openEuler内核编程技术

实训指导书

第四章 第4讲

《锁机制》

软件所制

**术语表**

|  |  |  |  |
| --- | --- | --- | --- |
| 英文 | 中文 | 含义 | 别名 |
|  |  |  |  |
|  |  |  |  |

第四章 第4讲 锁机制

# 实训1：设计并实现一个内核模块，在加载到内核时将输出的信息保存到当前目录下的output.txt中，output.txt中周期打印1，2，3…(60min)

## 相关知识

1. **锁概念**

加锁（locking）是一种广泛应用的同步技术。当内核控制路径必须访问共享数据结构或进入临界区时，就需要为自己获取一把“锁”。由锁机制保护的资源非常类似于限制于房间内的资源，当某人进入房间时，就把门锁上。如果内核控制路径希望访问资源，就试图获取钥匙“打开门”。当且仅当资源空闲时，它才能成功。然后，只要它还想使用这个资源，门就依然锁着。当内核控制路径释放了锁时，门就打开，另一个内核控制路径就可以进入房间。

1. **自旋锁概念**

Linux锁的应用之一在多处理器环境中，取名叫自旋锁（spin lock）。如果内核控制路径发现自旋锁“开着”，就获取锁并继续自己的执行。相反，如果内核控制路径发现锁由运行在另一个CPU上的内核控制路径“锁着”，就在周围“旋转”，反复执行一条紧凑的循环指令，直到锁被释放。

自旋锁的循环指令表示“忙等”。即使等待的内核控制路径无事可做（除了浪费时间），它也在CPU上保持运行。不过，自旋锁通常非常方便，因为很多内核资源只锁1毫秒的时间片段；所以说，等待自旋锁的释放不会消耗太多CPU的时间。

一般来说，由自旋锁所保护的每个临界区都是禁止内核抢占的。在单处理器系统上，这种锁本身并不起锁的作用，自旋锁技术仅仅是用来禁止或启用内核抢占。请注意，在自旋锁忙等期间，因为并没有进入临界区，所以内核抢占还是有效的，因此，等待自旋锁释放的进程有可能被更高优先级的所取代。这种设计是合理的，因为不能因为占用CPU太久而使系统死锁。

1. **通用自旋锁**

自旋锁的状态值为1表示解锁状态，说明有1个资源可用；0或负值表示加锁状态，0说明可用资源数为0。Linux内核为通用自旋锁提供了API函数初始化、测试和设置自旋锁。

1） 通用自旋锁API函数功能说明

宏定义 功能说明

spin\_lock\_init(lock) 初始化自旋锁，将自旋锁设置为1，表示有一个资源可用。

spin\_is\_locked(lock) 如果自旋锁被置为1（未锁），返回0，否则返回1。

spin\_unlock\_wait(lock) 等待直到自旋锁解锁（为1），返回0；否则返回1。

spin\_trylock(lock) 尝试锁上自旋锁（置0），如果原来锁的值为1，返回1，否则返回0。

spin\_lock(lock) 循环等待直到自旋锁解锁（置为1），然后，将自旋锁锁上（置为0）。

spin\_unlock(lock) 将自旋锁解锁（置为1）。

spin\_lock\_irqsave(lock, flags) 循环等待直到自旋锁解锁（置为1），然后，将自旋锁锁上（置为0）。关中断，将状态寄存器值存入flags。

spin\_unlock\_irqrestore(lock, flags) 将自旋锁解锁（置为1）。开中断，将状态寄存器值从flags存入状态寄存器。

spin\_lock\_irq(lock) 循环等待直到自旋锁解锁（置为1），然后，将自旋锁锁上（置为0）。关中断。

spin\_unlock\_irq(lock) 将自旋锁解锁（置为1）。开中断。

spin\_unlock\_bh(lock) 将自旋锁解锁（置为1）。开启底半部的执行。

spin\_lock\_bh(lock) 循环等待直到自旋锁解锁（置为1），然后，将自旋锁锁上（置为0）。阻止软中断的底半部的执行。

2）通用自旋锁结构

在Linux中，每个自旋锁都用spinlock\_t结构表示：

typedef struct {

raw\_spinlock\_t raw\_lock;

#ifdef CONFIG\_GENERIC\_LOCKBREAK /\*引入另一个自旋锁\*/

unsigned int break\_lock;

