进程

* 一个正在执行的程序
* 计算机中正在运行的程序的一个实例
* 可分配给处理器并由处理器执行的一个实体

计算机系统的发展有三条主线：

* 多道程序批处理操作

为了让处理器和I/O设备(包括存储设备)同时保持忙状态，以实现最大的效率。其关键机制是：在响应表示I/O事务结束的信号时，操作系统将对内存中驻留的不同程序进行处理器切换

* 分时事务系统

其主要设计目标是及时响应单个用户的要求，但由于成本原因，又要可以同时支持多个用户。由于用户反应时间相对较慢，因此这两个目标可以同时实现

* 实时事务系统

有时，会出现很多用户都在对数据库进行查询或修改的情况，事务处理系统和分时系统的主要差别在于，前者局限于一个或几个应用，而后者的用户可以开始程序、执行作业，以及使用各种各样的应用程序

也可把进程视为由一组元素组成的实体，进程的两个基本元素是**程序代码**（program code，可能被执行相同程序的其他进程共享）和与代码相关联的**数据集**（set of data）。假设处理器开始执行这个程序代码，并且我们把这个执行实体称为进程。进程执行的任意时刻，都可由如下元素来表征:

* **标识符:** 与进程相关的唯一标识符，用来区分其他进程(PID, handle)
* **状态:** 若进程正在执行，则进程处于运行态。
* **优先级:** 相对于其他进程的优先顺序
* **程序计数器:** 程序中即将执行的下一条指令的地址
* **内存指针:** 包括程序代码和进程相关数据的指针，以及与其他进程共享内存块的指针
* **上下文数据:** 进程执行时处理器的寄存器中的数据
* **I/O状态信息:** 包括显式I/O请求、分配给进程的I/O设备（如磁带驱动器）和被进程使用的文件列表。
* **记账信息:** 包括处理器时间总和、使用的时钟数总和、时间限制、记账号等。

这些信息都存放在一个称为**进程控制块**(process control block)的数据结构中，由操作系统创建和管理。

进程状态

操作系统会为待执行程序创建进程或任务。处理器以某种顺序执行指令序列中的指令，这种顺序由程序计数器寄存器中不断变化的值给出，因为程序计数器可能会指向不同进程的不同代码部分(存在一个小分派器(调度器)使处理器切换进程)；程序的执行则涉及其内部的一系列指令。

列出为进程执行的指令序列，可描述单个进程的行为，这样的序列称为**进程轨迹**(process trace)。给出各个进程轨迹的交替方式，就可描述处理器的行为。

进程的创建，切换与终止

**进程的创建** 将一个新进程添加到正被管理的进程集时，操作系统需要建立用于管理该进程的数据结构，并在内存中给它分配地址空间。

步骤：

1. **为新进程分配一个唯一的进程标识符**。此时，主进程表会添加一个新表项，每个进程一个表项。
2. **为进程分配空间**。这包括进程映像中的所有元素。因此，操作系统必须知道私有用户地址空间(程序和数据)和用户栈需要多少空间
3. **初始化进程控制块**。进程标识部分包括进程ID和其他相关ID
4. **设置正确的链接**。例如，若操作系统将每个调度队列都维护为一个链表，则新进程必须放在就绪或就绪/挂起链表中。
5. **创建或扩充其他数据结构**。例如，操作系统可因编制账单和/或评估性能，为每个进程维护一个记账文件

触发进程创建的事件通常有4个，如下表所示:

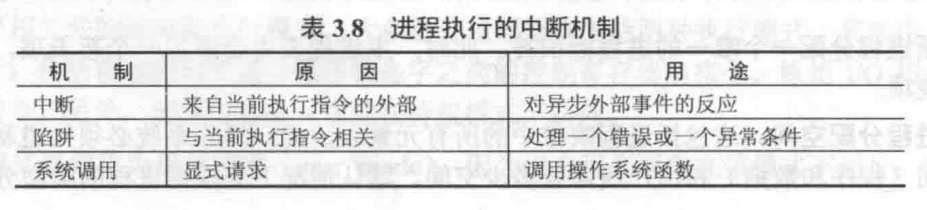
|  |  |
| --- | --- |
| **事 件** | **说 明** |
| 新的批处理作业 | 磁带或磁盘中的批处理作业控制流通常会提供给操作系统。当操作系统准备接收新工作时，将读取下一个作业控制命令 |
| 交互登录 | 终端用户登录到系统 |
| 为提供服务而由操作系统创建 | 操作系统可以创建一个进程，代表用户程序执行一个功能，使用户无须等待(如控制打印的进程) |
| 由现有进程派生 | 基于模块化的考虑或开发并行性，用户程序可以指示创建多个进程 |

