7.2 内核同步措施

西安邮电大学

内核同步措施

- 为了避免并发，防止竞争。内核提供了一组同步方法来提供对共享数据的保护，原子操作在前面已经介绍过了，下面介绍其他几种。

中断屏蔽

使用

```
local_irq_disable() //屏蔽中断
...                 //临界区
local_irq_enable() //开中断
```

在进入临界区之前屏蔽系统的中断，从而保证正在执行的内核任务不被中断处理程序所抢占，防止某些静态条件的发生。在退出临界区后，重新打开中断。

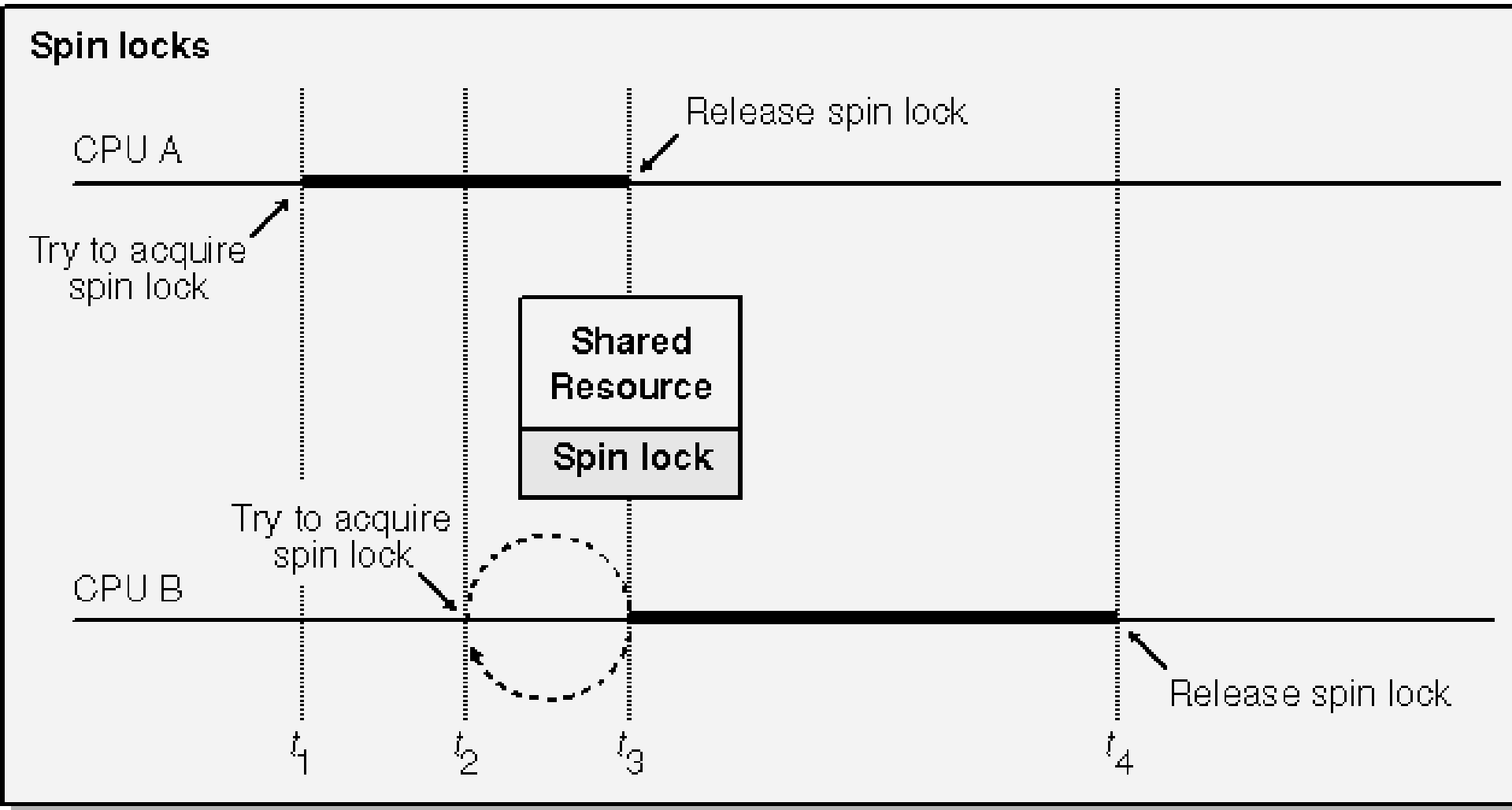
中断屏蔽的缺点

- local_irq_disable()和local_irq_enable()这两个函数都只能禁止和开启本地CPU内的中断，并不能解决多处理器引发的竞态(并行)。
- 在屏蔽中断期间所有的中断都无法得到处理，（讲完这句话后出现下面的内容）

禁止/打开当前处理器上所有的中断：
local_irq_save()/local_irq_restore()
