当 CPU 空闲时它都在做什么?

CPP开发者 2018-02-08 05:51

> 编译：Linux 中国 / qhwdw，英文：Gustavo Duarte
> https://linux.cn/article-9303-1.html

在上篇文章中我说了操作系统行为的基本原理是，*在任何一个给定的时刻*，在一个 CPU 上**有且只有一个任务是活动的**。但是，如果 CPU 无事可做的时候，又会是什么样的呢?

事实证明，这种情况是非常普遍的，对于绝大多数的个人电脑来说，这确实是一种常态：大量的睡眠进程，它们都在等待某种情况下被唤醒，差不多在 100% 的 CPU 时间中，都处于虚构的“空闲任务”中。事实上，如果一个普通用户的 CPU 处于持续的繁忙中，它可能意味着有一个错误、bug、或者运行了恶意软件。

因为我们不能违反我们的原理，*一些任务需要在一个 CPU 上激活*。首先是因为，这是一个良好的设计：持续很长时间去遍历内核，检查是否*有*一个活动任务，这种特殊情况是不明智的做法。最好的设计是*没有任何例外的情况*。无论何时，你写一个 `if` 语句，Nyan Cat 就会喵喵喵。其次，我们需要使用空闲的 CPU 去做*一些事情*，让它们充满活力，你懂得，就是创建天网计划呗。

因此，保持这种设计的连续性，并领先于那些邪恶计划一步，操作系统开发者创建了一个**空闲任务**，当没有其它任务可做时就调度它去运行。我们可以在 Linux 的 引导过程 中看到，这个空闲任务就是进程 0，它是由计算机打开电源时运行的第一个指令直接派生出来的。它在 rest_init 中初始化，在 init_idle_bootup_task 中初始化空闲调度类scheduling class。

简而言之，Linux 支持像实时进程、普通用户进程等等的不同调度类。当选择一个进程变成活动任务时，这些类按优先级进行查询。通过这种方式，核反应堆的控制代码总是优先于 web 浏览器运行。尽管在通常情况下，这些类返回 NULL，意味着它们没有合适的任务需要去运行 —— 它们总是处于睡眠状态。但是空闲调度类，它是持续运行的，从不会失败：它总是返回空闲任务。

好吧，我们来看一下这个空闲任务*到底做了些什么*。下面是 cpu_idle_loop，感谢开源能让我们看到它的代码：

```
while (1) {
```

```
    while(!need_resched()) {
        cpuidle_idle_call();
    }
    /*
     [Note: Switch to a different task. We will return to this loop when the idle task is again selected to run.]
     */
    schedule_preempt_disabled();
}
```

cpu_idle_loop

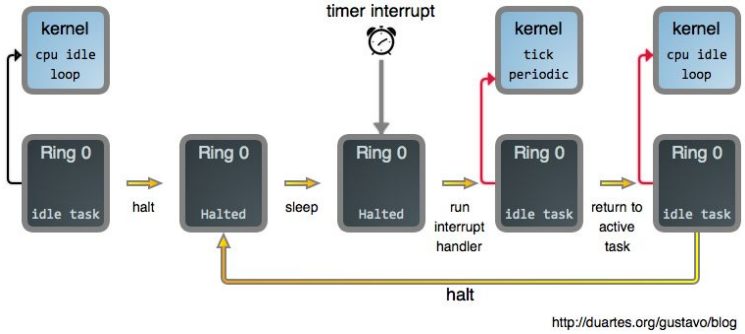
我省略了很多的细节，稍后我们将去了解任务切换，但是，如果你阅读了这些源代码，你就会找到它的要点：由于这里不需要重新调度（即改变活动任务），它一直处于空闲状态。以所经历的时间来计算，这个循环和其它操作系统中它的“堂兄弟们”相比，在计算的历史上它是运行的最多的代码片段。对于 Intel 处理器来说，处于空闲状态意味着运行着一个 halt 指令：

```
static inline void native_halt(void)
    {
    asm volatile("hlt": : :"memory");
    }
```

