CH4、Thread

Thread 之管理

目錄：

Thread

(一)Def：又叫"Lightening Process"，是 OS 分配 CPU Time 的對象單位

[恐]It's a basic unit of CPU utilization

Thread 建立之後，其私有的(private)內容組成含有：

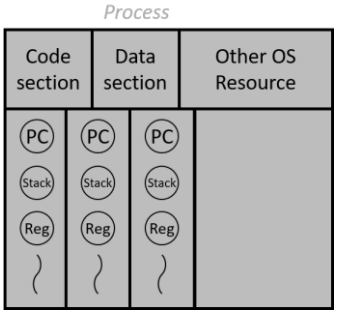
Process Counter、CPU registers value、Stack、Thread ID、State…等

Note：record in TCB (Thread Control Block)

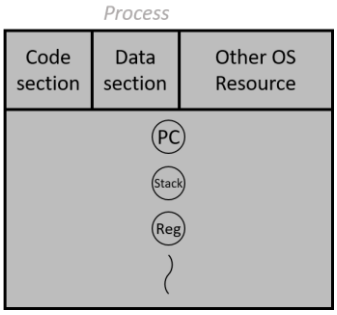
此外，同一個 Process 內之不同 Threads 彼此共享此 Process 的：

Code Section、Data Section*(上二者合稱 Memory (Address) Space)*、other OS Resources(ex：open file、I/O Resources、Signal…等)*(PC, Stack, Reg 不共享)*

1. Multi-Threading Model



2. Process = Single-Thread Model



Note：類比：Process=飛機、Thread=引擎

(二)優點

1. Responsiveness

   Def：當 Process 內執行中的 Thread 被 Blocked，則 CPU 可以切給此 Process 內其他 Available Thread 執行，故整個 Process 不會被 Blocked，仍持續 going，所以若將 Multithreading 用在 User-Interactive Application，可增加對 User 之回應程度

2. Resource sharing

   Def：因為 Process 內之多條 Threads 共享此 Process Code Section，所以在同一個 Memory Space 上，可有多個工作同時執行

3. Economy

   Def：因為同一個 Process 內之不同 Threads 彼此共享此 Process 的 memory 及 other OS resources，所以 Threads 之私有成份量少，故 Thread 之 Creation、Context Switching、fork…等 Thread Management Cost is cheap

4. Scalability (Utilize of Multiprocessors Architecture)

   Def：可以作到同一個 Process 內之不同 Threads 可以在不同 CPUs 上

平行執行，可以增加對 Multiprocessors System 之效益(平行程度)提升

Process vs Thread

Heavy Weight Process	Lightening Process
Single-Threaded Model	Multithreading Model
是 OS 分配 Resources 之對象單位	是 OS 分配 CPU Time 之對象單位
不同的 Process 不會有共享的 Memory 及 other resources *(除非是採用 shared memory 溝通)*	同一 Process 內之 Thread 彼此共享此 Process 之 Memory 及 other resources
若 Process 內的 single Thread 被 Blocked，整個 Process 亦被 Blocked	只要 Process 內尚有 available Thread 可執行，整個 Process 不會被 Blocked
Process 之 Creation、Context Switching 慢，管理成本高	Thread 之 Creation、Context Switching 快，管理成本低
對於 Multiprocessors 架構之效益發揮較差	對於 Multiprocessors 架構之效益發揮較佳
Process 無此議題 *(除非是採用 shared memory 溝通)*	因為同一個 Process 內之 Threads 彼此共享此 Process Data Section，因此必須對共享的 Data 提供互斥存取機制，防止 race condition

Process 與 Thread 並沒有功能上的差別，只有『效能』上的差別*(飛機都能載人，但引擎數可能不同)*

哪此功作適合用 Multi-threading？

例：同一時間可以執行多個工作：Client-Server model

反之不適合使用 Multi-threading：

例：一個時間點最多只有一個工作可執行，ex：命令解譯器(UNIX 之 shell)

Thread 分類：User Thread 與 kernel Thread

區分角度：Thread Management 工作(如：Thread creation、destroy、suspend、wakeup、scheduling、context switching)由誰負責

