流云哭翠

geyimeng.blog.chinaunix.net

永远记得 ：长远的路靠的必然是实力。

流云哭翠

博客访问： 514775
博文数量： 93
博客积分： 2283
博客等级： 大尉
技术积分： 1970
用 户 组： 普通用户
注册时间： 2012-07-15 22:19

加关注 短消息
论坛 加好友

文章分类

全部博文（93）
网络编程（7）
TCP/IP协议原理（10）
mysql（2）
面试笔试题目（5）
算法（4）
计算机基础（2）
C / C++（8）
linux C编程（18）
linux（29）
个人日记（1）
未分配的博文（7）

文章存档

2012年（93）

我的朋友


落雪风之

lmnos

minminms

KakitChe

bocaihua

zzywolf

wuyungan

leon_yu

C0114130

再谈互斥锁与条件变量！（终于搞清楚了啊！！！！！） 2012-07-21 08:48:13

分类： LINUX

pthread_cond_wait总和一个互斥锁结合使用。在调用pthread_cond_wait前要先获取锁。pthread_cond_wait函数执行时先自动释放指定的锁，然后等待条件变量的变化。在函数调用返回之前，自动将指定的互斥量重新锁住。

int pthread_cond_signal(pthread_cond_t * cond);

pthread_cond_signal通过条件变量cond发送消息，若多个消息在等待，它只唤醒一个。pthread_cond_broadcast可以唤醒所有。调用pthread_cond_signal后要立刻释放互斥锁，因为pthread_cond_wait的最后一步是要将指定的互斥量重新锁住，如果pthread_cond_signal之后没有释放互斥锁，pthread_cond_wait仍然要阻塞。

无论哪种等待方式，都必须和一个互斥锁配合，以防止多个线程同时请求pthread_cond_wait()（或pthread_cond_timedwait()，下同）的竞争条件（Race Condition）。mutex互斥锁必须是普通锁（PTHREAD_MUTEX_TIMED_NP）或者适应锁（PTHREAD_MUTEX_ADAPTIVE_NP），且在调用pthread_cond_wait()前必须由本线程加锁（pthread_mutex_lock()），而在更新条件等待队列以前，mutex保持锁定状态，并在线程挂起进入等待前解锁。在条件满足从而离开 pthread_cond_wait()之前，mutex将被重新加锁，以与进入pthread_cond_wait()前的加锁动作对应。

激发条件有两种形式，pthread_cond_signal()激活一个等待该条件的线程，存在多个等待线程时按入队顺序激活其中一个；而pthread_cond_broadcast()则激活所有等待线程。

下面是另一处说明：给出了函数运行全过程。 为什么在唤醒线程后要重新mutex加锁？

了解 pthread_cond_wait() 的作用非常重要 -- 它是 POSIX 线程信号发送系统的核心，也是最难以理解的部分。

首先，让我们考虑以下情况：线程为查看已链接列表而锁定了互斥对象，然而该列表恰巧是空的。这一特定线程什么也干不了 -- 其设计意图是从列表中除去节点，但是现在却没有节点。因此，它只能：

锁定互斥对象时，线程将调用 pthread_cond_wait(&mycond,&mymutex)。pthread_cond_wait() 调用相当复杂，因此我们每次只执行它的一个操作。

pthread_cond_wait() 所做的第一件事就是同时对互斥对象解锁（于是其它线程可以修改已链接列表），并等待条件 mycond 发生（这样当 pthread_cond_wait() 接收到另一个线程的“信号”时，它将苏醒）。现在互斥对象已被解锁，其它线程可以访问和修改已链接列表，可能还会添加项。 【要求解锁并阻塞是一个原子操作】

此时，pthread_cond_wait() 调用还未返回。对互斥对象解锁会立即发生，但等待条件 mycond 通常是一个阻塞操作，这意味着线程将睡眠，在它苏醒之前不会消耗 CPU 周期。这正是我们期待发生的情况。线程将一直睡眠，直到特定条件发生，在这期间不会发生任何浪费 CPU 时间的繁忙查询。从线程的角度来看，它只是在等待 pthread_cond_wait() 调用返回。