当操作系统为另一个进程的显式请求创建一个进程时，这个动作就称为**进程派生**(process spawning)。

当一个进程派生另一个进程时，前一个称为**父进程**(parent process)，被派生的进程称为**子进程**(child process)。典型情况下，相关进程需要相互之间的通信和合作。

**进程切换** 进程的切换也存在一系列的问题，何时切换，什么事件触发切换，如何实现切换

**何时切换进程** 进程切换可在操作系统从当前正运行进程中获取控制权的任何时刻发生，下表给出了可能把控制权交给操作系统的事件



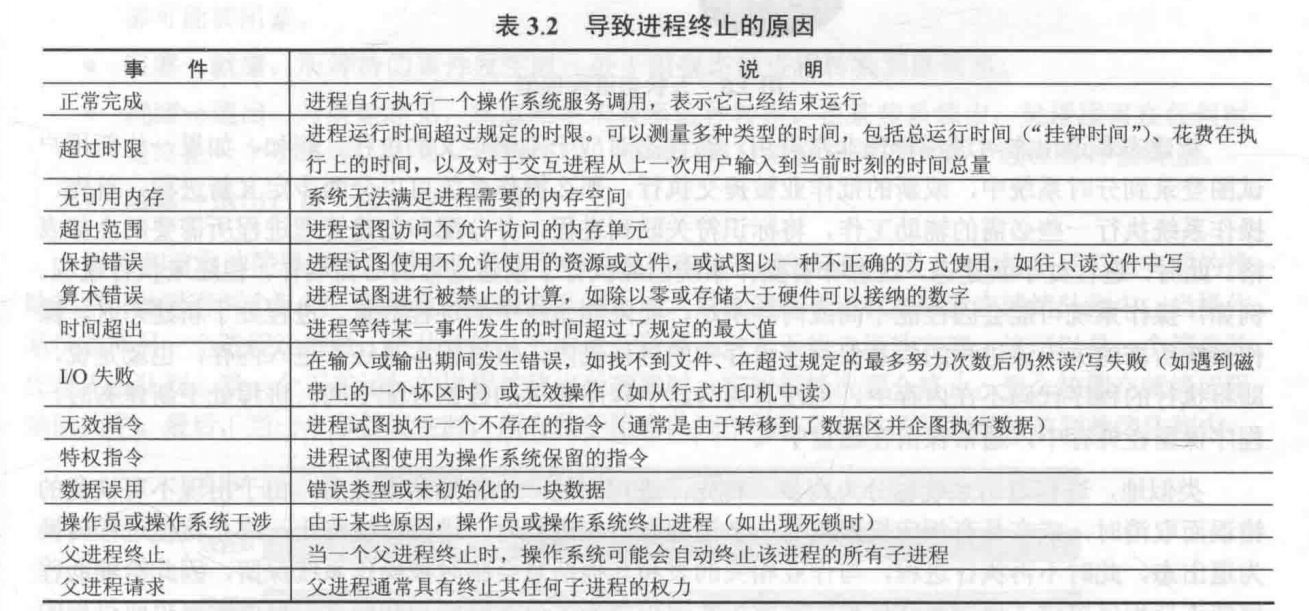
系统中断分为 中断(interrupt)和陷阱(trap)，对于中断，控制权首先转给中断处理器，中断处理器完成基本辅助工作后，再将控制权给与已发生的特定中断相关的操作系统例程(时钟中断，I/O中断，内存缺页)，对于陷阱，操作系统则确定错误或异常条件是否致命，致命时会将程序置为退出态，不致命时，操作系统可能会尝试恢复程序，可能会切换进程，或继续当前运行的进程。