#endif

#ifdef CONFIG\_DEBUG\_SPINLOCK /\*用于调试自旋锁\*/

unsigned int magic, owner\_cpu;

void \*owner;

#endif

#ifdef CONFIG\_DEBUG\_LOCK\_ALLOC

struct lockdep\_map dep\_map; /\*映射lock实例到lock-class对象

#endif

} spinlock\_t;

由于自旋锁的性能严重地影响着操作系统的性能，Linux内核提供了Lock-class和Lockdep跟踪自旋锁的使用对象和锁的状态，并可从/proc文件系统查询自旋锁的状态信息。自旋锁的调试通过配置项CONFIG\_DEBUG\_\*项打开。

对于对称多处理器系统（SMP），slock为一个int数据类型，对于单个处理器系统，slock定义为空。SMP的slock定义列出如下（在include/linux/spinlock\_types.h）：

typedef struct {

volatile unsigned int slock;

} raw\_spinlock\_t;

其中包含两个重要的字段意义如下：

slock：该字段表示自旋锁的状态：值为1表示“未加锁”状态，而任何负数和0都表示“加锁”状态。

break\_lock：表示进程正在忙等自旋锁（只在内核支持SMP和内核抢占的情况下使用该标志）。

1. **读写自旋锁**

"读/写自旋锁"用来解决读者/写者问题。如果有多个线程（进程、中断处理程序、底半部例程）以只读的方式访问一个临界区数据，读/写自旋锁允许多个线程同时读取数据。如果一个线程需要对临界区数据进行写操作，它必须获取写锁，只有在没有读者或写者进行操作时，写者才独占临界区数据进行写操作。读操作时需要获取读锁，写操作时需要获取写锁。

通俗易懂的说，就是读申请读锁，写申请写锁。读的时候，如果有写操作正在进行，则等待，其他情况就可以直接读操作。写的时候，如果有读操作或者写操作正在进行，那么等待。没有操作正在进行的时候，才可以进行写操作。

Linux内核为读/写自旋锁提供了操作API函数初始化、测试和设置自旋锁。

1）读写自旋锁API函数功能说明

宏定义 功能说明

rwlock\_init(lock) 初始化自旋锁值为0x01000000（未锁）。

read\_lock(lock) 加读者锁，即将读者计数加1。

read\_lock\_irqsave(lock, flags) 加读者锁，即将读者计数加1。并且关中断，存储状态标识到flags中。

read\_lock\_irq(lock) 加读者锁，即将读者计数加1。并且关中断。

read\_unlock(lock) 解读者锁，即将读者计数减1。

read\_unlock\_irqrestore(lock, flags) 解读者锁，即将读者计数减1。并且开中断，将状态标识从flags读到状态寄存器中。

read\_unlock\_irq(lock) 解读者锁，即将读者计数减1。并且开中断。

write\_lock(lock) 加写者锁，即将写者锁置0。

write\_lock\_irqrestore(lock, flags) 加写者锁，即将写者锁置0。并且关中断，存储状态标识到flags中。

write\_lock\_irq(lock) 加写者锁，即将写者锁置0。并且关中断。

write\_unlock(lock) 解写者锁，即将写者锁置1。

write\_unlock\_irqrestore(lock, flags) 解写者锁，即将写者锁置1。并且开中断，将状态标识从flags读到状态寄存器中。

write\_unlock\_irq(lock) 解写者锁，即将写者锁置1。并且开中断。

2）读写自旋锁结构

用户使用读/写自旋锁，应先自旋锁的状态值初始化为锁初始化为RW\_LOCK\_BIAS，即0x01000000，表示为未锁状态。

读/写自旋锁用结构rwlock\_t描述，它的主要成员为锁状态值变量lock，结构rwlock\_t列出如下（在include/linux/spinlock\_types.h中）：

typedef struct {

raw\_rwlock\_t raw\_lock;

……

} rwlock\_t;

typedef struct {

unsigned int lock;

} raw\_rwlock\_t;

在结构raw\_rwlock\_t中，读/写自旋锁状态变量lock为32位，它分为2个部分，0~23位是一个24位计数器，表示对临界数据区进行并发读操作的线程数，线程数以补码形式存入计数器；第24位为表示"未锁"的状态位，在没有线程读或写临界区时，设置为1，否则，设置为0。

