Thread也被稱為light weight process

Thread是使用CPU最基本單位

\*共用 VS 獨立

同process的Thread間共用 code、data(全域)、OS resource

每個thread也有自己thread control block(program counter、register、stack(local variable)、thread ID)

\*thread 好處:

1.及時反應(responsiveness): 不會因為執行其中一個程式而無法回應其他request

(server request&response、遊戲物件偵測)

2.分享資源(resource sharing): 很多thread在同一塊memory、可以有效率的溝通、共享資源

3. 有效運用Muiti-processor效能: thread能實現平行化運算

4. 有效率: thread比較容易建造(process要造Data section、比thread多的meta)、比較快context switch，並且減少access memory(共用memory)

Example

Web browser、web server、RPC server都會應用thread

\*Multithcore programming

Multithreader programming: 更有效率應用multiple core、增加同步化 (thread可平行處理)

Multicore system: 讓使用者、應用程式有使用thread的誘因

(以前是增加CPU時脈，到瓶頸後改增加core)

\*MP問題

1.如何切割工作 dividing activities

2.如何分割資料 data splitting

3.資料同步化問題 (同時讀寫問題) data dependency

4.平衡 (均等切割問題) balance

5.測試與除錯 testing and debugging

(跨core除錯會複雜(分散資料、加上執行先後順序))

\*User thread VS kernal thread

User thread: thread由user-level的thread library管理

通常較為快速(沒有system call的話)

Kernel thread: 直接由OS管理

通常較為緩慢

Kernel會schedule thread的執行

\*binding

兩者是binding 關係: user thread在使用時會連接到 kernel thread

所以如果kernel thread只有single thread，實際上相當於只有process

會在執行時動態調整bind的kernel thread

可能

多對一(kernel single thread)

缺點:

Blocking 問題 (kernel thread卡住)

沒有平行化運算

優點:

但是OS管理簡單

一對一 (大部分OS採用)

缺點:

限制user thread於kernel thread的數目

Overhead:創建user thread一定要找一個kernel thread

優點:

平行化

同步化

多對多

缺點:

複雜

Overhead可能大於帶來的效率

優點:

可無限制創造user thread

如果一kernel thread block，能夠重新分配一個