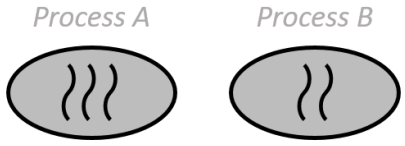
一、User Thread

1. Def：Thread Management 是由在 user site 之 Thread library 提供 APIs，供 user process 呼叫使用。
   Kernel 完全不知道(is unaware of) user thread 之存在
   Note：只知道有 process(single thread) Thread Management 不需 kernel 介入干預
2. 優點：Thread creation、context switching 等管理較快，成本較低
   缺點：
   1. 當 process 內某條執行中的 user thread is blocked，會導致整個 process 亦被 blocked*(即使 process 內還有其他 available Threads)*
   2. Multiprocessors 架構效益發揮較差*(因為 kernel 只看到一個 process、無法作到 Process 內之多條 user threads 平行執行)*
3. 例：舉凡 Thread library 皆是 user threads
   Ex：POSIX 的 PThread library、Mach 的 C-Thread library、Solaris 2 以上的 UI Thread library、Green Thread library

二、kernel Thread

1. Def：Thread 管理完全是由 kernel 負責，kernel 知道每一條 Thread 的存在，並進行管理
2. 優缺點：與 User Thread 相反
3. 例：大部分 OS 皆支持：Windows 系列(2000、NT)、UNIX、Linux、Solaris

Ex：[Modern 版]

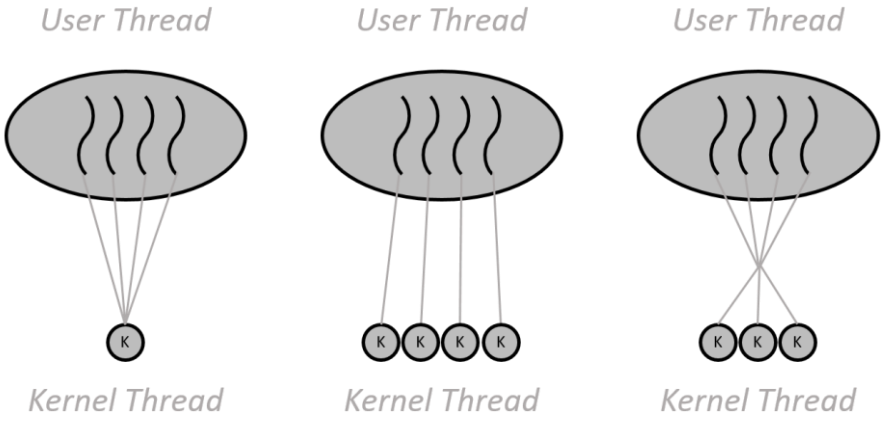


CPU Time 依分配對象數、平均分配，則 PA、PB 各分到多少%？
CPU Time if all Threads are 1.User Thread、2.kernel Thread

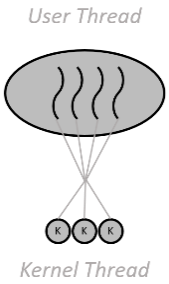
1. *user Thread：kernel 只知道有 2 個 Process 要分 CPU Time，所以：PA 與 PB 各為 50%、50%*
2. *kernel Thread：kernel 知道有 5 條 Thread 要分：PA 分到 60%、PB 分到 40%*

Multithreading Mode(3 種)

1. Many-to-One Model
   (一) Def：This model maps many user Thread to one kernel Thread. Thread Management is done in user space
   (二) 優缺點：同 user-Thread
   (三) 例：Thread library 皆是
2. One-to-One Model
   (一) Def：This model maps one user Thread to one kernel Thread. Thread Management is done in user space
   (二) 優缺點：同 kernel-Thread
   (三) 例：現代個人電腦大都採用此 Model：Windows NT、2000、OS、”Linux”
3. Many-to- Many Model
   (一) Def：This model maps many user Thread to a smaller or equal number of kernel Thread. Thread Management is done in user space
   (二) 優點：User thread 的第 1、2 點，再加上 3.負擔不若 one-to-one 重
   缺點：User thread 的第 1 點，再加上 2.製作設計上較複雜
   (三) 例：Solaris 2 以上(two level modeling)



