多核處理器上的 MMU 和 TLB

https://blog.csdn.net/gzxb1995/article/details/104910787/
i

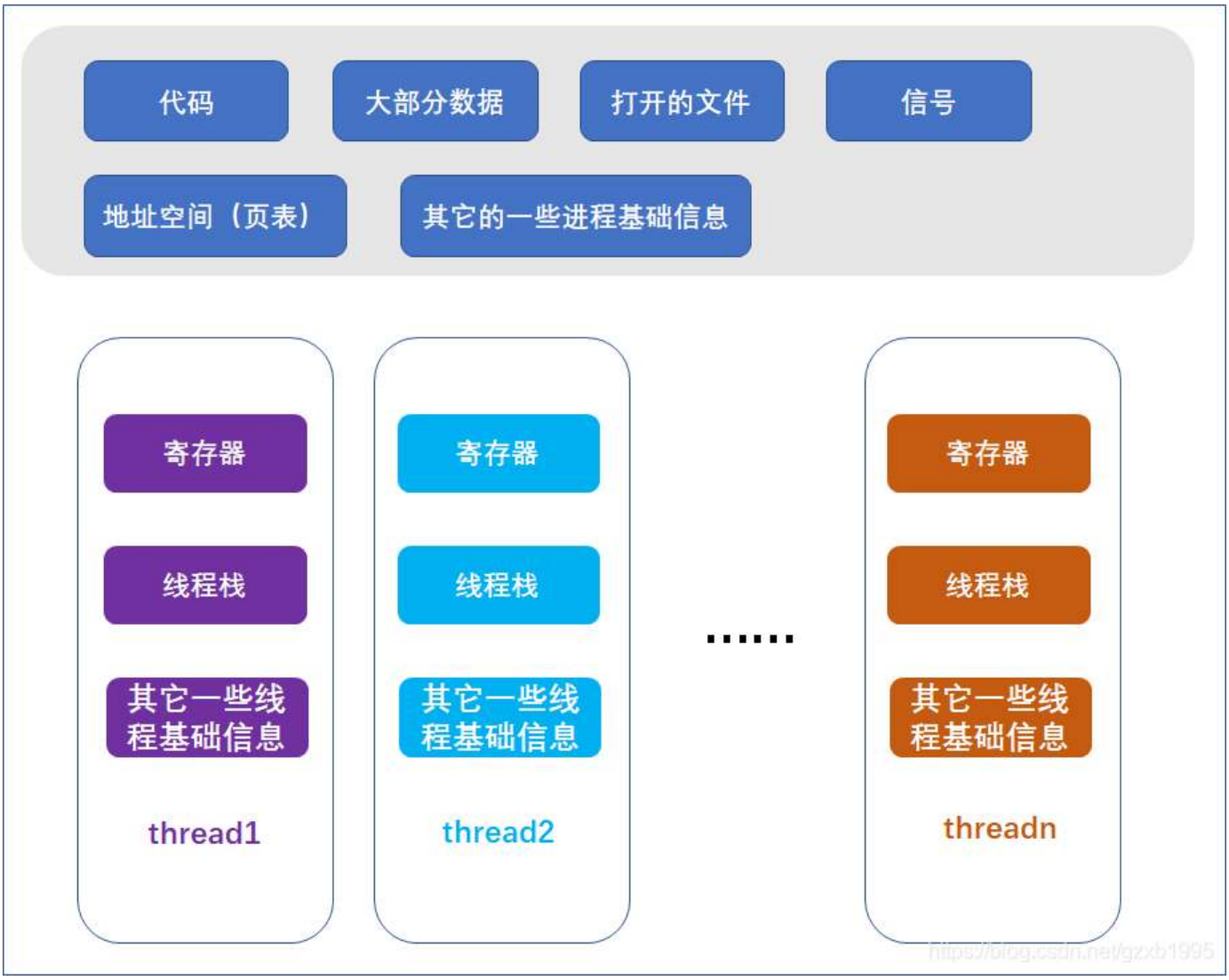
冉冉雲 2020-03-17 20:37:24 1577 收藏 8
分類專欄： 計算機體系結構 文章標籤： mmu linux 多進程 多線程

目錄

1 對多核處理器以及進程、線程的困惑

雖然不記得在哪個文獻上看到過，但確實記得看過類似的表述：**對多核處理器，同一時間只能運行一個進程裡的多個線程**。一直沒有深究過這句話的對錯，直到看到 linux 的進程、線程模型，才對這句話產生了懷疑。在表述清楚我的困惑之前，先簡單的介紹一下進程、線程以及 linux 中的進程與線程。

