第二章 进程与线程

跟踪CPU如何在程序间来回切换是困难的，所以设计人员引入了一种进程模型来解决这个问题，

进程实际上就是程序的抽象，之前我们说过，为用户提供抽象的环境是操作系统的基本功能。

进程被保存在内存中。

进程和程序间的区别是很微妙的，但非常重要。用一个比喻可以更容易理解这一点。想象一位有一手好厨艺的计算机科学家正在为他的女儿烘制生日蛋糕。他有做生日蛋糕的食谱，厨房里有所需的原料：面粉、鸡蛋、糖、香草汁等。在这个比喻中，做蛋糕的食谱就是程序（即用适当形式描述的算法），计算机科学家就是处理器(CPU)，而做蛋糕的各种原料就是输入数据。进程就是厨师阅读食谱、取来各种原料以及烘制蛋糕等一系列动作的总和。 现在假设计算机科学家的儿子哭着跑了进来，说他的头被一只蜜蜂盐了。计算机科学家就记录下他照着食谱做到哪儿了（保存进程的当前状态），然后拿出一本急救手册，按照其中的指示处理垫伤。这里，处理机从一个进程（做蛋糕）切换到另一个高优先级的进程（实施医疗救治），每个进程拥有各自的程序（食谱和急救手册）。当蜜蜂垫伤处理完之后，这位计算机科学家又回来做蛋糕，从他离开时的那一步继续做下去。这里的关键思想是：一个进程是某种类型的一个活动，它有程序、输入、输出以及状态。单个处理器可以被若干进程共享，它使用某种调度算法决定何时停止一个进程的工作，并转而为另一个进程提供服务。

进程的创建

4种主要事件会导致进程的创建：

1. 系统初始化。
2. 正在运行的程序执行了创建进程的系统调用。
3. 用户请求创建一个新进程。
4. 一个批处理作业的初始化。

进程的终止

通常由下列条件引起：

1. 正常退出（自愿的）。
2. 出错退出（自愿的）。
3. 严重错误（非自愿）。
4. 被其他进程杀死（非自s愿）。

进程的层次结构

在支持POSIX的类UNIX系统中进程是拥有明确的层次结构的，如最初始的操作系统这个进程就是由BIOS启动Init这个进程，来启动操作系统，之后的整个操作系统的所有进程都是由整个进程创建的，都是他的子进程，这也是因为，在UNIX系列的操作系统中，只有一种方法可以创建新的进程就是通过fork，而执行该命令，操作系统实际做的是，将父进程进行复制，复制出来的进程作为子进程，这也就使UNIX系统只有一个根进程。 而在Windows中则不一样，因为它在创建子进程的同时，父进程会得到一个特殊的令牌（称为句柄），这个句柄可以用来控制子进程，但是这个句柄可以传送给其他进程，这样做就不存在进程的层次了。

进程的状态

进程主要有**运行**、**就绪**和**阻塞**三种。 它们之间有四种可能的转换关系，分别是运行到阻塞，阻塞到就绪以及就绪和运行间的相互转换。 也就是说所有的运行都是由就绪转换的 当进程在等待输入时，运行的进程会被转到阻塞状态，直到该进程得到输入则会自动转到就绪，若当时有空闲的CPU那么它将转到运行状态。

运行状态和就绪状态在本质上是没有区别的，只是就绪的进程当前没有可以供其使用的CPU。

进程的实现

进程表

进程表是一个结构数组，每个进程占用一个进程表项，也称为，**进程控制块**。该表包含了进程状态的重要信息，包括程序计数器、堆栈指针 、内存分配状况、所打开文件的状态、账号和调度信息，以及其他在进程由运行态转换到就绪态或阻塞态时必须保存的信息，从而保证该进程随后能再次启动。