local_irq_disable()/local_irq_enable()

因此长时间屏蔽中断是很危险的，有可能造成数据丢失甚至系统崩溃

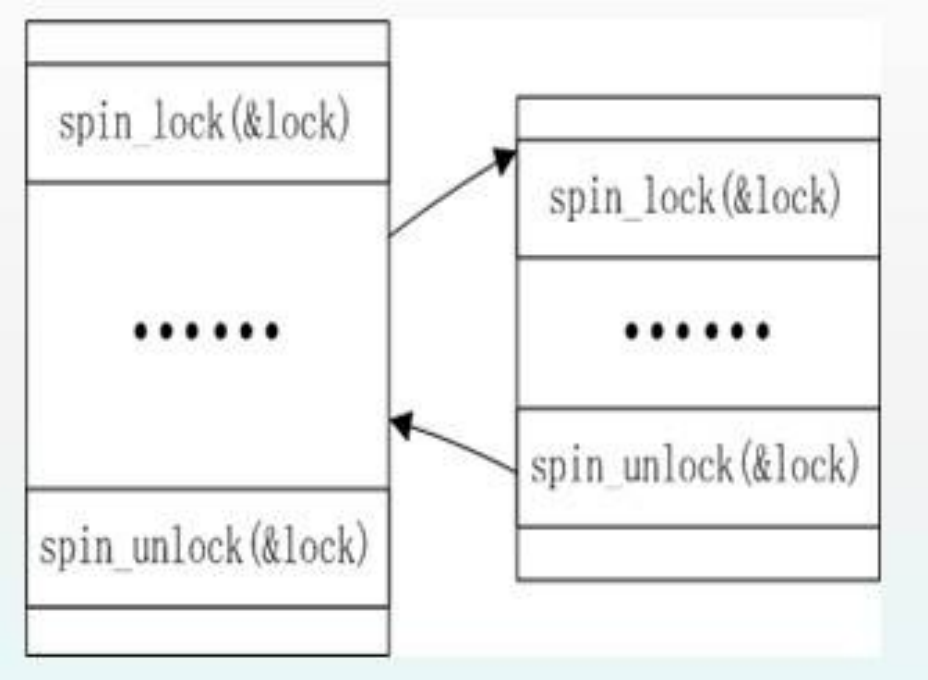
自旋锁


Spin locks
CPU A
Release spin lock
Try to acquire spin lock
Shared Resource
Spin lock
Try to acquire spin lock
CPU B
Release spin lock
t_1
t_2
t_3
t_4

自旋锁

- 自旋锁是专为防止多处理器并发而引入的一种锁，它在内核中大量应用于中断处理等部分，而对于单处理器来说，可简单采用关闭中断的方式就可以防止中断处理程序的并发执行
- 自旋锁最多只能被一个内核任务持有，比如，图中，在t1时刻，CPU A上的任务试图获得自选锁，并且成功获得，进入临界区，t2时刻，CPU B上的任务试图申请自旋锁，这个任务就会一直进行忙循环，也就是旋转，直到t3时刻锁被释放。