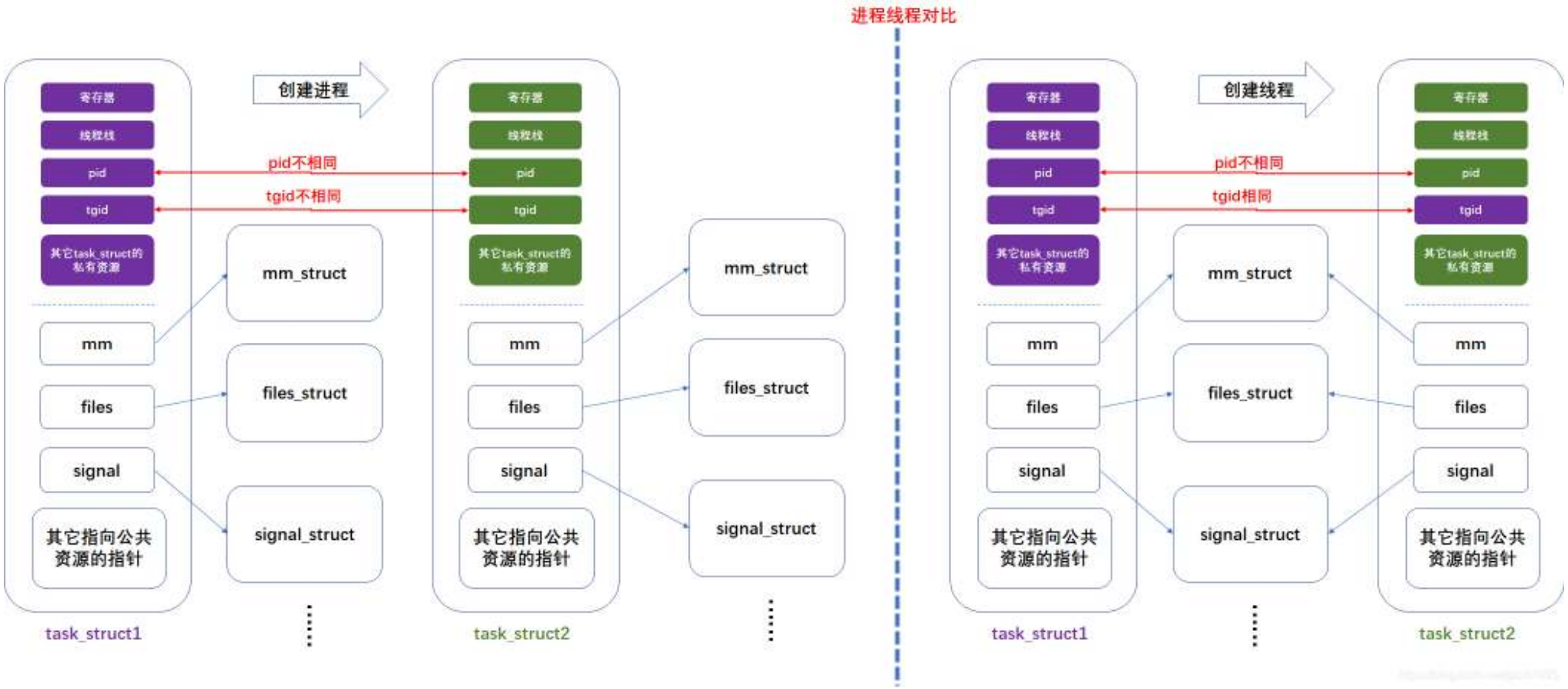
首先，進程的內含通常被認為是運行中的程序及相關資源的總和，而線程被包含在進程之中，是操作系統能夠進行運算調度的最小單位，大部分情況下，它是進程中的實際運作單位[1]。這樣說或許非常抽象，不妨借助圖像理解。對於含有多線程的進程來說，進程與線程的關係如下圖所示：



不難看出，進程持有其所有線程共享的公共資源，而不同線程都保存有自己的寄存器備份以及棧等內容，也就是說處理器的某個核心上各個寄存器的值來自於一個線程，在線程被切換出去時這些寄存器信息都會被保存回線程裡，而被調入的線程所保存的寄存器信息將被載入。 這說明真正在 CPU 上運行的是進程中一個個的線程，所以才將線程稱為進程中的實際運作單位。再打個比方，比如一個家，有房子有家具，家裡生活了一家人，這個家的物理實體（線程）被這一家人共享，而每天真正幹活的是家裡的每個人（線程），房子家具是不會幹活的。好了，進程、線程就介紹到這裡。下面介紹一下 linux 的進程、線程。

