下半部和推后执行的工作

中断处理程序只是执行上半部分。但是由于本身存在一定的局限，所以他只能完成中断的上半部分，包括

- 中断处理程序以异步方式执行，并且他有可能会打断其他需要代码
- 如果当前有一个中断处理程序正在执行，因为当前处理器所有其他中断都会被屏蔽，因为禁止中断硬件与操作系统无法通信，因此中断处理程序执行的越快越好
- 由于中断处理程序往往需要对硬件进行操作，所以它们通常有很高的时限要求
- 中断处理程序不在进程上下文中运行，所以他们不能阻塞，限制了他们要做的事情

下半部

在理想的情况下，最好是中断处理程序的所有部分都交给下半部分去执行，我们一般会遵循这样一种规则

- 如果一个任务对时间非常敏感，将其放在中断处理程序中执行
- 如果一个任务和硬件相关，将其放在中断处理程序中运行
- 如果一个任务要保证不被其他中断，将其放在中断处理程序中执行

方法1：软中断

- 软中断是在编译期间静态编译的，他不想tasklet那样能被动态的注册和注销。
- 最多可能有32个不同的软中断，这是不可改变的
- 一个软中断不会抢占另外一个软中断。唯一能抢占软中断的就是中断处理程序，其他的软中断（甚至是相同类型的软中断）可以在其他处理器上同时运行
- 一个注册的软中断必须在标记后才会被执行，这被称作触发软中断。通常，中断处理程序会在返回前标记它的软中断，使其在稍后被执行，一般会在下面这几个地方
    - 从一个硬件中断代码返回处返回的时候
    - 在ksofitirqd内核线程中
    - 在那些显式检查和执行待处理的软中断代码中，如网络子系统中

软中断和简单的使用

软中断保留给系统中对时间要求最严格的下半部使用目前也只有网络和SCSI直接使用

- 首先，分配索引，在编译期间通过一个枚举类型来静态的声明软中断
- 然后，调用open-softirq注册软中断处理程序
- 通过在枚举类型的列表中添加新项以及调用open-softirq()进行注册后，新的软中断程序就能运行

在中断处理程序中触发软中断是最常见的形式，在这种情况下，中断处理程序执行硬件设备的相关操作，然后触发相应的软中断

方法2：tasklet

tasklet的工作方式

tasklet是利用软中断实现的一种下半部机制。我们之前提到过，他和进程没有任何关系，和软中断很相似，但是他更简单，方便使用，他的工作如下

- 首先禁止中断，然后检查对应的tasklet链表，分为低优先级和高优先级的链表
- 然后将当前处理器的该链表设置为null，清空链表
- 允许响应中断，在中断返回的时候开始处理tasklet（下半部）
- 循环遍历获得链表上每一个待处理的tasklet
- 如果是多处理器系统，我们就要判断这个tasklet是否是在其他处理器上在运行的，如果是，那么现在就不要执行，跳到下一个待处理的tasklet去。同一时间，相同的tasklet只能有一个执行
- 如果当前这个tasklet没有执行，就需要设置对应的状态。这样别的处理器就不会再执行它了
- 检查count值是否为0，确保tasklet没有被禁止。如果tasklet被禁止了则跳到下一个挂起的tasklet去
- 现在在这一切准备就绪之后，我们就开始运行tasklet处理陈或许
- 运行完毕后调整对应的状态和标志
- 重复下一个tasklet，直到处理完所有的tasklet

使用tasklet

- 你既可以静态地创建tasklet，也可以动态的创它
- 因为是靠软中断实现，所以tasklet不能睡眠。这意味着你不能在tasklet使用信号量或者什么其他的阻塞函数
- 由于tasklet运行时允许响应中断，所以你必须做好预防工作（如屏蔽中断然后获取一个锁）
- 通过调用tasklet schedule函数并传递给它相应的tasklet struct指针，该tasklet就会被调度以便执行

ksoftirqd

如果有大量的软中断同时出现，内核会唤醒一组内核线程来处理这些负载。这些线程在最低优先级运行。这个方案能够保障在软中断负担很重的时候，程序不会因为得不到处理时间而处于饥饿状态。但是过量的软中断也会处理。

