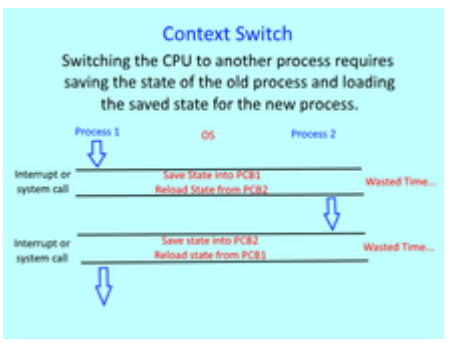
WIKIPEDIA

上下文切换

在计算中，**上下文切换**是存储进程或线程状态的过程，因此可以在稍后恢复并从同一点继续执行。这允许多个进程共享单个CPU，并且是多任务操作系统的基本特征。

短语“上下文切换”的确切含义在使用上有很大不同。在多任务环境中，它指的是为一个任务存储系统状态的过程，以便可以暂停该任务并恢复其他任务。上下文切换也可能因中断而发生，例如当任务需要访问磁盘存储时，为其他任务释放CPU时间。某些操作系统还需要上下文切换以在用户模式和内核模式任务之间移动。上下文切换的过程可能会对系统性能产生负面影响，但这种影响的大小取决于正在执行的切换的性质。



上下文切换的过程

内容

成本

上下文切换通常是计算密集型的，操作系统的大部分设计都是为了优化上下文切换的使用。从一个进程切换到另一个进程需要一定的时间进行管理 - 保存和加载寄存器和内存映射，更新各种表和列表等。上下文切换实际涉及的内容在这些意义之间以及处理器和操作之间有所不同系统。例如，在Linux内核中，上下文切换涉及切换寄存器，堆栈指针和程序计数器，但与地址空间切换无关，但在进程切换中也会发生地址空间切换。[1] [2]此外，类似的上下文切换发生在用户线程之间，特别是绿色线程，并且通常非常轻量级，节省并恢复最小上下文。在极端情况下，例如在Go中的 goroutine之间切换时，上下文切换等同于协程良率，这比子程序调用仅稍微昂贵一些。

何时切换？

上下文切换有三种潜在的触发器：

多任务

最常见的情况是，在某些调度方案中，必须将一个进程从CPU中切换出来，以便可以运行另一个进程。这种上下文切换可以由进程自行启动，例如通过等待I / O或同步操作来完成。在先发制人的多任务系统上，调度器也可以切换仍然可运行的进程。为了防止其他进程中断CPU时间，抢占式调度程序通常会配置一个定时器中断，以在进程超过其时间片时触发。此中断确保调度程序将获得控制权来执行上下文切换。

中断处理

现代架构是中断驱动的。这意味着如果CPU从磁盘请求数据，例如，它不需要等待，直到读取结束; 它可以发出请求并继续执行其他一些操作。读取结束后，可以*中断* CPU 并显示读取结果。对于中断，安装了一个称为*中断处理程序*的程序，它是处理磁盘中断的中断处理程序。

发生中断时，硬件会自动切换部分上下文（至少足以让处理程序返回到中断代码）。处理程序可以保存更多的上下文，具体取决于特定硬件和软件设计的细节。为了最小化处理中断所花费的时间，通常只改变上下文的最小部分。该内核不产卵或安排特殊的工艺处理中断，而是处理程序在中断处理开始建立的（往往是局部）上下文中执行。一旦中断服务完成，在中断发生之前有效的上下文被恢复，以便被中断的进程可以在其正常状态下恢复执行。

用户和内核模式切换

当在操作系统中需要用户模式和内核模式之间的转换时，上下文切换不是必需的; 模式转换本身*不是*一个上下文切换。但是，根据操作系统的不同，上下文切换也可能在此时发生。