各種操作系統教材上往往會對進程、線程做出涇渭分明的定義或說明，也確實有操作系統實現了清晰的進程、線程抽象，但 linux 沒有。linux 沒有為進程（process）和線程（thread）分別做抽象，而是使用一個名為 task_struct 的結構來描述調度的一個單元。對於那些進程擁有的**公共資源**，比如地址空間、打開的文件、信號等，linux 分別使用相應的對象（結構體）來描述它們，例如描述進程地址空間的 **mm_struct**。在 task_struct 中，不會完整的保存描述公共資源的對象，僅維護一個指向這些對象的指針。這樣一來，假如兩個 task_struct 中相應指針指向了同一個描述公共資源的對象實例，那麼就說明

這兩個 task_struct 共享該公共資源，比如共享地址空間，共享打開的文件等。具體共享哪些東西是可以控制的，當我們使用 **clone** 系統調用去創建一個 task_struct 時，可以通過傳參告訴內核新創建的任務與當前任務共享哪些資源。那麼如何用 task_struct 去體現進程和線程呢？不難想到，假如兩個 task_struct 不共享任何公共資源，它們就被視為兩個進程；相反，如果兩個 task_struct 共享所有公共資源，它們就被視為一個進程下的兩個線程。事實上，linux 中，用 fork 創建進程、用 pthread_create 創建線程，其內部都是通過調用 clone，並為 clone 傳遞不共享/共享公資源的參數來實現的，如下圖：


进程线程对比
寄存器
线程栈
pid
tgid
其它task_struct的私有资源
mm
files
signal
其它指向公共资源的指针
创建进程
pid不相同
tgid不相同
mm_struct
files_struct
signal_struct
task_struct1
task_struct2
创建线程
pid不相同
tgid相同

鋪墊結束，下面來說說我的**困惑**所在。如果**對多核處理器，同一時間只能運行一個進程裡的多個線程**這個說法成立的話，對於進程、線程抽象清晰的操作系統，這個場景勉強還能想像：OS 的調度器工作在兩個層次，以進程為單位切換，調度一個進程內的多個線程到各個 core 上執行。而 linux 這樣對進程、線程沒有清晰的抽象，內核面對的是位於各個 core 的調度隊列上的一個個的 task_struct，其調度也是以 task_struct 為單位的，而這些 task_struct 可能共享公共資源（線程），也可能不共享公共資源（進程），難道在調度的時候各個核之間還要同步，確保調度的都是共享公共資源（特別是地址空間）的 task_struct？這場景，怎麼想怎麼彆扭！

於是我開始懷疑**對多核處理器，同一時間只能運行一個進程裡的多個線程**這個說法是錯誤的，下面就要找證據證明。那麼，從什麼方向入手才能查找到有針對性的資料呢？我是這樣考慮的，假如**對多核處理器，同一時間只能運行一個進程裡的多個線程**成立，那麼制約**同一時間運行不同進程的多個線程**的要素是什麼呢？不難想到是 MMU，MMU 可謂是連接虛擬地址和物理地址的橋樑，MMU+頁表=虛擬地址空間，而進程、線程之間的重要區別就是是否共享共享地址空間。**假如，一個多核處理器，所有 core 都共享一個 MMU 的話，那麼同一時間確實只能運行一個進程裡的多個線程**。

因此，最關鍵的問題來了，MMU 是被共享的麼？

2 多核處理器上的 MMU 和 TLB

接著上文，為了弄清楚 MMU 是否是被共享的，我對這個問題做了些搜索，並在 stackoverflow 上發現，有人討論過這個問題：[Do multi-core CPUs share the MMU and page tables?]有興趣的同學可以自己去看看裡面的回答。這裡我給出答案，**MMU 通常不是共享的**（我不確定對所有處理器成立）。不妨看一下酷睿 i7 的存儲系統框圖：