实时要求

一些特定事件一定要在所制定的若干毫秒内发生。

逻辑程序计数器

线程

多线程之间会共享同一块地址空间和所有可用数据的能力，这是进程所不具备的。注意之前说过，尤其在UNIX系统中每个进程都有自己独立的内存空间。

线程实际上就是轻量化的进程，创建一个线程要比创建一个进程快10-100倍

~~但是线程和进程实质上的区别还是没搞明白，以后明白了再来写吧~~

好像明白了点：线程是进程内的进程，为什么要这样套娃呢？这是因为如果一个程序采用多个进程，比较难监控其运行顺序，因为某些进程会被阻塞，从而运行其他的进程，很多时候这会影响程序的正确顺序，而如果在进程内部进行操作，就可以由程序主动管理这些进程，进程里面的进程就是线程。这样一个程序就进一步减少了因为阻塞而导致的资源浪费，可以进一步提升程序的效率。

总的来说：进程是对于操作系统来说的，而线程更多可能是对于程序的，进程增大操作系统的效率，线程增大程序的效率。

多线程解决方案

多线程解决方案是相对于单线程来讲的。 单线程就是顺序执行所有任务，如果遇到堵塞就等待，CPU空转，再来了新的任务，就得排在后面了，很浪费资源。

在多线程中，引入了一个**调度线程(dispatcher thread)**，我目前理解的是它是一个只负责调度的线程，就是根据当前各个线程的情况负责安排工作的线程。在web服务器中它读入来自外部的请求，当它检查完请求后，他会选择一个空闲或者阻塞的工作线程来处理这个请求。此外调度线程还会判断当前请求的网页是否在cache中，如果在cache中则直接发送给用户，如果不在cache中，则向磁盘请求读取，当线程阻塞在磁盘读取时，调度线程就会挑选另一个线程运行或者把另一个当前处于就绪状态的工作线程投入运行。

状态机解决就方案

状态机解决方案是平时听到最少的一种了，它需要非阻塞的系统调用来支持才可以完成正常的设计，如上面的例子中就就需要非阻塞的磁盘读取系统调用，当请求到达时，先判断请求的网页是否在cache内，如果在就直接返回，如果不在就调用非阻塞的磁盘读取系统调用，然后将这个请求的状态记录到一个表格中，再去接收或处理其他请求，当磁盘读取完，再从表格中载入之前记录的状态，将网页返回给用户，再重复这个操作。这种方式就称为**有限状态机(finite-state machine)**

线程的堆栈

每个线程都有自己的堆栈，为什么呢 *我也没看明白，应该是因为不太理解堆栈和程序运行的关系导致的*

多线程的实现方法

一般来说多线程的实现方法主要分为两种，分别是**在用户空间中实现线程**和**在内核中实现线程**，再加上这两种方法的混合使用，一共是三种，前两种方法都有各自的优劣。

方法	优势	劣势
用户空间	1.可以在不支持的操作系统上实现 2.允许每个进程有自己定制的调度算法 3.进程的创建、删除等操作占有资源少执行快	1.实现阻塞系统调用较复杂 2.在进程中是没有时钟的，如果线程不肯主动放弃CPU的使用权，那么将没有办法使其强制放弃 3.缺页问题
内核	1.不需要考虑阻塞系统调用的问题	1.所有线程使用统一的线程表，会有上限要求

在用户空间中实现线程 在用户空间中实现线程 实际上 **在内核中实现线程**

thread_yield允许线程主动让出CPU供其他线程使用因为线程不能像进程一样用中断使其强制让出CPU。

守护进程（daemon）停留在后台处理诸如电子邮件、Web页面、新闻、打印之类活动的进程。

我的想法

跟踪CPU在各个进程之间切换的操作是困难的，不妨让进程像子弹一样排入弹夹，CPU就是子弹的击发器，这样CPU相当于待在原地不动，让进程轮流从CPU前通过，CPU只负责处理即可，再由另外一个元件来确定当前

的进程是否需要进行处理，如果需要处理就放入弹夹，如果不需要处理就放在外面等待，直到需要进行处理时再放入弹夹。

逻辑程序计数器 应该就是进程中计算当前运算到那一步的寄存器。