现在继续说明，假设另一个线程（称作“2 号线程”）锁定了 mymutex 并对已链接列表添加了一项。在对互斥对象解锁之后，2 号线程会立即调用函数 pthread_cond_broadcast(&mycond)。此操作之后，2 号线程将使所有等待 mycond 条件变量的线程立即苏醒。这意味着第一个线程（仍处于

pthread_cond_wait() 调用中）现在将苏醒。

现在，看一下第一个线程发生了什么。您可能会认为在 2 号线程调用 pthread_cond_broadcast(&mymutex) 之后，1 号线程的 pthread_cond_wait() 会立即返回。不是那样！实际上，pthread_cond_wait() 将执行最后一个操作：重新锁定 mymutex。一旦 pthread_cond_wait() 锁定了互斥对象，那么它将返回并允许 1 号线程继续执行。那时，它可以马上检查列表，查看它所感兴趣的更改。

来看一个例子（你是否能理解呢?）：

In Thread1:

```
pthread_mutex_lock(&m_mutex);
pthread_cond_wait(&m_cond,&m_mutex);
pthread_mutex_unlock(&m_mutex);
```

In Thread2:

```
pthread_mutex_lock(&m_mutex);
pthread_cond_signal(&m_cond);
pthread_mutex_unlock(&m_mutex);
```

为什么要与pthread_mutex 一起使用呢？ 这是为了应对 线程1在调用pthread_cond_wait()但线程1还没有进入wait cond的状态的时候，此时线程2调用了 cond_singal 的情况。 如果不用mutex锁的话，这个cond_singal就丢失了。加了锁的情况是，线程2必须等到 mutex 被释放（也就是 pthread_cod_wait() 释放锁并进入wait_cond状态 ，此时线程2上锁） 的时候才能调用cond_singal.

pthread_cond_signal即可以放在pthread_mutex_lock和pthread_mutex_unlock之间，也可以放在pthread_mutex_lock和pthread_mutex_unlock之后，但是各有有缺点。

之间：

```
pthread_mutex_lock
      xxxxxxx
pthread_cond_signal
pthread_mutex_unlock
```

缺点：在某下线程的实现中，会造成等待线程从内核中唤醒（由于cond_signal)然后又回到内核空间（因为cond_wait返回后会有原子加锁的 行为），所以一来一回会有性能的问题。但是在LinuxThreads或者NPTL里面，就不会有这个问题，因为在Linux 线程中，有两个队列，分别是cond_wait队列和mutex_lock队列， cond_signal只是让线程从cond_wait队列移到mutex_lock队列，而不用返回到用户空间，不会有性能的损耗。
所以在Linux中推荐使用这种模式。

之后：

```
pthread_mutex_lock
      xxxxxxx
pthread_mutex_unlock
pthread_cond_signal
```

优点：不会出现之前说的那个潜在的性能损耗，因为在signal之前就已经释放锁了
缺点：如果unlock和signal之前，有个低优先级的线程正在mutex上等待的话，那么这个低优先级的线程就会抢占高优先级的线程（cond_wait的线程），而这在上面的放中间的模式下是不会出现的。

上一篇：线程的要点总结
下一篇：linux内存管理

0

回顶部

相关热门文章

Linux环境变量	linux 常见服务端口	linux dhcp peizhi roc
contiki Makefile中添加projec...	xmanager 2.0 for linux配置	关于Unix文件的软链接
未来目标	【ROOTFS搭建】busybox的httpd...	求教这个命令什么意思，我是新...
expect basic 1: 那些转义，li...	openwrt中luci学习笔记	sed -e "/grep/d" 是什么意思...
String s = new String("abc")...	Linux里如何查找文件内容...	谁能够帮我解决LINUX 2.6 10...

给主人留下些什么吧！~~

评论热议

请登录后评论。
登录 注册