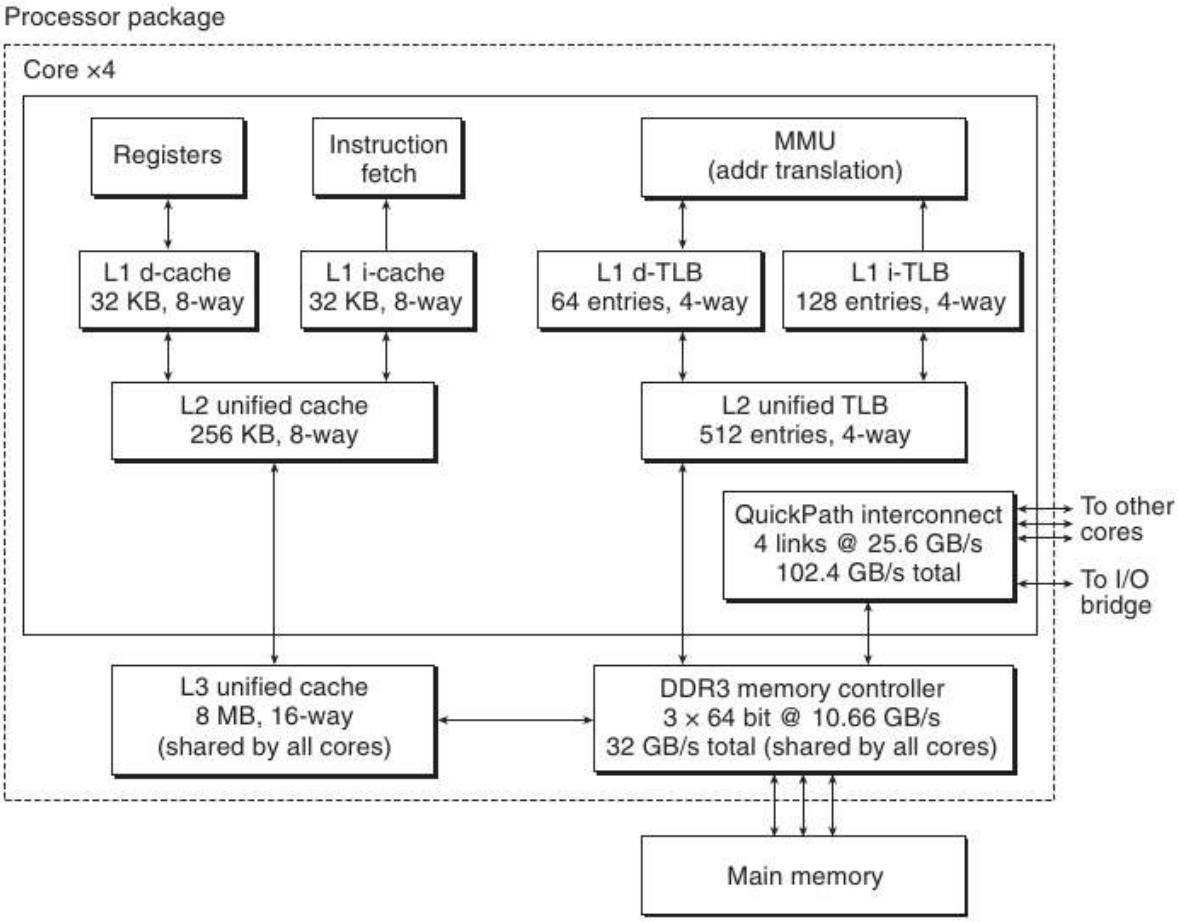


Figure 9.21 **The Core i7 memory system.**

為了進一步佐證多核處理器可以同時運行不同進程的多個線程，我在《Understanding Linux Kernel》中找到這麼一段話：

In a multiprocessor system, each CPU has its own TLB, called the local TLB of the CPU. Contrary to the hardware cache, the corresponding entries of the TLB need not be synchronized, because processes running on the existing CPUs may associate the same linear address with different physical ones.

