# 操作系统

## 进程和线程的区别

进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。**但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。**

**1) 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.**

2) 线程的划分尺度小于进程，使得多线程程序的并发性高。

3) 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4) 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。**但是线程不能够独立执行，**必须依存在应用程序中，由应用程序提供多个线程执行控制。

5) 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。**这就是进程和线程的重要区别。**

## 进程的通信方式

1. **管道/匿名管道(pipe)**

管道是半双工的，数据只能向一个方向流动；需要双方通信时，需要建立起两个管道。只能用于父子进程或者兄弟进程之间(具有亲缘关系的进程); 单独构成一种独立的文件系统：管道对于管道两端的进程而言，就是一个文件，但它不是普通的文件，它不属于某种文件系统，而是自立门户，单独构成一种文件系统，并且只存在与内存中。数据的读出和写入：一个进程向管道中写的内容被管道另一端的进程读出。写入的内容每次都添加在管道缓冲区的末尾，并且每次都是从缓冲区的头部读出数据。

管道的实质是一个内核缓冲区，进程以先进先出的方式从缓冲区存取数据，管道一端的进程顺序的将数据写入缓冲区，另一端的进程则顺序的读出数据。

**2. 有名管道(FIFO)**

匿名管道，由于没有名字，只能用于亲缘关系的进程间通信。为了克服这个缺点，提出了有名管道(FIFO)。有名管道不同于匿名管道之处在于它提供了一个路径名与之关联，**以有名管道的文件形式存在于文件系统中**，这样，**即使与有名管道的创建进程不存在亲缘关系的进程，只要可以访问该路径，就能够彼此通过有名管道相互通信.有名管道的名字存在于文件系统中，内容存放在内存中。**

**有名管道和无名管道在读写前需要确定对方的存在，否则将阻塞。**

**3. 信号(Signal)**

信号是Linux系统中用于进程间互相通信或者操作的一种机制，信号可以在任何时候发给某一进程，而无需知道该进程的状态。如果该进程当前并未处于执行状态，则该信号就有内核保存起来，直到该进程回复执行并传递给它为止。如果一个信号被进程设置为阻塞，则该信号的传递被延迟，直到其阻塞被取消是才被传递给进程。

信号是软件层次上对中断机制的一种模拟，是一种异步通信方式，，信号可以在用户空间进程和内核之间直接交互，内核可以利用信号来通知用户空间的进程发生了哪些系统事件。

**信号生命周期和处理流程**

1. 信号被某个进程产生，并设置此信号传递的对象（一般为对应进程的pid），然后传递给操作系统。
2. 操作系统根据接收进程的设置（是否阻塞）而选择性的发送给接收者，如果接收者阻塞该信号（且该信号是可以阻塞的），操作系统将暂时保留该信号，而不传递，直到该进程解除了对此信号的阻塞（如果对应进程已经退出，则丢弃此信号），如果对应进程没有阻塞，操作系统将传递此信号。
3. 目的进程接收到此信号后，将根据当前进程对此信号设置的预处理方式，暂时终止当前代码的执行，保护上下文（主要包括临时寄存器数据，当前程序位置以及当前CPU的状态）、转而执行中断服务程序，执行完成后在回复到中断的位置

**4. 消息(Message)队列**

消息队列是存放在内核中的消息链表，每个消息队列由消息队列标识符表示。与管道（无名管道：只存在于内存中的文件；命名管道：存在于实际的磁盘介质或者文件系统）不同的是消息队列存放在内核中，只有在内核重启(即，操作系统重启)或者显示地删除一个消息队列时，该消息队列才会被真正的删除。另外与管道不同的是，消息队列在某个进程往一个队列写入消息之前，并不需要另外某个进程在该队列上等待消息的到达。

消息队列允许一个或多个进程向它写入与读取消息. 消息队列可以实现消息的随机查询,消息不一定要以先进先出的次序读取,也可以按消息的类型读取.比FIFO更有优势。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺。