最后，操作系统可被来自正执行程序的**系统调用**(supervisor call)激活。例如，进程执行了一个I/O请求，这时该调用会转移到作为操作系统代码一部分的一个例程。使用系统调用会将用户进程置为阻塞态。

完整的进程切换步骤如下：

1. 保存处理器上下文，包括程序计数器和其他寄存器
2. 更新当前处于运行态进程的进程控制块，包括把进程的状态改变为另一状态。还需更新其他相关的字段，包括退出运行态的原因和记账信息。
3. 把该进程的进程控制块移到相应的队列
4. 选择另一个进程执行
5. 更新所选进程的进程控制块，包括进程的状态改为运行态
6. 更新内存管理数据结构。是否需要更新取决于管理地址转换的方式
7. 载入程序计算器和其他寄存器先前的值，将处理器的上下文恢复为所选进程上次退出运行态时的上下文

**进程终止** 下表概括了进程终止的典型原因。任何一个计算机系统都必须为进程提供表示其完成的方法，批处理作业中应包含一个Halt指令或其他操作系统显式服务调用来终止。在前一种情况下，Halt指令将产生一个中断，警告操作系统一个进程已经完成。对交互式应用程序，用户的行为将指出何时进程完成。(表中的原因不一定会使进程直接终止，可能只是会触发一个异常，若异常未被处理最后进程会终止)



五状态模型

对于可运行的进程，处理器以一种**轮转**(round-robin)方式操作(依次给队列中每个进程一定的执行时间，然后进程返回队列，阻塞情况除外)。但是，即使对一些简单例子，这一实现都是不合适的：存在一些处于非运行态但已就绪等待执行的进程，同时还存在另外一些处于阻塞态等待I/O操作结束的进程。因此，若使用单个队列，分派器不能只考虑队列中最老的进程，而应扫描这个列表，查找那些未被阻塞且在队列时间最长的进程。

解决该问题的一种较好方法是，将非运行态分成两个状态：就绪态(ready)和阻塞态(blocked)。

* **运行态：**进程正在执行
* **就绪态：**进程做好了准备，只要有机会就开始执行
* **阻塞/等待态：**进程在某些事件发生前不能执行，如I/O操作完成
* **新建态：**刚刚创建的进程，操作系统还未把它加入可执行进程组，通常为进程控制块已经创建但还未加载到内存的新进程
* **退出态：**操作系统从可执行进程组中释放出的进程，要么它自身已停止，要么它因某种原因被取消

被挂起的进程

特点：

1. 该进程不能立即执行
2. 该进程可能在也可能不在等待一个事件。若在等待一个事件，那么阻塞条件不依赖于挂起条件，阻塞事件的发生不会使进程立即执行。
3. 为阻止该进程执行，可通过代理使其置于挂起态，代理可以是进程本身，也可以是父进程或操作系统。
4. 除非代理显式地命令系统进行状态转换，否则该进程无法从这一状态转移。

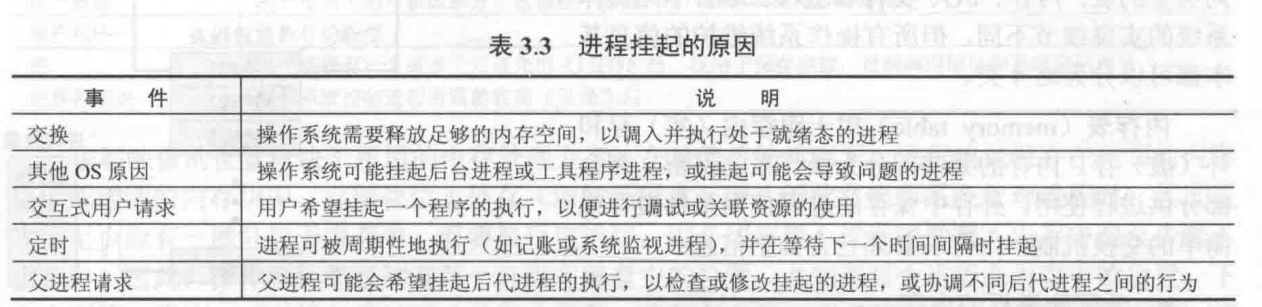
因为I/O活动远慢于计算速度，即使是多道程序设计，处理器多数时间仍可能处于空闲状态，可以通过交换操作，把内存中某个进程的一部分或全部移到磁盘中。当内存中不存在就绪态的进程时，操作系统就把被阻塞的进程换出到磁盘中的挂起队列(suspend queue)，即临时从内存中“踢出”进程队列。操作系统此后要么从挂起队列中取出另一个进程，要么接受一个新进程请求，将其放入内存运行。