每个处理器都有这样的线程。所有线程的名字就叫做ksoftirqd，他就是用来干这个的

方法3：工作队列

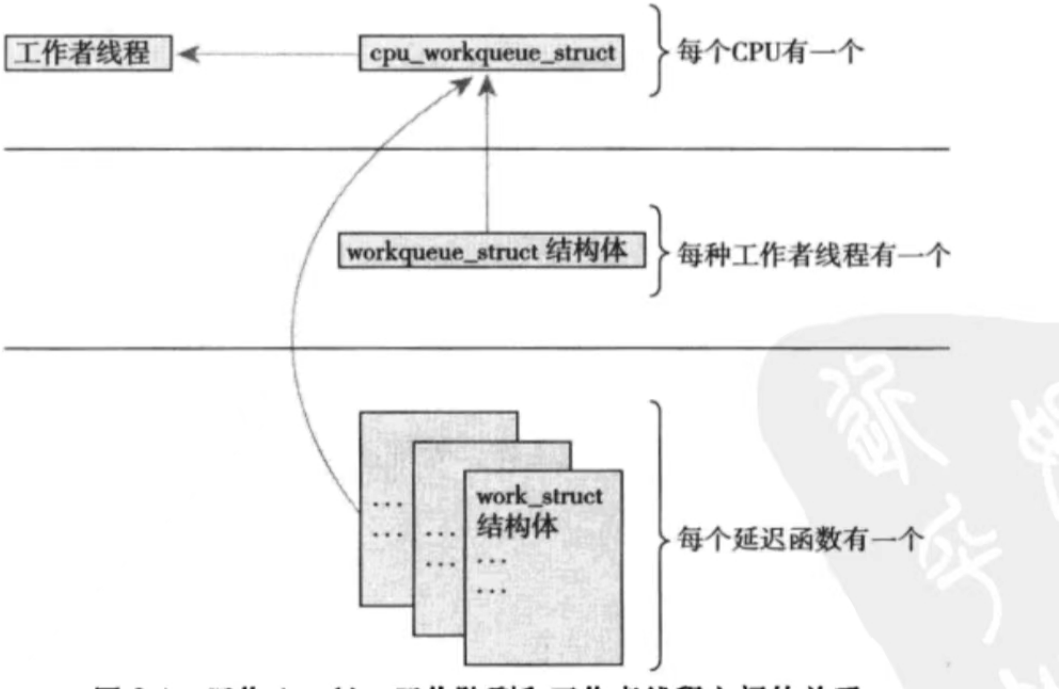
工作队列是另外一种将工作推后的形式，他和我们前面讨论的所有其他的形式都不相同

工作队列可以把工作推后，交由一个内核线程去执行，这个下半部分总会在进程上下文中执行

如果你需要在工作者线程中执行大量的处理操作，这样做会带来好处。处理器密集类型和性能要求严格的任务回英国拥有自己的工作者线程而获得好处，这么做也有助于减轻缺省线程的负担，避免工作队列中其他需要完成的工作处于饥饿状态

工作队列的实现

- 对于每一个workqueue-struct，他下面每一个cpu都有一个cpu-workqueue-struct
- 对于表示工作的数据结构，我们把它叫做woek-struct。他们都要执行work-thread工作。
- 这些结构体被链接成为链表，每个处理器上的每种类型的队列都对应这样一个链表
- 当一个工作者线程被唤醒时，他会执行它链表上所有的工作，工作被执行完毕，他就将相应的work-struct对象从链表上移去，当链表上不再有对象的时候，他就会继续休眠
- 图示如下



下半部机制的选择

- 一般情况下，不考虑软中断
- 如果代码多线索化考虑的并不充分，选择tasklet
- 如果要把任务推后到进程上下文完成，选择工作队列

下面是下半部接口的比较

表 8-3　对下半部的比较

下半部	上下文	顺序执行保障
软中断	中断	没有
tasklet	中断	同类型不能同时执行
工作队列	进程	没有（和进程上下文一样被调度）

内核同步

内核并发简介

内核有类似可能造成并发的原因，他们是：

- 中断：可以在任何时间异步发生，也可以随时打断当前正在执行的代码
- 软中断和tasklet：内核能在任何时刻唤醒或调度软中断和tasklet，打断当前正在执行的代码
- 内核抢占：内核具有抢占性，所以内核中的任务可能会被另外一个任务抢占
- 睡眠以及用户空间的同步：在内核执行进程可能会睡眠，这就会唤醒调度程序，从而导致调度一个新的用户进程执行
- 对称多处理：两个或多个处理器可以同时执行代码

