憶體和資源

Utilization of MP Architectures 多處理器架構

- 每個執行緒可以並行在不同處理器上執行

Thread 種類

- Thread Management
- **依照所處模式分類**
  - **User Threads 使用者執行緒**
    - 在 User Mode 執行
    - OS 不知道有 Thread 存在
    - OS 不介入管理
  - **Kernel Threads 核心執行緒**
    - 在 Monitor Mode 執行
    - OS 知道有 Thread 存在
    - OS 介入管理

User Thread	Kernel Thread
User Level 的 Thread Library 支援管理	Kernel 負責管理
Kernel 不知道 User Threads 存在	Kernel 掌握所有 Threads
Thread 管理成本低(對 OS 來說)	Thread 管理成本高(對 OS 來說)
Context Switch, Creation 快，因為 OS 不須介入	Context Switch, Creation 慢
User Thread 發出 Blocking System Call，會導致整個 Process Blocked	Thread 發出 Blocking System Call 不會使整個 Process Blocked
無法有效利用 Multiprocessor	可以安排不同 Thread 在不同 CPU 執行，發揮 Multiprocessor 效益

Multiprocessor

Many to One 多對一

- **多個 User Thread**對應**一個 Kernel Thread**
- 管理工作是在**使用者空間**執行，有效率
- **缺點**
  - **如果有一個 User Level Thread 發出 Blocking System Call，將造成整個 Process 暫停執行**
  - 雖然可以產生所需要的 Thread 數量，但是**只有一個 Kernel Thread**，OS 一次只能使用一個執行緒，且 OS 不知道其他 User Thread 存在，無法將 Block 以外的 Thread 分配給其他的處理器，**不能在多個處理器上並行執行**

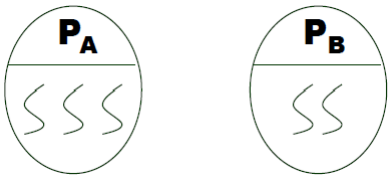
One to One 一對一

- 每一個 User Thread 都對應到一個 Kernel Thread，當一個 User Thread Blocked，其他 Thread 還可以執行
- **並行執行的功能**多個 Threads 在 Multiprocessor 上並行執行
- **必須限制執行緒產生的個數**

Many to Many 多對多

- 多個 User Threads 對應到多個 Kernel Thread
  - Kernel thread <= User Threads

例題



- 有兩個 Process A(3 Threads) 和 Process B(2 Threads)
- 若 OS 採用平居份配原則，P_A 和 P_B 各分配多少百分比的 CPU Time
  1. User Thread
  2. Kernel Thread
- **Ans**
  1. Kernel 不知道有 User Thread，所以 CPU Time --> PA = PB = 50%
  2. 每個 Thread 平均分配到 20% 的 CPU Time
     - PA = 60%
     - PB = 40%

Thread Pools 執行緒池

- **產生 Thread 需要花費時間(產生時間差)**
- **如果無限制產生 Thread，可能會耗盡系統資源**
- **概念**
  - 一個 Process 開始執行時，產生一些 Thread 並將它們放到 Pool 中等待工作
  - 當有一個工作要求產生，從這些 Pool 喚醒一個 Thread 給要求者來執行所需的工作
  - 工作完成後，Thread 就回到 Pool 中等待其他工作
  - 如果有一個工作要求產生，但是 Pool 沒有 Thread 可以執行，這個工作要等待到有為止
- **優點**
  - 使用**現存的 Thread**比等待產生一個 Thread 來的快
  - Thread Pool 限制**Thread 的個數**，Pool 沒有空閒的 Thread，就不會再產生新的 Thread
- Thread Pool 中的 Thread 個數可以根據
  - CPU 個數
  - 實體記憶體大小
  - 預期客戶要求個數