如果自旋锁设置了"未锁"状态且无读者，那么lock值为0x01000000；如果写者已获得自旋锁且无读者，则未锁状态位清0，lock值为0x00000000。如果有一个、2个或多个线程获取锁对临界数据区进行读操作，则lock值为0x00ffffff、0x00fffffe等（第24位清0表示未锁，第0~23位为读者个数的补码）。

1. **顺序锁**

顺序锁是对读/写锁的优化，它允许读写同时进行，提高了并发性，读写操作同时进行的概率较小时，其性能很好。顺序锁对读/写锁进行了下面的改进：

写者不会阻塞读者，即写操作时，读者仍可以进行读操作。

写者不需要等待所有读者完成读操作后才进行写操作。

写者与写者之间互斥，即如果有写者在写操作时，其他写者必须自旋等待。

如果在读者进行读操作期间，有写者进行写操作，那么读者必须重新读取数据，确保读取正确的数据。

要求临界区的共享资源不含指针，因为如果写者使指针失效，读者访问该指针，将导致崩溃。

1）顺序锁API函数功能说明

顺序锁实际上由一个自旋锁和一个顺序计数器组成，有的应用已包括自旋锁，只需要一个顺序计数器配合就可以实现顺序锁。针对这两种情况，Linux内核给顺序锁提供了两套API函数。一套API函数为\*seq\*，完整地实现了顺序锁；另一套API函数为\*seqcount\*，只包含了顺序计数器，需要与用户的自旋锁配套实现顺序锁。

函数名 功能说明

seqlock\_init(x) 初始化顺序锁，将顺序计数器置0。

write\_seqlock(seqlock\_t \*sl) 加顺序锁，将顺序号加1。写者获取顺序锁s1访问临界区，它使用了函数spin\_lock。

write\_sequnlock(seqlock\_t \*sl) 解顺序锁，使用了函数spin\_unlock，顺序号加1。

write\_tryseqlock(seqlock\_t \*sl) 功能上等同于spin\_trylock，顺序号加1。

read\_seqbegin(const seqlock\_t \*sl) 返回顺序锁s1的当前顺序号，读者没有开锁和释放锁的开销。

read\_seqretry(const seqlock\_t \*sl, unsigned start) 检查读操作期间是否有写者访问了共享资源，如果是，读者就需要重新进行读操作，否则，读者成功完成了读操作。

seqcount\_init(x) 初始化顺序号。

read\_seqcount\_begin(const seqcount\_t \*s) 读者在读操作前用此函数获取当前的顺序号。

read\_seqcount\_retry(const seqcount\_t \*s, unsigned start) 读者在访问完后调用此函数检查在读期间是否有写者访问临界区。如果有，读者需要重新进行读操作，否则，完成读操作。

write\_seqcount\_begin(seqcount\_t \*s) 写者在访问临界区前调用此函数将顺序号加1，以便读者检查是否在读期间有写者访问过。

write\_seqcount\_end(seqcount\_t \*s) 写者写完成后调用此函数将顺序号加1，以便读者能检查出是否在读期间有写者访问过。

2）顺序锁的数据结构

顺序锁用结构seqlock\_t描述，它包括顺序计数器sequence和自旋锁lock。结构seqlock\_t列出如下（在include/linux/seqlock.h中）：

typedef struct {

unsigned sequence;

spinlock\_t lock;

} seqlock\_t;

在结构seqlock\_t中，顺序计数器sequence存放顺序号，每个读者在读数据前后两次读顺序计数器，并检查两次读到的顺序号是否相同。如果不相同，说明新的写者已经开始写并增加了顺序计数器，表明刚读到的数据无效。

写者通过调用函数write\_seqlock获取顺序锁，将顺序号加1，调用函数write\_sequnlock释放顺序锁，再将顺序号加1。这样，写者正在写操作时，顺序号为奇数，写完临界区数据后，顺序号为偶数。

读者应以循环查询方法读取临界区数据，读者执行的临界区代码的方法列出如下：

do {

seq = read\_seqbegin(&foo); //返回当前的顺序号

... //临界区数据操作

} while (read\_seqretry(&foo, seq));

在上述代码中，读者在读临界区数据之前，先调用函数read\_seqbegin获致当前的顺序号，如果顺序号seq为奇数，说明写者正写临界区数据，或者seq值与顺序号当前值不等，表明读者正读时，写者开始写，函数read\_seqretry返回1，读者继续循环等待写者完成。

## 任务描述

1、在内核模块中，创建并运行一个线程，这个线程实现的功能是：在加载到内核的时候将输出的信息保存到当前目录下的output.txt中，output.txt中周期打印1，2，3…