在中断处理程序中，能够避免并发访问的安全代码称作中断安全代码，在对称多处理的机器中能避免并发访问的安全代码称为SMP安全代码，在内核抢占能避免并发的安全代码称为抢占安全代码

了解需要保护什么

找出哪些数据需要保护是关键所在，寻找那些代码不需要保护反而是相对容易的

要给数据枷锁而不是给代码加锁

死锁

预防死锁的简单规则

- 按顺序加锁
- 防止发生饥饿：如果A一直不发生那么B需要一直等待吗?
- 不要重复请求同一个锁
- 设计要简单

原子操作

原子操作是其他同步方法的基石。它可以保证以指令以原子的方式执行，执行的过程不被打断

我们建立了专门的原子操作，这么做的目的是

- 首先，让原子函数只接受原子类型的操作数，确保原子操作只与这一特殊类型来使用
- 确定该类型的数据不会被传递给任何非原子函数

原子性和顺序性都是需要保证的，其中原子性要通过原子操作保证，顺序性通过屏障（barrier）来保证

读写自旋锁

对于读写自旋锁，我们要防止一个问题就是可能导致的死锁

- 假如这个读者正在进行操作，包括写操作的中断发生了
- 由于读锁还没有全部被释放，写操作会自选，但是读操作只能在包含写操作的中断返回后才能继续
- 于是这样死锁就发生了

针对这种情况，读锁的作用不单单是对读进行保护，还需要禁止有写操作的中断，否则就会有死锁

顺序锁

顺序锁，通常叫做seq锁

- 当有疑义的数据写入的时候，会得到一个锁，而且序列值会增加。
- 在读取数据之前和之后，序列号都会被读取。如果读取的序列号相同，说明在读操作进行的过程中没有被写操作打断过。
- 此外如果你读的值是偶数，那么就表明写操作没有发生

在下面这几种情况下最适合用seq锁

- 你的数据存在很多读者
- 你的数据写者很少
- 虽然写者很少，但是你希望写优先于读，而且不允许读者让写者饥饿
- 你的数据比较简单

顺序与屏障

当处理多处理器之间或硬件设备之间的同步问题时，有时需要在你的程序代码中以指定的顺序发出读和写内存的指令，但是很多时候难以这么做。但是通过屏障可以保证不这么做

- rmb方法提供了一个读的内存屏障。它确保跨越rmb的载入动作不会发生重排序
- wmb方法提供了一个写的内存屏障，这个函数功能和前面的相似，区别是它是针对储存而非载入。

定时器和时间管理

内核中的时间概念

内核提供了两种设备计时：系统定时器和硬件时钟，实体时钟是用来存放时间的设备

- 内核必须在硬件的帮助下才能计算和管理时间
- 系统定时器以某种频率自行触发时钟中断
- 时钟中断对管理系统尤为重要。有一些利用时间中断执行的工作
    - 更新系统运行时间
    - 更新实际时间
    - 更新资源消耗和处理器时间的统计值

理想的HZ值

提高节拍率值意味着时钟中断产生会更加频繁，对应也就是中断处理程序会更加频繁的执行

好处有

- 更高的时钟中断解析度
- 提高时间驱动事件的准确度
- 内核定时器能够以更高的频度和准确度运行
- 对系统运行时间的测量会更加精细
- 提高进程抢占的准确度

坏处

- 会频繁打断处理器高速缓存并且增加耗电

我们使用全局变量jiffes来记录系统启动以来产生的节拍总数

延迟执行

忙等待

最简单的延迟方法是忙等待。也就是死循环

短延迟

对于你延迟的话可以使用udelay等代码进行

schedule-timeout

这个方法会让延迟任务睡眠到指定的时间之后再运行。但是不能保证刚好等于，只能接近