# linux多线程设计

## 概念

线程，有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。一个标准的线程由线程ID，当前指令[指针](http://baike.baidu.com/item/%E6%8C%87%E9%92%88)(PC），[寄存器](http://baike.baidu.com/item/%E5%AF%84%E5%AD%98%E5%99%A8)集合和[堆栈](http://baike.baidu.com/item/%E5%A0%86%E6%A0%88)组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以创建和撤消另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有[就绪](http://baike.baidu.com/item/%E5%B0%B1%E7%BB%AA)、[阻塞](http://baike.baidu.com/item/%E9%98%BB%E5%A1%9E)和[运行](http://baike.baidu.com/item/%E8%BF%90%E8%A1%8C)三种基本状态。就绪状态是指线程具备运行的所有条件，逻辑上可以运行，在等待处理机；运行状态是指线程占有处理机正在运行；阻塞状态是指线程在等待一个事件（如某个信号量），逻辑上不可执行。每一个程序都至少有一个线程，若程序只有一个线程，那就是程序本身。

线程是程序中一个单一的顺序控制流程。进程内一个相对独立的、可调度的执行单元，是系统独立调度和分派CPU的基本单位指[运行](http://baike.baidu.com/item/%E8%BF%90%E8%A1%8C)中的程序的调度单位。在单个程序中同时运行多个线程完成不同的工作，称为[多线程](http://baike.baidu.com/item/%E5%A4%9A%E7%BA%BF%E7%A8%8B)。

## 问题

为什么有了进程，还要引入线程呢？

优点：

1. 和进程相比，它是一种非常“节俭”的多任务操作方式。在linux系统下，启动一个新的进程必须分配给它的代码段、堆栈段和数据段，这是一种“昂贵”的多任务工作方式。运行一个进程中的多个线程，它们之间使用相同的地址空间，而且线程间切换需要的时间也远远小于进程间切换所需要的时间。据统计，一个进程的开销大约是一个线程开销的30倍左右。
2. 线程间方便的通信机制。对于不同进程来说，它们具有独立的数据空间，要进行数据的传递只能通过进程间的通信的方式进行，这种方式不仅费时，而且很不方便。线程则不然，由于同一进程下的线程之间共享数据空间，所以一个线程的数据可以直接为其它线程所用，这不仅快捷，而且方便。
3. 使多CPU系统更加有效。操作系统会保证当线程数不大于CPU数目时，不同的线程运行于不同的CPU上。改善程序结构。一个既长又复杂的进程可以考虑分个

线 程，成为几个独立或半独立的运行部分，这样的程序会利于理解和修改。

## linux下多线程编程

linux系统下的多线程遵循POSIX线程接口，称为pthread.编写Linux下的多线程程序，需要使用头文件pthread.h，链接是需要使用库libpthread.a。

创建线程

int pthread\_create(pthread\_t \*tidp,const pthread\_attr\_t \*attr,void \*(start\_rtn)(void),void \*arg)

tidp:线程id

attr:线程属性（通常为空）

start\_rtn:线程要执行的函数

arg:start\_rtn的参数

编译

#gcc filename –lpthread

终止线程

如果进程中任何一个线程中调用exit或者\_exit,那么整个进程都会终止。线程的正常退出方式有:

1. 线程从启动里程中返回
2. 线程可以被另一个进程终止
3. 线程自己调用pthread\_exit函数

线程等待

int pthread\_join(pthread\_t tid,void \*\*rval\_ptr)

功能：阻塞调用线程，直到指定的线程终止

tid :等待退出的线程id

Rval\_ptr:县城推出的返回值的指针

线程标识

pthread\_t pthread\_self(void)

获取调用线程的thread identifier

线程清除

线程终止有两种情况：正常终止和非正常终止。线程主动调用pthread\_exit或者从线程函数中return都将使线程正常退出，这是可预见的退出方式；非正常终止是 线程在其他线程的干预下，或者由自身运行出错（比如访问非法地址）而退出，这种退出方式是不可预见的。从pthread\_cleanup\_push的调用点到pthread\_cleanup\_pop之间的程序段中的终止动作（包括调用pthread\_exit()和异常终止 ，不包括return） 都将执行pthread\_cleanup\_push()所指定的清理函数。

void pthread\_cleanup\_push(void (\*rtn)(void \*),void \*arg)

将清除函数压入清除栈

void pthread\_cleanup\_pop(int execute)