## 审核要求

1. 正确编写满足功能的源文件，正确编译。
2. 正常加载、卸载内核模块；且内核模块功能满足任务所述。
3. 提交相关源码与运行截图。

## 参考答案

1、环境准备

（1）内核源码准备

查看系统下是否存在当前内核版本的源码：

|  |
| --- |
| # ls |

如果不存在，则下载对应版本的内核源码：

此处示例的内核版本对应的源码是：https://gitee.com/openeuler/raspberrypi-kernel/repository/archive/openEuler-20.03-LTS-raspi.zip

下载完解压之后，拷贝到上述系统目录下（或者其他指定目录下，目录需与Makefile中所述一致）。

|  |
| --- |
| # wget https://gitee.com/openeuler/raspberrypi-kernel/repository/archive/openEuler-20.03-LTS-raspi.zip # 下载  # unzip openeuler-raspberrypi-kernel-openEuler-20.03-LTS-raspi.zip # 解压 |

（2）内核源码编译

|  |
| --- |
| # cd raspberrypi-kernel  # make openeuler-raspi\_defconfig # 加载内核配置  # make prepare && make scripts # 内核预编译（耗时短） |

|  |
| --- |
| [root@openEuler build]# pwd  /usr/lib/modules/4.19.90-00782-g241fba542c1a /build  [root@openEuler build]#  [root@openEuler build]# make openeuler-raspi\_defconfig  HOSTCC scripts/basic/fixdep  HOSTCC scripts/kconfig/conf.o  YACC scripts/kconfig/zconf.tab.c  LEX scripts/kconfig/zconf.lex.c  HOSTCC scripts/kconfig/zconf.tab.o  HOSTLD scripts/kconfig/conf  #  # configuration written to .config  #  [root@openEuler build]#  [root@openEuler build]# make prepare  UPD include/config/kernel.release  WRAP arch/arm64/include/generated/uapi/asm/errno.h  WRAP arch/arm64/include/generated/uapi/asm/ioctl.h  WRAP arch/arm64/include/generated/uapi/asm/ioctls.h  WRAP arch/arm64/include/generated/uapi/asm/ipcbuf.h  WRAP arch/arm64/include/generated/uapi/asm/kvm\_para.h  WRAP arch/arm64/include/generated/uapi/asm/mman.h  WRAP arch/arm64/include/generated/uapi/asm/msgbuf.h  WRAP arch/arm64/include/generated/uapi/asm/poll.h  WRAP arch/arm64/include/generated/uapi/asm/resource.h  WRAP arch/arm64/include/generated/uapi/asm/sembuf.h  WRAP arch/arm64/include/generated/uapi/asm/shmbuf.h  WRAP arch/arm64/include/generated/uapi/asm/socket.h  WRAP arch/arm64/include/generated/uapi/asm/sockios.h  WRAP arch/arm64/include/generated/uapi/asm/swab.h  WRAP arch/arm64/include/generated/uapi/asm/termbits.h  WRAP arch/arm64/include/generated/uapi/asm/termios.h  WRAP arch/arm64/include/generated/uapi/asm/types.h  WRAP arch/arm64/include/generated/asm/bugs.h  WRAP arch/arm64/include/generated/asm/delay.h  WRAP arch/arm64/include/generated/asm/div64.h  WRAP arch/arm64/include/generated/asm/dma.h  WRAP arch/arm64/include/generated/asm/dma-contiguous.h  WRAP arch/arm64/include/generated/asm/early\_ioremap.h  WRAP arch/arm64/include/generated/asm/emergency-restart.h  WRAP arch/arm64/include/generated/asm/hw\_irq.h  WRAP arch/arm64/include/generated/asm/irq\_regs.h  WRAP arch/arm64/include/generated/asm/kdebug.h  WRAP arch/arm64/include/generated/asm/kmap\_types.h  WRAP arch/arm64/include/generated/asm/local.h  WRAP arch/arm64/include/generated/asm/local64.h  WRAP arch/arm64/include/generated/asm/mcs\_spinlock.h  WRAP arch/arm64/include/generated/asm/mm-arch-hooks.h  WRAP arch/arm64/include/generated/asm/msi.h  WRAP arch/arm64/include/generated/asm/preempt.h  WRAP arch/arm64/include/generated/asm/qrwlock.h  WRAP arch/arm64/include/generated/asm/rwsem.h  WRAP arch/arm64/include/generated/asm/segment.h  WRAP arch/arm64/include/generated/asm/serial.h  WRAP arch/arm64/include/generated/asm/set\_memory.h  WRAP arch/arm64/include/generated/asm/sizes.h  WRAP arch/arm64/include/generated/asm/switch\_to.h  WRAP arch/arm64/include/generated/asm/trace\_clock.h  WRAP arch/arm64/include/generated/asm/unaligned.h  WRAP arch/arm64/include/generated/asm/user.h  WRAP arch/arm64/include/generated/asm/vga.h  UPD include/generated/uapi/linux/version.h  UPD include/generated/utsrelease.h  CC kernel/bounds.s  UPD include/generated/bounds.h  UPD include/generated/timeconst.h  CC arch/arm64/kernel/asm-offsets.s  UPD include/generated/asm-offsets.h  CALL scripts/checksyscalls.sh  LDS arch/arm64/kernel/vdso/vdso.lds  CC arch/arm64/kernel/vdso/gettimeofday.o  AS arch/arm64/kernel/vdso/note.o  AS arch/arm64/kernel/vdso/sigreturn.o  VDSOL arch/arm64/kernel/vdso/vdso.so.dbg  VDSOSYM include/generated/vdso-offsets.h  [root@openEuler build]# |