自旋锁



- 设计自旋锁的初衷是在短期间内进行轻量级的锁定，所以自旋锁不应该被持有时间过长。
- 自旋锁在内核中主要用来防止多处理器中并发访问临界区，防止内核抢占造成的竞争
- 自旋锁不允许任务睡眠，持有自旋锁的任务睡眠会造成自死锁，因此自旋锁能够在中断上下文中使用。

自旋锁

- 自旋锁的定义如下：

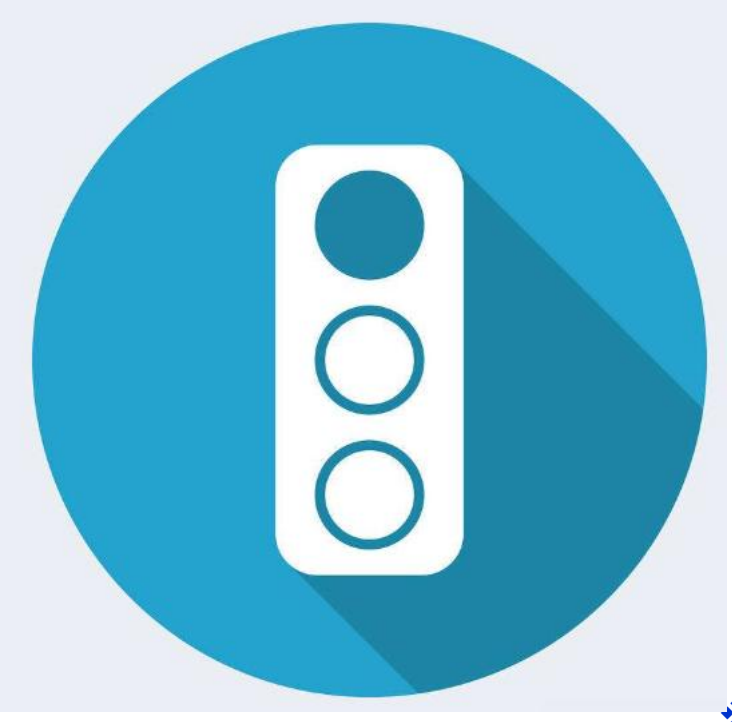
```
typedef struct raw_spinlock { unsigned int slock;} raw_spinlock_t;
typedef struct {
        raw_spinlock_t raw_lock;
            ...
    } spinlock _t;
```

- 使用自旋锁的基本形式如下：

```
DEFINE_SPINLOCK(mr_lock); /* 定义一个自旋锁 */
    spin_lock(&mr_lock);
    /*临界区*/
    spin_unlock(&mr_lock);
```

关于自旋锁还有很多的变种，内核提供了一组API，在此就不一一介绍了

信号量



- Linux中的信号量是一种睡眠锁。若有一个任务试图获得一个已被持有的信号量时，信号量会将其推入等待队列，然后让其睡眠。这时处理器获得自由而去执行其它代码。当持有信号量的进程将信号量释放后，在等待队列中的一个任务将被唤醒，从而便可以获得这个信号量
- 信号量具有睡眠特性，适用于锁会被长时间持有的情况，只能在进程上下文中使用。

内核中如何定义信号量

```
struct semaphore {
        spinlock_t      lock;
        unsigned int    count;
        struct list_head  wait_list;
}s
```

lock：自旋锁，防止多处理器并行造成错误。
count：计数器，大于0，表示可用资源数，小于0，其绝对值表示等待的进程数
wait_list：等待资源的进程队列。

信号量的相关操作

- down()操作，也就是P操作

```
void down(struct semaphore *sem)
{
        unsigned long flags;
          spin_lock_irqsave(&sem->lock, flags); /*加锁，使信号量的
        操作在关闭中断的状态下进行，防止多处
                理器并发操作造成错误*/
        if (sem->count > 0)) /*若信号量可用，则将引用计数减1 */
                sem->count--;
        else /*如果无信号量可用，则调用__down()函数进入睡眠
                                等待状态 */
                __down(sem);
        spin_unlock_irqrestore(&sem->lock, flags); /* 对信号量的
                                操作解锁 */
}
```