步骤

在交换机中，当前正在执行的进程的状态必须以某种方式保存，以便在重新调度时可以恢复该状态。

进程状态包括进程可能使用的所有寄存器，尤其是程序计数器，以及可能需要的任何其他特定于操作系统的数据。这通常存储在称为过程控制块（PCB）或开关框架的数据结构中。

PCB可能存储在内核内存的每个进程堆栈中（与用户模式调用堆栈相反），或者可能存在某些特定的操作系统定义的数据结构以用于此信息。PCB 的句柄被添加到准备运行的进程队列中，通常称为就绪队列。

由于操作系统已经有效地暂停了一个进程的执行，因此它可以通过从就绪队列中选择进程并恢复其PCB来切换上下文。在这样做时，来自PCB的程序计数器被加载，并且因此执行可以在选择的过程中继续。进程和线程优先级可以影响从就绪队列中选择哪个进程（即，它可能是优先级队列）。

性能

由于运行任务调度器，TLB刷新以及间接由于在多个任务之间共享CPU缓存而导致上下文切换本身的性能成本。[3]在单个进程的线程之间的切换可以比两个单独的过程之间更快，因为线程共享相同的虚拟存储器的地图，所以TLB冲洗是没有必要的。[4]

硬件与软件

上下文切换可以主要通过软件或硬件来执行。一些处理器，如Intel 80386及其后继者[5]，通过使用指定为任务状态段或TSS 的特殊数据段，为上下文切换提供硬件支持。可以使用针对全局描述符表中 TSS描述符的CALL或JMP指令显式触发任务切换。如果中断描述符表中存在任务门，则会触发中断或异常时隐式发生。当发生任务切换时，CPU可以自动从TSS加载新状态。

与硬件中执行的其他任务一样，人们会认为这很快; 然而，主流的操作系统，包括的Windows和Linux的，[6]不使用此功能。这主要是由于两个原因：

- 硬件上下文切换不会保存所有寄存器（只有通用寄存器，不是浮点寄存器 - 尽管CR位控制寄存器中的TS位会自动打开，导致执行浮点指令时出现故障并使操作系统有机会根据需要保存并恢复浮点状态）。
- 相关的性能问题，例如软件上下文切换可以是选择性的，只存储那些需要存储的寄存器，而硬件上下文切换存储几乎所有的寄存器，而不管它们是否需要。

参考

1. *IA-64 Linux内核：设计和实现*，4.7切换地址空间 (http://www.informit.com/articles/article.aspx?p=29961&seqNum=7)
2. *操作系统*，5.6上下文切换，p。118 (https://books.google.com/books?id=orZ0CLxEMXEC&pg=PA118)
3. 传播李; 陈鼎; Kai Shen。“量化上下文切换的成本” (https://www.usenix.org/legacy/events/expcs07/papers/2-li.pdf)（PDF）。
4. Ulrich Drepper（2014年10月9日）。“内存部分3：虚拟内存” (https://lwn.net/Articles/253361/)。LWN.net。
5. “上下文切换定义” (http://www.linfo.org/context_switch.html)。Linfo.org。检索2013-09-08。
6. Bovet，丹尼尔皮埃尔; Cesati，Marco（2006）。*了解Linux内核，第三版* (https://books.google.com/books?id=h0lltXyJ8aIC&lpg=PA104&dq=Linux%20hardware%20TSS&pg=PA104#v=onepage&q=Linux%20hardware%20TSS)。O'Reilly媒体。页。104. ISBN 978-0-596-00565-8。检索2009-11-23。

外部链接

- (http://wiki.osdev.org/Context_Switching)在OSDev.org 上下文切换 (http://wiki.osdev.org/Context_Switching)
- (http://www.linfo.org/context_switch.html)Linux信息项目的上下文切换定义 (http://www.linfo.org/context_switch.html)（LINFO）
- (http://msdn.microsoft.com/en-us/library/ms682105(VS.85).aspx)来自Microsoft Developer Network（MSDN）的上下文切换 (http://msdn.microsoft.com/en-us/library/ms682105(VS.85).aspx)
- 架构与设计-中断处理 (http://www.freebsd.org/doc/en/books/arch-handbook/smp-design.html)在FreeBSD.org