2、编写任务所述内核模块

**一、参考答案源码**

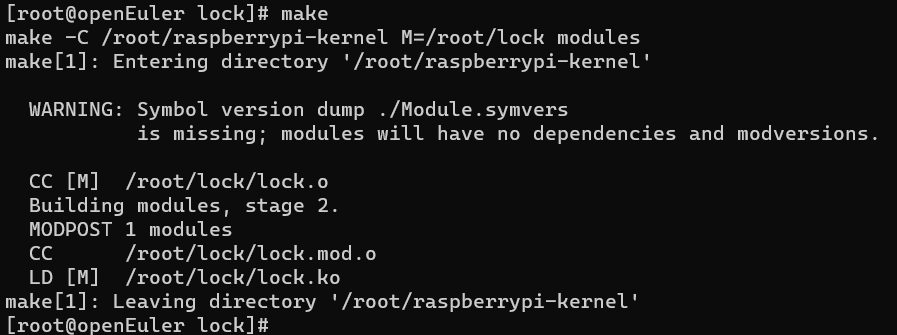
（1）lock.c

|  |
| --- |
| #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <linux/string.h>  #include <linux/slab.h>  #include <linux/delay.h>  #include <linux/kthread.h>  #include <linux/err.h>  static struct task\_struct \*kthread\_test = NULL;  static int myVal = 1;  static int length(int value)  {  int len = 1;  value /= 10;  while (value)  {  len++;  value /= 10;  }  return len;  }  static char\* itos(int value)  {  int len = length(value);  char\* info = (char\*) kmalloc(len + 2, GFP\_ATOMIC);  int index = len - 1;  info[index--] = value % 10 + '0';  value /= 10;  while(value)  {  info[index--] = value % 10 + '0';  value /= 10;  }  info[len] = '\n';  info[len + 1] = '\0';  return info;  }  struct file \* open\_file(char\* filename)  {  struct file \* fp = filp\_open(filename, O\_RDWR | O\_CREAT | O\_APPEND, 0644);  if (IS\_ERR(fp))  {  printk("Create file error\n");  return NULL;  }  return fp;  }  static int write\_file(char\* info, char\* filename)  {  struct file \* fp = open\_file(filename);  kernel\_write(fp, info, strlen(info), &fp->f\_pos);  filp\_close(fp, NULL);  return 0;  }  static int MyPrintk(void \*data)  {  while(!kthread\_should\_stop())  {  char\* info = itos(myVal);  write\_file(info, "/root/lock/output");  kfree(info);  myVal++;  ssleep(1);  }  return 0;  }  static int \_\_init init\_mytask(void)  {  kthread\_test = kthread\_run(&MyPrintk, NULL, "kthreadtest");  return 0;  }  static void \_\_exit exit\_mytask(void)  {  if (kthread\_test)  {  kthread\_stop(kthread\_test);  kthread\_test = NULL;  }  }  module\_init(init\_mytask);  module\_exit(exit\_mytask);  MODULE\_LICENSE("GPL"); |