要使用交换，在进程行为模型中必须增加一个新的状态：挂起态。当内存中所有进程都处于阻塞态时，操作系统可把其中一个设为挂起态，并将它移到磁盘，此时内存所释放的空间就可被调入的另一个进程使用。

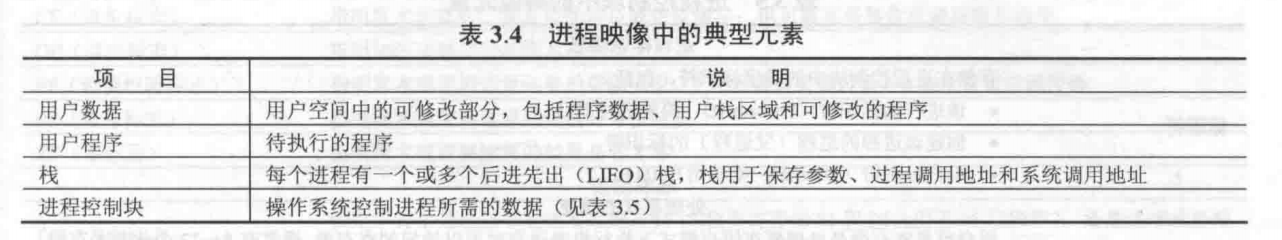
操作系统执行换出操作后，将进程取到内存中的方式有两种：接纳一个新创建的进程，或调入一个之前挂起的进程，显然操作系统倾向于调入一个此前挂起的进程，并为它提供服务，而非增加系统的总负载数。

但这一推理也带来了一个难题，即所有已被挂起的进程都处于阻塞态。显然，这时把被阻塞的进程取回内存没有任何意义，因为它仍然未做好执行的准备。但是，由于每个挂起的进程最初都阻塞在某个特定的事件上，因此发行该事件时，进程不再阻塞在某个特定的事件上，因此发行该事件时，进程将不再阻塞而可以继续执行。

**挂起的其他用途** 到目前为止，挂起进程等价于不在内存中的进程。不在内存中的进程，不论它是否在等待一个事件，都不能立即执行。



进程映像为程序、数据、栈和属性的集合，属性集为进程控制块



进程控制块中的信息分为三类：

* 进程标识信息
* 进程状态信息
* 进程控制信息

**进程控制块的作用** 进程控制块(windows系统为EPROCESS结构)是操作系统中最重要的数据结构。每个进程控制块都包含操作系统所需进程的所有信息。实际上，操作系统的每个模块，包括那些涉及调度、资源分配、中断处理、性能监控和分析的模块，都能读取和修改它们，可以说资源控制块集合定义了操作系统的状态。

这就带来了一个重要的设计问题。操作系统中的很多例程需要访问进程控制块中的信息。直接访问这些表并不困难。每个进程都有一个唯一的ID，它可以用作进程控制块的指针表的索引。困难的是保护这些进程控制块，具体表现为两个问题：

* 一个例程（如中断处理程序）中的错误可能会破坏进程控制块，进而破坏系统对受影响进程的管理能力
* 进程控制块结构或语义中的设计变化可能会影响到操作系统中的许多模块

这些问题可要求操作系统的所有例程都通过一个处理程序例程来解决，处理程序例程的任务不仅是保护进程控制块，而且还是读写这些块的唯一仲裁程序