native_halt

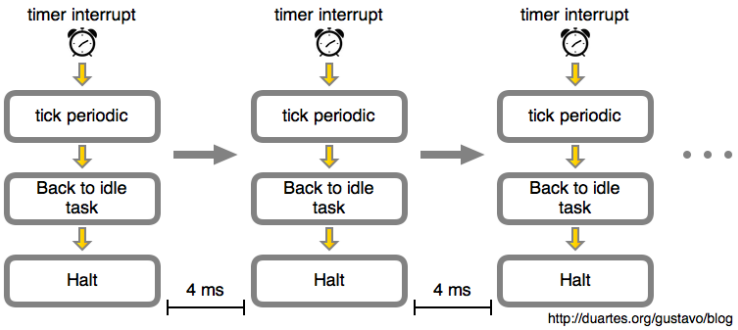
`hlt` 指令停止处理器中的代码执行，并将它置于 `halt` 的状态。奇怪的是，全世界各地数以百万计的 Intel 类的 CPU 们花费大量的时间让它们处于 `halt` 的状态，甚至它们在通电的时候也是如此。这并不是高效、节能的做法，这促使芯片制造商们去开发处理器的深度睡眠状态，以带来着更少的功耗和更长休眠时间。内核的 cpuidle 子系统 是这些节能模式能够产生好处的原因。

现在，一旦我们告诉 CPU 去 `halt`（睡眠）之后，我们需要以某种方式让它醒来。如果你读过 上篇文章《你的操作系统什么时候运行？》，你可能会猜到*中断*会参与其中，而事实确实如此。中断促使 CPU 离开 `halt` 状态返回到激活状态。因此，将这些拼到一起，下图是当你阅读一个完全呈现的 web 网页时，你的系统主要做的事情：



除定时器中断外的其它中断也会使处理器再次发生变化。如果你再次点击一个 web 页面就会产生这种变化，例如：你的鼠标发出一个中断，它的驱动会处理它，并且因为它产生了一个新的输入，突然进程就可运行了。在那个时刻， `need_resched()` 返回 `true`，然后空闲任务因你的浏览器而被踢出而终止运行。

如果我们呆呆地看着这篇文章，而不做任何事情。那么随着时间的推移，这个空闲循环就像下图一样：



在这个示例中，由内核计划的定时器中断会每 4 毫秒发生一次。这就是滴答tick周期。也就是说每秒钟将有 250 个滴答，因此，这个*滴答速率（频率）*是 250 Hz。这是运行在 Intel 处理器上的 Linux 的典型值，而其它操作系统喜欢使用 100 Hz。这是由你构建内核时在 `CONFIG_HZ` 选项中定义的。

对于一个*空闲 CPU* 来说，它看起来似乎是个无意义的工作。如果外部世界没有新的输入，在你的笔记本电脑的电池耗尽之前，CPU 将始终处于这种每秒钟被唤醒 250 次的地狱般折磨的小憩中。如果它运行在一个虚拟机中，那我们正在消耗着宿主机 CPU 的性能和宝贵的时钟周期。

在这里的解决方案是 动态滴答，当 CPU 处于空闲状态时，定时器中断被 暂停或重计划，直到内核*知道*将有事情要做时（例如，一个进程的定时器可能要在 5 秒内过期，因此，我们不能再继续睡眠了），定时器中断才会重新发出。这也被称为*无滴答模式*。

最后，假设在一个系统中你有一个*活动进程*，例如，一个长时间运行的 CPU 密集型任务。那样几乎就和一个空闲系统是相同的：这些示意图仍然是相同的，只是将空闲任务替换为这个进程，并且相应的描述也是准确的。在那种情况下，每 4 毫秒去中断一次任务仍然是无意义的：它只是操作系统的性能抖动，甚至会使你的工作变得更慢而已。Linux 也可以在这种单一进程的场景中停止这种固定速率的滴答，这被称为 自适应滴答 模式。最终，这种固定速率的滴答可能会 完全消失。

对于阅读一篇文章来说，CPU 基本是无事可做的。内核的这种空闲行为是操作系统难题的一个重要部分，并且它与我们看到的其它情况非常相似，因此，这将帮助我们理解一个运行中的内核。