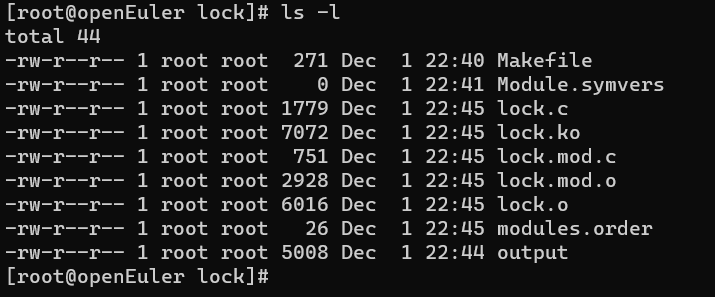
（2）Makefile

|  |
| --- |
| ifneq ($(KERNELRELEASE),)  obj-m :=lock.o  else  KERNELDIR ?=/root/raspberrypi-kernel  PWD := $(shell pwd)  default:  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules  endif  .PHONY:clean  install:  insmod lock.ko  uninstall:  rmmod lock  clean:  -rm \*.mod.c \*.o \*.order \*.symvers \*.ko |

执行make编译源码：



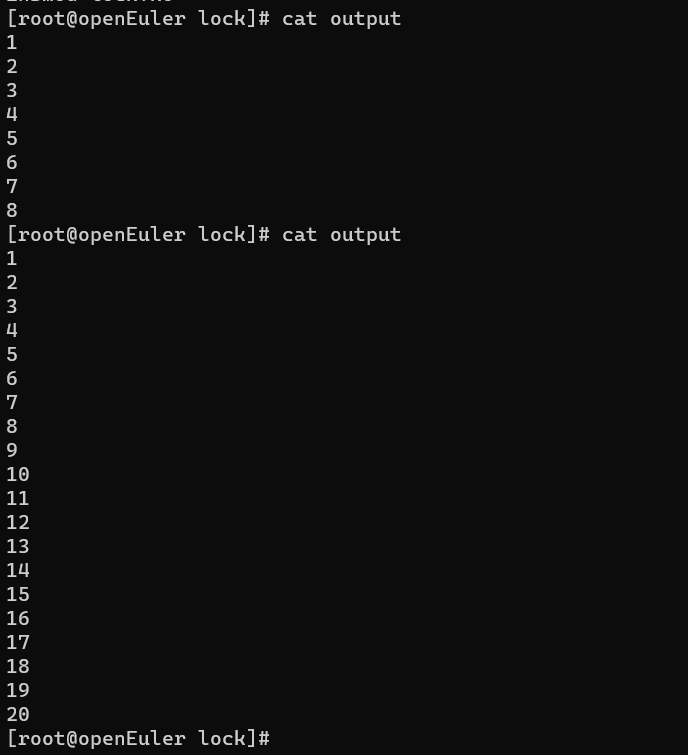
编译完成后的文件列表：

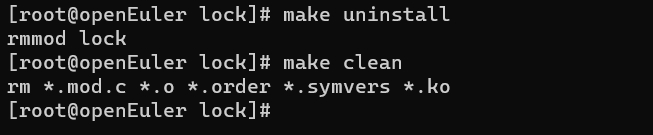


进行模块加载、查看、卸载：

执行加载模块后，会在当前目录创建output文件，output文件会不停打印1、2、3…直到执行卸载模块







一、参考答案源码

# 实训2： 使用自旋锁保护变量(60min)

* 1. **相关知识**

同上

* 1. **任务描述**

1、在实训1的基础上，分别用函数setMyVar和getMyVar实现变量的写和读，并且打印出读和写的具体值。在线程while循环中调用set和get函数，每次对变量加1，可以从output.txt中查看结果。

* 1. 对上文的变量用自旋锁进行保护，调用两个线程对变量进行读取，一个线程从0开始，一个线程从100开始，每次加1，要求在加锁、解锁和获取锁冲突时候都要输出状态。
  2. **审核要求**