**5. 共享内存(share memory)**

使得多个进程可以可以直接读写同一块内存空间，是最快的可用IPC形式。是针对其他通信机制运行效率较低而设计的。为了在多个进程间交换信息，内核专门留出了一块内存区，可以由需要访问的进程将其映射到自己的私有地址空间。进程就可以直接读写这一块内存而不需要进行数据的拷贝，从而大大提高效率。由于多个进程共享一段内存，因此需要依靠某种同步机制（如信号量）来达到进程间的同步及互斥。

**6. 信号量(semaphore)**

信号量是一个计数器，用于多进程对共享数据的访问，信号量的意图在于进程间同步。PV原语。

**7. 套接字(socket)**

套接字是支持TCP/IP的网络通信的基本操作单元，可以看做是不同主机之间的进程进行双向通信的端点，简单的说就是通信的两方的一种约定，用套接字中的相关函数来完成通信过程。

套接字的端口是一个信息缓冲区，用于保留Socket中的输入/输出信息。套接字有几种类型：

**一是流套接字**，流套接字在域中通过TCP/IP连接实现。流套接字提供的是一个有序、可靠、双向字节流的连接，因此发送的数据可以确保不会丢失、重复或乱序到达，而且它还有一定的出错后重新发送的机制。**二是数据报套接字**，它不需要建立连接和维持一个连接，它们在域中通常是通过UDP/IP协议实现的。它对可以发送的数据的长度有限制，数据报作为一个单独的网络消息被传输,它可能会丢失、复制或错乱到达，UDP不是一个可靠的协议，但是它的速度比较高。**三是原始套接字**，原始套接字允许对较低层次的协议直接访问，比如IP、 ICMP协议，它常用于检验新的协议实现，或者访问现有服务中配置的新设备，因为RAW SOCKET可以自如地控制Windows下的多种协议，能够对网络底层的传输机制进行控制，所以可以应用原始套接字来操纵网络层和传输层应用。比如，我们可以通过RAW SOCKET来接收发向本机的ICMP、IGMP协议包，或者接收TCP/IP栈不能够处理的IP包，也可以用来发送一些自定包头或自定协议的IP包。网络监听技术很大程度上依赖于SOCKET\_RAW。

原始套接字可以读写内核没有处理的IP数据包，而流套接字只能读取TCP协议的数据，数据报套接字只能读取UDP协议的数据。因此，如果要访问其他协议发送数据必须使用原始套接字。

## 死锁

在两个或者多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗的讲就是两个或多个进程无限期的阻塞、相互等待的一种状态。

### 死锁产生的四个条件：

1.互斥条件：一个资源一次只能被一个进程使用

2.请求与保持条件：一个进程因请求资源而阻塞时，对已获得资源保持不放

3.不剥夺条件：进程获得的资源，在未完全使用完之前，不能强行剥夺

4.循环等待条件：若干进程之间形成一种头尾相接的环形等待资源关系

### 死锁的预防 ：

〈1〉打破互斥条件。即允许进程同时访问某些资源。但是，有的资源是不允许被同时访问的，像打印机等等，这是由资源本身的属性所决定的。所以，这种办法并无实用价值。

   〈2〉打破不可抢占条件。即允许进程强行从占有者那里夺取某些资源。就是说，当一个进程已占有了某些资源，它又申请新的资源，但不能立即被满足时，它必须释放所占有的全部资源，以后再重新申请。它所释放的资源可以分配给其它进程。这就相当于该进程占有的资源被隐蔽地强占了。这种预防死锁的方法实现起来困难，会降低系统性能。

    〈3〉打破占有且申请条件。可以实行资源预先分配策略。即进程在运行前一次性地向系统申请它所需要的全部资源。如果某个进程所需的全部资源得不到满足，则不分配任何资源，此进程暂不运行。只有当系统能够满足当前进程的全部资源需求时，才一次性地将所申请的资源全部分配给该进程。由于运行的进程已占有了它所需的全部资源，所以不会发生占有资源又申请资源的现象，因此不会发生死锁。但是，这种策略也有如下缺点：