- down()操作中调用_down()函数，而__down调用__down_common(),后者是各种down()操作的统一函数。

信号量的相关操作

- 释放信号量的up()操作，相当于V操作：

```
    void up(struct semaphore *sem)
    {
            unsigned long flags;
            spin_lock_irqsave(&sem->lock, flags); /* 对信号量操作
                        进行加锁 */
            if list_empty(&sem->wait_list) /* 如果该信号量的等待
                  队列为空，则释放信号量 */
                    sem->count++;
            else /* 否则唤醒该信号量的等待队列队头的进程 */
                    __up(sem);
            spin_unlock_irqrestore(&sem->lock, flags); /* 对信号 操作
进行解锁 */
    }
```

信号量的操作函数列表

函数	描述
down(struct semaphore *);	获取信号量，如果不可获取，则进入不可中断睡眠状态（目前已经不建议使用）。
down_interruptible(struct semaphore *);	获取信号量，如果不可获取，则进入可中断睡眠状态
down_killable(struct semaphore *);	获取信号量，如果不可获取，则进入可被致命信号中断的睡眠状态。
down_trylock(struct semaphore *);	尝试获取信号量，如果不能获取，则立刻返回
down_timeout(struct semaphore *, long jiffies);	在给定时间（jiffies）内获取信号量，如果不能够获取，则返回。
up(struct semaphore *);	释放信号量。

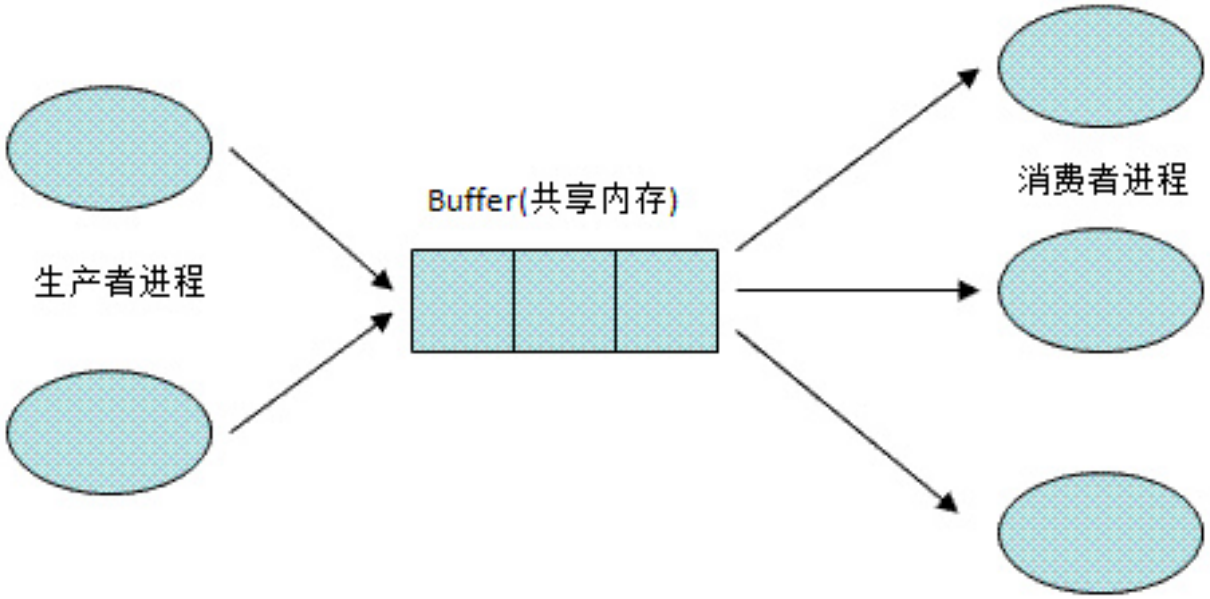
信号量与自旋锁的比较

需求	建议的加锁方法
低开销加锁	优先使用自旋锁
短期锁定	优先使用自旋锁
长期加锁	优先使用信号量
中断上下文中加锁	使用自旋锁
持有锁时需要睡眠、调度	使用信号量