1. 正确编写满足功能的源文件，正确编译。

2. 正常加载、卸载内核模块；且内核模块功能满足任务所述。

1. 提交相关源码与运行截图。
   1. **参考答案**

**一、参考答案源码**

1、spinlock.c

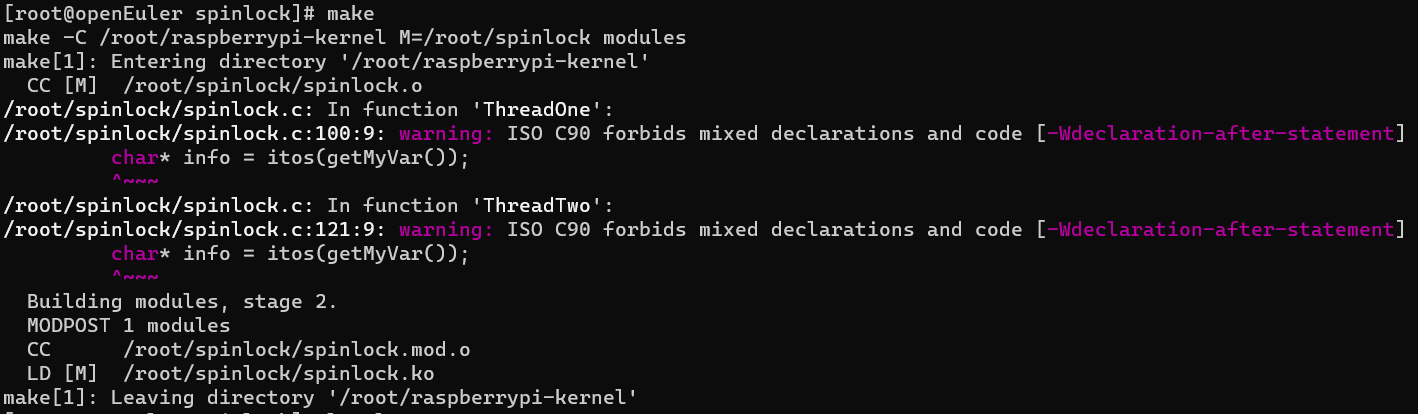
|  |
| --- |
| #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <linux/string.h>  #include <linux/slab.h>  #include <linux/delay.h>  #include <linux/kthread.h>  #include <linux/err.h>  #include <linux/spinlock.h>  #include <linux/sched.h>  static struct task\_struct \*Thread1 = NULL;  static struct task\_struct \*Thread2 = NULL;  static spinlock\_t lock;  static int count = 0;  static int myVar = 0;  void setMyVar(int input){  spin\_lock(&lock);//获得自旋锁,保护临界区  printk("try to lock setmyvar\n");  if(count)  printk("busy setmyvar\n");  //临界区操作  printk("lock setmyvar\n");  count++;  myVar = input;  printk("setmyvar is %d \n",myVar);  //释放自旋锁  spin\_unlock(&lock);  printk("unlock setmyvar\n");  count--;  }  static int getMyVar(void){  int temp;  spin\_lock(&lock);//获得锁  printk("try to lock getmyvar\n");  if(count)  printk("busy getmyvar\n");  printk("lock getmyvar\n");  count++;  temp=myVar;  printk("getmyvar is %d \n",myVar);  spin\_unlock(&lock);  printk("unlock getmyvar\n");  count--;  return temp;  }  static int length(int value)  {  int len = 1;  value /= 10;  while (value)  {  len++;  value /= 10;  }  return len;  }  static char\* itos(int value)  {  int len = length(value);  char\* info = (char\*) kmalloc(len + 2, GFP\_ATOMIC);  int index = len - 1;  info[index--] = value % 10 + '0';  value /= 10;  while(value)  {  info[index--] = value % 10 + '0';  value /= 10;  }  info[len] = '\n';  info[len + 1] = '\0';  return info;  }  struct file \* open\_file(char\* filename)  {  struct file \* fp = filp\_open(filename, O\_RDWR | O\_CREAT | O\_APPEND, 0644);  if (IS\_ERR(fp))  {  printk("Create file error\n");  return NULL;  }  return fp;  }  static int write\_file(char\* info, char\* filename)  {  struct file \* fp = open\_file(filename);  kernel\_write(fp, info, strlen(info), &fp->f\_pos);  filp\_close(fp, NULL);  return 0;  }  static int ThreadOne(void \*data)  {  int i = 100;  while(!kthread\_should\_stop())  {  printk("enter threadone\n");  write\_file("enter threadone\n", "/root/lock/output");  char\* info = itos(getMyVar());  write\_file("Thread one get: ", "/root/lock/output");  write\_file(info, "/root/lock/output");  kfree(info);  setMyVar(i);  info = itos(getMyVar());  write\_file("Thread one set: ", "/root/lock/output");  write\_file(info, "/root/lock/output");  kfree(info);  write\_file("leave threadone\n", "/root/lock/output");  ssleep(1);  i++;  }  return 0;  }  static int ThreadTwo(void \*data)  {  int j = 0;  while(!kthread\_should\_stop())  {  printk("enter threadtwo\n");  char\* info = itos(getMyVar());  write\_file("enter threadtwo\n", "/root/lock/output");  write\_file("Thread two get: ", "/root/lock/output");  write\_file(info, "/root/lock/output");  kfree(info);  setMyVar(j);  info = itos(getMyVar());  write\_file("Thread two set: ", "/root/lock/output");  write\_file(info, "/root/lock/output");  kfree(info);  write\_file("leave threadtwo\n", "/root/lock/output");  ssleep(1);  j++;  }  return 0;  }  static int init\_lock(void )  {  spin\_lock\_init(&lock);  return 0;  }  static int \_\_init init\_mytask(void)  {  init\_lock();  Thread1 = kthread\_run(ThreadOne, NULL, "thread1");  Thread2 = kthread\_run(ThreadTwo, NULL, "thread2");  return 0;  }  static void \_\_exit exit\_mytask(void)  {  if(Thread1){  printk("stop thread1");  kthread\_stop(Thread1);  }  if(Thread2){  printk("stop thread2");  kthread\_stop(Thread2);  }  }  module\_init(init\_mytask);  module\_exit(exit\_mytask);  MODULE\_LICENSE("GPL"); |