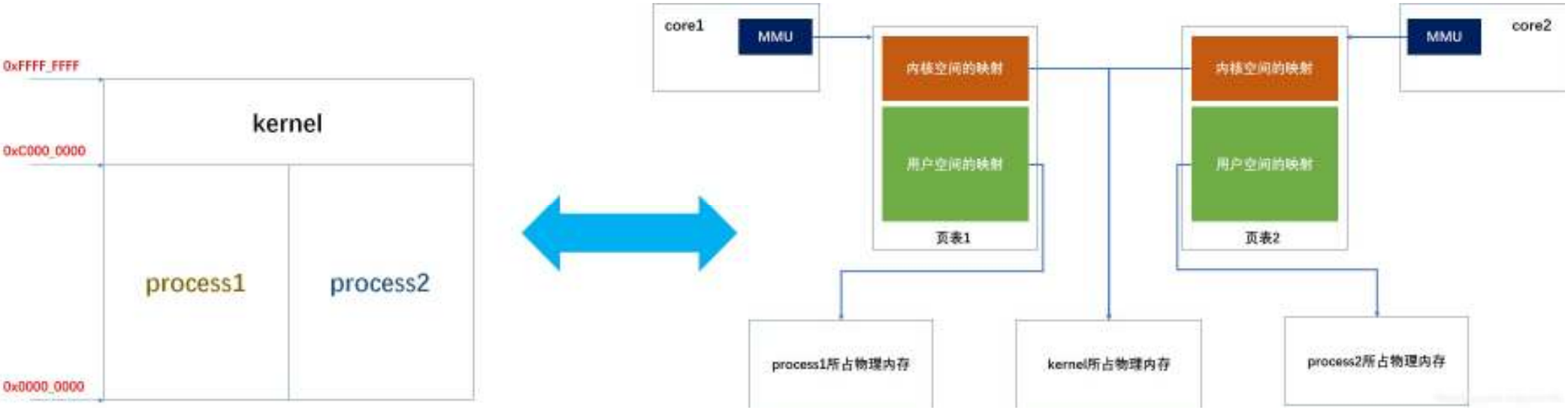
翻譯如下：

在多處理器系統中，每個 CPU 擁有自己的 TLB（譯者補：快表，專門用於頁表的 Cache）稱為本地 TLB。不同於硬件 Cache，（多個）TLB 的表項之間不需要做同步，因為運行在各個 CPU 上的進程可能**將同一個虛擬地址映射到不同的物理地址處（譯者補：說明不共享地址空間）**。

當然，可能會有人指出**多處理器**和**多核**之間是有差異的，但兩者之間也有很多共同之處。多核中，每個核都擁有自己的 MMU，它們也可以像多處理器系統那樣，每個 MMU 都聯繫到

獨立的頁表，這樣同一時間，不同的虛擬地址空間可以並存，因此也就支持**同時運行不同進程的多個線程**。

需要指出的是，雖然不同進程之間通過維護各自的頁表使得虛擬地址空間相互獨立（除了少數情況，如共享內存），但那僅限於用戶部分，而內核部分的地址映射對於所有進程都是相同的。也就是說，頁表中內核部分的映射對於所有進程來說都是相同的。如下圖：


0xFFFF_FFFF
kernel
0xC000_0000
process1
process2
0x0000_0000
core1
MMU
内核空间的映射
用户空间的映射
页表1
内核空间的映射
用户空间的映射
页表2
MMU
core2
process1所占物理内存
kernel所占物理内存
process2所占物理内存

在結束這個問題的討論之前，不妨分析一下 linux 的進程核線程的開銷大小。我將從創建、切換、通信這三個方面做簡要分析：

- **創建**
  在 linux 中，創建進程或線程本質上都是在內核空間建立一個 task_struct 結構，但著不意味著進程、線程擁有同樣的開銷。創建線程只需要和當前的 task_struct 共享資源即可；而創建進程時，不僅要建立一個 task_struct 結構，還需要複製當前 task_struct 的公共資源。雖然 linux 有寫時複製的機制，但這只是延後複製，而非不複製，債是要還的，晚一些也是要還的。因此就創建方面來說，進程的開銷大於線程的開銷。
- **切換**
  linux 切換的都是 task_struct，但被換出的 task_struct 和換入的 task_struct 是否共享公共資源，這對切換的開銷有不小的影響。假如兩者不共享公共資源（進程切換），那麼當前 core 的 **Cache** 和 **TLB** 都會失效，換入的進程在運行初期，Cache 和 TLB 尚未有效建立時，運行會比較慢。而有效建立 Cache 和 TLB 是需要時間的，考慮到調度往往以毫秒為單位，因此這個時間是無法忽略的。假如兩者共享公共資源（線程切換），這就意味著兩者共享虛擬地址空間，也就無需失效 **Cache** 和 **TLB**。當然，考慮到兩個線程運行的指令、訪問的數據可能會有部分位於不同的地址處，因此換入的線程在運行初期，可能會有較多的 Cache 和 TLB **未命中**，但這比起進程切換需要失效 Cache 和 TLB 要好得多。所以，就切換方面來說，進程的開銷大於線程的開銷。
- **通信**
  由於進程不共用地址空間，因此進程間實現通信要麻煩一些，代價也大一些；而線程間共享地址空間，最簡單的，使用全局變量就可以實現通信（儘管有些時候這種同步方式不一定恰當）。因此，進程間通信開銷大於線程間通信開銷。

綜上，進程的開銷是大於線程的開銷的。但值得指出的是，並不能因為線程開銷小而把它當作萬金油。進程也有著地址空間隔離，相互影響小的優點。總的來說，進程、線程各有其適用的場景，應當恰當的適用它們。

參考文獻

[1] 維基百科—線程

[2] Do multi-core CPUs share the MMU and page tables?

[3] Understanding Linux Kernel

---

i