内核其它的同步措施

同步措施	含义
互斥锁	互斥锁和信号量为1的信号量含义类似，可以允许睡眠。
完成变量	完成量是基于等到队列机制的，如果在内核中一个任务需要发出信号通知另一个任务发生了某个特定的事件，可以用完成变量。
RCU机制	读-拷贝修改锁机制，允许多个读者同时访问被保护的数据，又允许多个读者和多个写者同时访问被保护的数据。

生产者-消费者并发实例



- 问题描述

一个生产厂家、一个经销商、一个仓库。 厂家生产一批产品并放在仓库里，再通知经销商来批发。经销商卖完产品再向厂家下订单，生产厂家才生产下一批产品。

- 问题分析

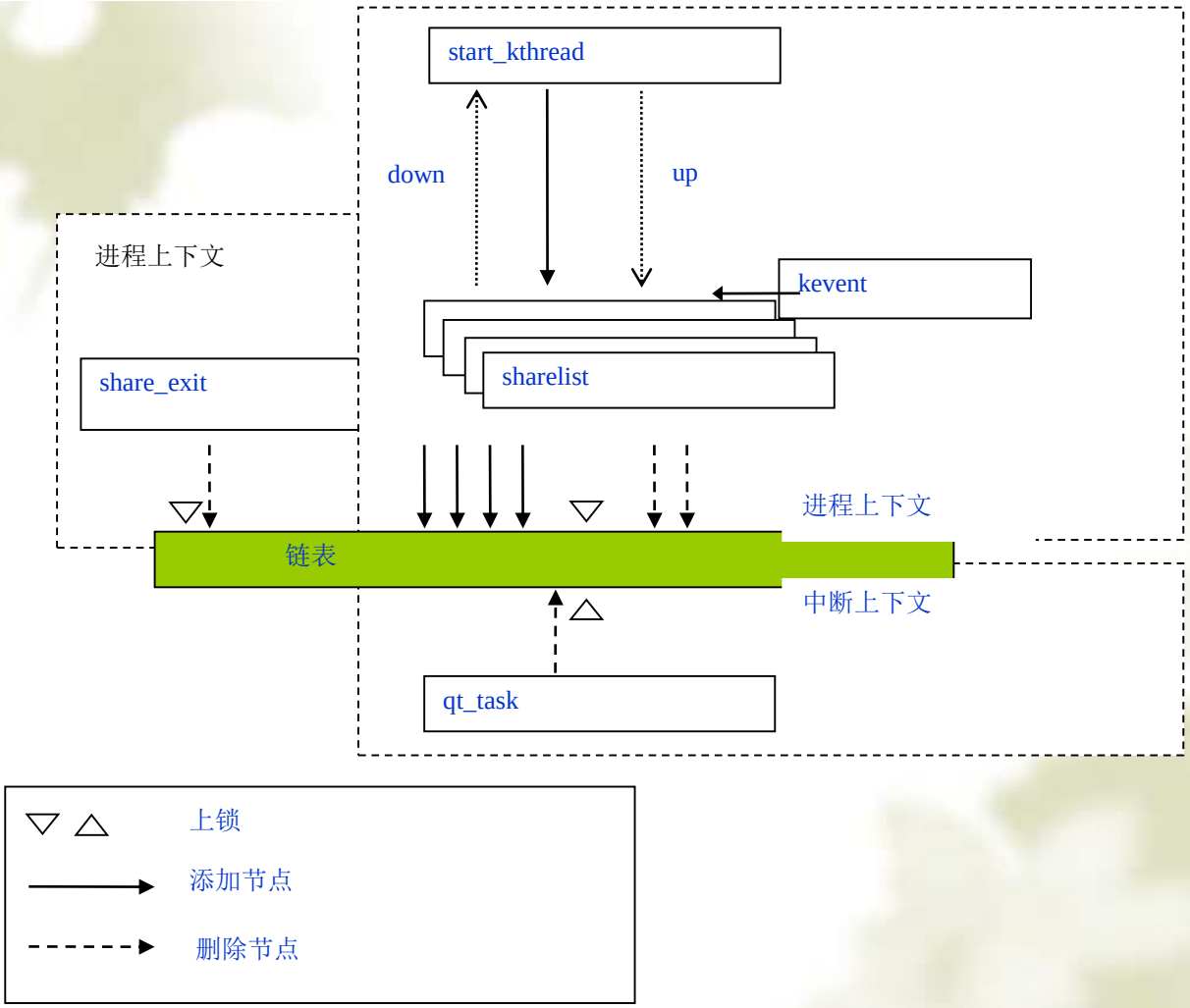
这是典型的生产者和消费问题，生产者和消费者用内核线程来模拟，“公共缓冲区”为临界区。同一时刻，只能有一个线程访问临界区。