（1）在许多情况下，一个进程在执行之前不可能知道它所需要的全部资源。这是由于进程在执行时是动态的，不可预测的；

（2）资源利用率低。无论所分资源何时用到，一个进程只有在占有所需的全部资源后才能执行。即使有些资源最后才被该进程用到一次，但该进程在生存期间却一直占有它们，造成长期占着不用的状况。这显然是一种极大的资源浪费；

（3）降低了进程的并发性。因为资源有限，又加上存在浪费，能分配到所需全部资源的进程个数就必然少了。

（4）打破循环等待条件，实行资源有序分配策略。采用这种策略，即把资源事先分类编号，按号分配，使进程在申请，占用资源时不会形成环路。所有进程对资源的请求必须严格按资源序号递增的顺序提出。进程占用了小号资源，才能申请大号资源，就不会产生环路，从而预防了死锁。这种策略与前面的策略相比，资源的利用率和系统吞吐量都有很大提高，但是也存在以下缺点：

（1）限制了进程对资源的请求，同时给系统中所有资源合理编号也是件困难事，并增加了系统开销；

（2）为了遵循按编号申请的次序，暂不使用的资源也需要提前申请，从而增加了进程对资源的占用时间。

### 银行家算法

**系统在进行资源分配时，如何使系统不进入不安全状态。**

## ****进程调度****

又称为低级调度，其主要任务是按照某种方法和策略从就绪队列中选取一个进程，将处理机分配给它。进程调度是操作系统中最基本的一种调度，在一般操作系统中都必须配置进程调度。进程调度的频率很高，一般几十毫秒一次。

### 进程调度方式

1. **非抢占式方式**   
   系统一旦把处理机分配给就绪队列中优先级最高的进程后，该进程就一直执行下去，直至完成；或者发生某事件而不得不放弃处理机。通常用于批处理系统。
2. **抢占式方式**   
   当一个进程正在执行时，系统可以基于某种策略剥夺CPU给其他进程。剥夺的原则有：优先权原则、短进程优先原则和时间片原则。一般用在分时系统和实时系统中。

### 调度的基本准则

1. **CPU利用率**。尽可能是CPU处于忙碌状态
2. **系统吞吐量**。表示单位时间CPU完成作业的数量，短作业消耗的处理机时间较短
3. **周转时间**。从作业提交到作业完成经历的时间。
4. **等待时间**。进程处于等处理机状态时间之和，从提交到开始执行，等待时间越长，用户满意度越低。
5. **响应时间**。指从用户提交请求到系统首次产生响应所用的时间。在交互式系统尤其明显，调度策略应尽可能降低响应时间，使响应时间处在用户能接受的范围之内

### 进程调度算法

**1.先来先服务调度算法（FCFS）**

当在作业调度中采用该算法时，每次调度都是从后备作业队列中选择一个或多个最先进入该队列的作业，将它们调入内存，为它们分配资源，创建进程，然后放入就绪队列。

**2.短作业（进程）优先调度算法（SJF、SPF）**

短作业优先（SJF）的调度算法是从后备作业队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行；而短进程（SPF）调度算法则是从就绪队列中选出一个估计运行时间最短的进程，将处理机分配给它，使它立即执行并一直执行到完成，或发生某事件被阻塞放弃处理机。—非抢占式

**3.优先权调度算法**

当该算法用于作业调度时，系统从后备作业队列中选择若干个优先级最高的，且系统能满足资源要求的作业装入内存运行；当该算法用于进程调度时，将把处理机分配给就绪进程队列中优先级最高的进程投入运行。分为非抢占式优先级算法和抢占式优先级算法。

**4.高响应比优先调度算法**

高响应比优先调度算法主要用于作业调度，该算法是对FCFS调度算法和SJF调度算法的一种综合平衡，同时考虑每个作业的等待时间和估计的运行时间。 在每次进行作业调度时，先计算后备作业队列中每个作业的响应比，从中选出响应比最高的作业投入运行。



**5.时间片轮转调度算法**

系统将就绪进程按到达的顺序排成一个队列，按FCFS原则，进程调度程序总是选择就绪队列中的第一个进程执行，且只运行一个时间片。时间用完后，即使此进程并未完成，仍然将处理机分配给下一个就绪的进程，将此进程返回到就绪队列的末尾，等候重新运行。