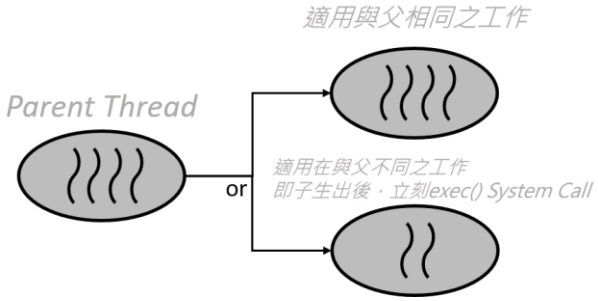
例：下圖中若有 2 個 IO-Bound thread，請問 kernel thread 使用數為何？



2 個給 IO Bound thread、剩餘 1 個給 CPU Bound thread 使用

Multithreading issues

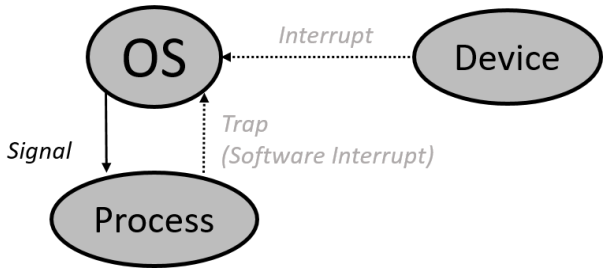
1. fork() issue



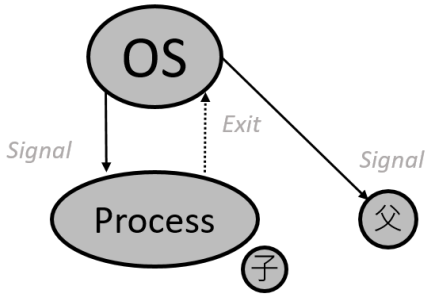
2. “Signal” delivery issue

(1) Signal：It’s used in UNIX to notify the process that a particular event has occurred

1. 當 Process 要結束，或者有 Device 要結束一個 Process 時，會分別發出 Trap 與 Interrupt 給 OS，而後 OS 再 Signal 到 Process，使之結束



2. 當子行程結束時，會先發出 Exit 訊號給 OS，OS 會 Signal 通知父行程，並且也 Signal 給原本的子行程，以結束之



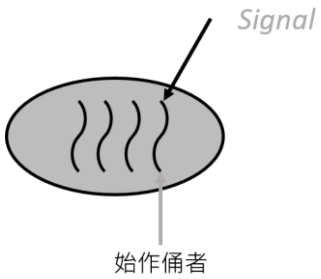
當 Process 收到 Signal 之後，它必須處理，可由 Process 自己處理、或交給 default signal handler 處理

(2) Signal 種類：

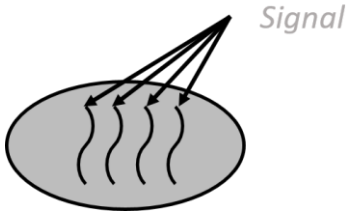
1. synchronous signal(自作自受型)：ex：Divide-by-zero、illegal Memory Access
2. Asynchronous signal(池魚之殃型)：ex：”control-c(break)” by administrator、”Time out” by Timer

(3) Signal Delivery issue 的 4 個 options

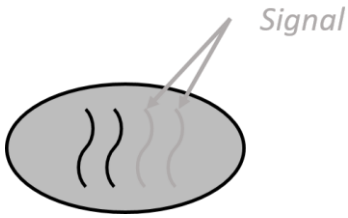
1. Ex：Synchronous signals



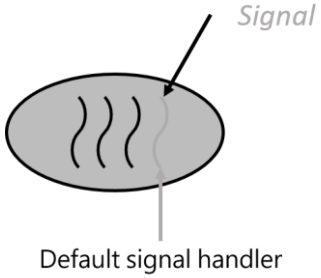
2. Ex：control-break



3.