具体代码参见教材。

内核多任务并发控制实例

- 假设存在这样一个的内核共享资源—链表。另外我们构造一个内核多任务访问链表的场景：内核线程向链表加入新节点；内核定时器定时删除结点；系统调用销毁链表。
- 上面三种内核任务并发执行时，有可能会破坏链表数据的完整性，所以我们必须对链表进行同步访问保护，以确保数据一致性。
- 本实例讲在下一讲中视频录制中具体讲解。

内核多任务并发控制实例

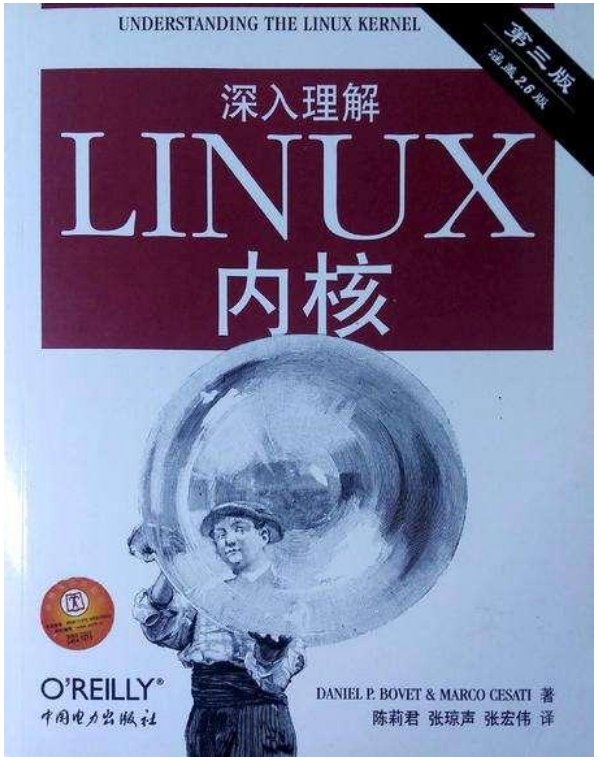


并发控制实例示意图

动手实践

- Linux内核之旅网站 http://www.kerneltravel.net/
- 第七期“内核中的调度与同步”为大家介绍内核中存在的各种任务调度机理以及它们之间的逻辑关系，在此基础上向大家解释内核中需要同步保护的根本原因和保护方法，
- 并提供了一个内核共享链表同步访问的例子，帮助大家理解内核编程中的同步问题。
- 下载其代码并在最新内核版下进行调试

参考资料



深入理解Linux内核 第三版第五章

Linux内核设计与实现 第三版第十章

http://www.wowotech.net/ ，蜗窝科技网站内核同步有系列文章

带着思考离开

在多核环境下，中断屏蔽机制与单核有什么差异？

谢谢大家！

THANK YOU