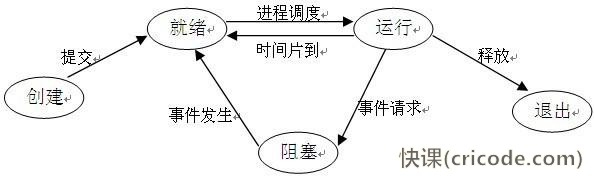
操作系统

1



如上图所示，进程包括三种状态：就绪态、运行态和阻塞态。详细说明如下：  
  
  
注意：创建和退出不是进程的状态。阻塞和就绪的区别：阻塞是等待除CPU以外的资源，而就绪等待的是CPU资源。  
  
  
1）就绪——执行：对就绪状态的进程，当进程调度程序按一种选定的策略从中选中一个就绪进程，为之分配了处理机后，该进程便由就绪状态变为执行状态；  
2）执行——阻塞：正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如进程提出输入/输出请求而变成等待外部设备传输信息的状态，进程申请资源（主存空间或外部设备）得不到满足时变成等待资源状态，进程运行中出现了故障（程序出错或主存储器读写错等）变成等待干预状态等等；   
3）阻塞——就绪：处于阻塞状态的进程，在其等待的事件已经发生，如输入/输出完成，资源得到满足或错误处理完毕时，处于等待状态的进程并不马上转入执行状态，而是先转入就绪状态，然后再由系统进程调度程序在适当的时候将该进程转为执行状态；  
4）执行——就绪：正在执行的进程，因时间片用完而被暂停执行，或在采用抢先式优先级调度算法的系统中,当有更高优先级的进程要运行而被迫让出处理机时，该进程便由执行状态转变为就绪状态。

2

线程是指进程内的一个执行单元,也是进程内的可调度实体.  
与进程的区别:  
(1)地址空间:进程内的一个执行单元;进程至少有一个线程;它们共享进程的地址空间;而进程有自己独立的地址空间;  
(2)资源拥有:进程是资源分配和拥有的单位,同一个进程内的线程共享进程的资源  
(3)线程是处理器调度的基本单位,但进程不是.  
(4)二者均可并发执行.

进程和线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。进程和线程的区别在于：

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.   
线程的划分尺度小于进程，使得多线程程序的并发性高。   
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。   
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。   
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。

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.   
线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.   
一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.

3

# 管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。  
# 有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。  
# 信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。  
# 消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。  
# 信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。  
# 共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。  
# 套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

4

三种同步方式：1、互斥锁（mutex)、2、条件同步（cond)、3、信号量（semphore).

<http://blog.csdn.net/zsf8701/article/details/7844316>

线程的最大特点是资源的共享性，但资源共享中的同步问题是多线程编程的难点。linux下提供了多种方式来处理线程同步，最常用的是互斥锁、条件变量和信号量。

### 一、互斥锁(mutex)

通过锁机制实现线程间的同步。

1. 初始化锁。在Linux下，线程的互斥量数据类型是pthread\_mutex\_t。在使用前,要对它进行初始化。  
   静态分配：pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;  
   动态分配：int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutex\_attr\_t \*mutexattr);
2. 加锁。对共享资源的访问，要对互斥量进行加锁，如果互斥量已经上了锁，调用线程会阻塞，直到互斥量被解锁。  
   int pthread\_mutex\_lock(pthread\_mutex \*mutex);  
   int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);
3. 解锁。在完成了对共享资源的访问后，要对互斥量进行解锁。  
   int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);
4. 销毁锁。锁在是使用完成后，需要进行销毁以释放资源。  
   int pthread\_mutex\_destroy(pthread\_mutex \*mutex);

### 二、条件变量(cond)

互斥锁不同，条件变量是用来等待而不是用来上锁的。条件变量用来自动阻塞一个线程，直到某特殊情况发生为止。通常条件变量和互斥锁同时使用。条件变量分为两部分: 条件和变量。条件本身是由互斥量保护的。线程在改变条件状态前先要锁住互斥量。条件变量使我们可以睡眠等待某种条件出现。条件变量是利用线程间共享的全局变量进行同步的一种机制，主要包括两个动作：一个线程等待"条件变量的条件成立"而挂起；另一个线程使"条件成立"（给出条件成立信号）。条件的检测是在互斥锁的保护下进行的。如果一个条件为假，一个线程自动阻塞，并释放等待状态改变的互斥锁。如果另一个线程改变了条件，它发信号给关联的条件变量，唤醒一个或多个等待它的线程，重新获得互斥锁，重新评价条件。如果两进程共享可读写的内存，条件变量可以被用来实现这两进程间的线程同步。

1. 初始化条件变量。  
   静态态初始化，pthread\_cond\_t cond = PTHREAD\_COND\_INITIALIER;  
   动态初始化，int pthread\_cond\_init(pthread\_cond\_t \*cond, pthread\_condattr\_t \*cond\_attr);
2. 等待条件成立。释放锁,同时阻塞等待条件变量为真才行。timewait()设置等待时间,仍未signal,返回ETIMEOUT(加锁保证只有一个线程wait)  
   int pthread\_cond\_wait(pthread\_cond\_t \*cond, pthread\_mutex\_t \*mutex);  
   int pthread\_cond\_timewait(pthread\_cond\_t \*cond,pthread\_mutex \*mutex,const timespec \*abstime);
3. 激活条件变量。pthread\_cond\_signal,pthread\_cond\_broadcast（激活所有等待线程）  
   int pthread\_cond\_signal(pthread\_cond\_t \*cond);  
   int pthread\_cond\_broadcast(pthread\_cond\_t \*cond); //解除所有线程的阻塞
4. 清除条件变量。无线程等待,否则返回EBUSY  
   int pthread\_cond\_destroy(pthread\_cond\_t \*cond);

### 三、信号量(sem)

如同进程一样，线程也可以通过信号量来实现通信，虽然是轻量级的。信号量函数的名字都以"sem\_"打头。线程使用的基本信号量函数有四个。

1. 信号量初始化。  
   int sem\_init (sem\_t \*sem , int pshared, unsigned int value);  
   这是对由sem指定的信号量进行初始化，设置好它的共享选项(linux 只支持为0，即表示它是当前进程的局部信号量)，然后给它一个初始值VALUE。
2. 等待信号量。给信号量减1，然后等待直到信号量的值大于0。  
   int sem\_wait(sem\_t \*sem);
3. 释放信号量。信号量值加1。并通知其他等待线程。  
   int sem\_post(sem\_t \*sem);
4. 销毁信号量。我们用完信号量后都它进行清理。归还占有的一切资源。  
   int sem\_destroy(sem\_t \*sem);