4. Ex：solaris



3. Thread pool

(一) 緣由：在 Client-Server Model 中，當 Server 收到 Client's Request 後，Server 才建立 Thread 去服務此一請求。然而 Thread Creation 仍需耗用一些時間。因此對 Client 之回應不是那麼迅速

(二) 解法：採用"Thread Pool"機制。Process(Server)事先建立一些 Threads，置於 Thread Pools 中，當收到 Client's Request 後，就從 Thread pool 中指派一條 Available 的 Thread 去服務此請求，不需 Creation。因此回應較快當此 Thread 完成工作後，再回 Thread pool 中 stand by。如果 Threads Pool 中無可用的 Threads，則 Client's Request 需等待

Note：通常 OS 會限制 Thread pool size

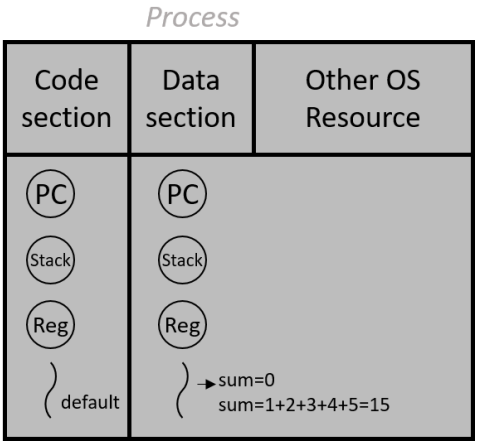
Thread 程式追蹤(以 PThread library 為例)

例 p4-49：

```
#include <pThread.h>
void *runner(void *param)
main()
{
        pThread-t tid;      //tid 變數表 ThreadID
        pThread-attr-t attr;       //表 Thread attributes set

        pThread-attr-init(&attr);
        pThread-create(&tid, %attr, runner argv[1]);
        //根據 attr 屬性值，建立一條 Thread。ID 記在 tid 中，執行 running 副程式
        pThread-join(tid, NULL);
        //main thread 在此，等待直到 tid thread 結束
        printf(sum);
}

void *runner(void *param)
{
        int sum;
        sum=0;
        for(i=0;i<upper;i++)
                sum+=i;
        pThread-exit(0);
}
```

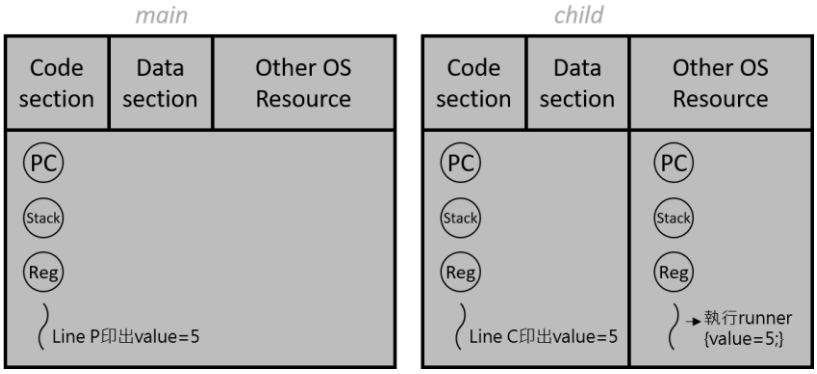


例 p4-50：

```
#include <pThread.h>
int value = 0;
main()
{
        int pid;
        pThread-t tid;
        pThread-attr-t attr;

        if(pid==0)
        {
                pThread-create(&tid, %attr, runner argv[1]);
                printf(value);       //line C
        }
        else if(pid>0)
        {
                wait();
                printf(value);       //line P
        }
}

void *runner()
{
        value=5;
        pThread-exit(0);
}
```



例 p4-82(43)：Shared by thread

1. Static local variables
2. Program text/exec. Binary
3. Register valve of CPU
4. Heap Memory
5. Programing Counter
6. Stack Memory
7. Open file
8. IO recourses
9. Local variables
10. Global variables

共享：1、2、4、7、8、10
私有：3、5、6、9

[補充]：Symmetric Multi-Threading
在相同的 Physical Processor 上，創造多個 Logical processor，讓多個 Threads 可以平行處理

前面有提到 Multi-Threading 之適不適用情況

哪此功作適合用 Multi-threading？
　　例：同一時間可以執行多個工作：Client-Server model
反之不適合使用 Multi-threading：
　　例：一個時間點最多只有一個工作可執行，ex：命令解譯器(UNIX 之 shell)

故當需要執行一個時間內，主要只有『單一』工作要執行時，不使用 Mutli-Threading，而是使用 Symmetric Multi-Threading(SMT) on Symmetric Multi-Processors(SMP)