将清除函数弹出清除栈

## 线程同步

工作原理

线程是[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B)中的实体，一个进程可以拥有多个线程，一个线程必须有一个[父进程](http://baike.baidu.com/item/%E7%88%B6%E8%BF%9B%E7%A8%8B)。线程不拥有[系统资源](http://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E8%B5%84%E6%BA%90)，只有[运行](http://baike.baidu.com/item/%E8%BF%90%E8%A1%8C)必须的一些[数据结构](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)；它与父进程的其它线程共享该进程所拥有的全部资源。线程可以创建和[撤消](http://baike.baidu.com/item/%E6%92%A4%E6%B6%88)线程，从而实现程序的并发执行。一般，线程具有[就绪](http://baike.baidu.com/item/%E5%B0%B1%E7%BB%AA)、[阻塞](http://baike.baidu.com/item/%E9%98%BB%E5%A1%9E)和[运行](http://baike.baidu.com/item/%E8%BF%90%E8%A1%8C)三种基本状态。在多[中央处理器](http://baike.baidu.com/item/%E4%B8%AD%E5%A4%AE%E5%A4%84%E7%90%86%E5%99%A8)的[系统](http://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F)里，不同线程可以同时在不同的中央处理器上[运行](http://baike.baidu.com/item/%E8%BF%90%E8%A1%8C)，甚至当它们属于同一个[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B)时也是如此。大多数支持多处理器的[操作系统](http://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F)都提供[编程接口](http://baike.baidu.com/item/%E7%BC%96%E7%A8%8B%E6%8E%A5%E5%8F%A3)来让[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B)可以控制自己的线程与各处理器之间的关联度（affinity）。有时候，线程也称作[轻量级进程](http://baike.baidu.com/item/%E8%BD%BB%E9%87%8F%E7%BA%A7%E8%BF%9B%E7%A8%8B)。就象[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B" \t "_blank)一样，线程在程序中是独立的、并发的执行路径，每个线程有它自己的[堆栈](http://baike.baidu.com/item/%E5%A0%86%E6%A0%88)、自己的[程序计数器](http://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%A1%E6%95%B0%E5%99%A8)和自己的[局部变量](http://baike.baidu.com/item/%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F)。但是，与分隔的[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B)相比，进程中的线程之间的隔离程度要小。它们[共享内存](http://baike.baidu.com/item/%E5%85%B1%E4%BA%AB%E5%86%85%E5%AD%98)、[文件](http://baike.baidu.com/item/%E6%96%87%E4%BB%B6)句柄和其它每个[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B)应有的状态。[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B)可以支持多个线程，它们看似同时执行，但互相之间并不同步。一个进程中的多个线程共享相同的内存地址空间，这就意味着它们可以访问相同的[变量](http://baike.baidu.com/item/%E5%8F%98%E9%87%8F)和[对象](http://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1)，而且它们从同一堆中分配对象。尽管这让线程之间共享信息变得更容易，但您必须小心，确保它们不会妨碍同一[进程](http://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B)里的其它线程。Java 线程工具和 [API](http://baike.baidu.com/item/API)看似简单。但是，编写有效使用线程的复杂程序并不十分容易。因为有多个线程共存在相同的内存空间中并共享相同的[变量](http://baike.baidu.com/item/%E5%8F%98%E9%87%8F)，所以您必须小心，确保您的线程不会互相干扰。

进行多线程编程，因为无法知道哪个线程会在哪个时候对共享资源进行操作，因此让如何保护共享资源变得复杂，通过下面这些技术的使用，可以解决线程之间对 资源的竞争 ：

1、互斥量Mutex

2、信号灯Semaphore

3、条件变量Conditions

互斥量

互斥量是一个可以处于两态之一的变量：解锁和加锁。这样，只需要一个[二进制](http://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6/361457)位表示它，不过实际上，常常使用一个[整型量](http://baike.baidu.com/item/%E6%95%B4%E5%9E%8B%E9%87%8F)，0表示解锁，而其他所有的值则表示加锁。互斥量使用两个过程。当一个[线程](http://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/103101)（或进程）需要访问[临界区](http://baike.baidu.com/item/%E4%B8%B4%E7%95%8C%E5%8C%BA/8942134)时，它调用mutex\_lock。如果该互斥量当前是解锁的（即临界区可用），此调用成功，调用线程可以自由进入该临界区。

另一方面，如果该互斥量已经加锁，调用线程被阻塞，直到在临界区中的线程完成并调用mutex\_unlock。如果多个线程被阻塞在该互斥量上，将随机选择一个线程并允许它获得锁。

**互斥量和信号量的区别**

**1. 互斥量用于线程的互斥，信号量用于线程的同步。**

这是互斥量和信号量的根本区别，也就是互斥和同步之间的区别。

互斥：是指某一资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。

同步：是指在互斥的基础上（大多数情况），通过其它机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源

**2. 互斥量值只能为0/1，信号量值可以为非负整数。**

也就是说，一个互斥量只能用于一个资源的互斥访问，它不能实现多个资源的多线程互斥问题。信号量可以实现多个同类资源的多线程互斥和同步。当信号量为单值信号量是，也可以完成一个资源的互斥访问。

3. **互斥量的加锁和解锁必须由同一线程分别对应使用，信号量可以由一个线程释放，另一个线程得到。**