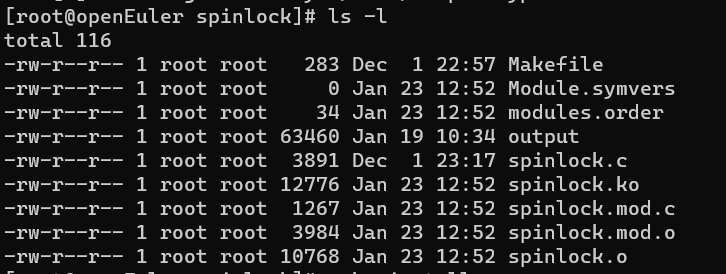
2、Makefile

|  |
| --- |
| ifneq ($(KERNELRELEASE),)  obj-m :=spinlock.o  else  KERNELDIR ?=/root/raspberrypi-kernel  PWD := $(shell pwd)  default:  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules  endif  .PHONY:clean  install:  insmod spinlock.ko  uninstall:  rmmod spinlock  clean:  -rm \*.mod.c \*.o \*.order \*.symvers \*.ko |

二、运行结果

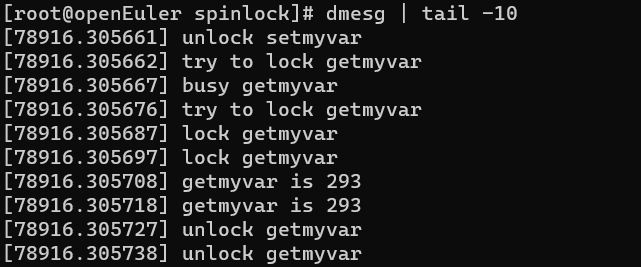
1、加载内核模块，查看文件夹。

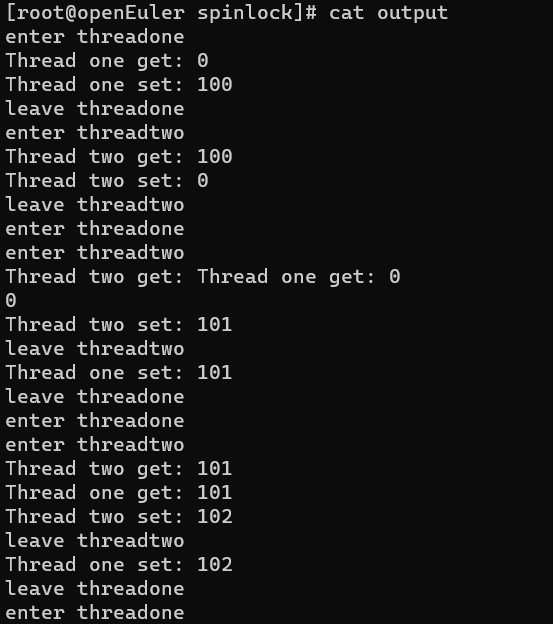


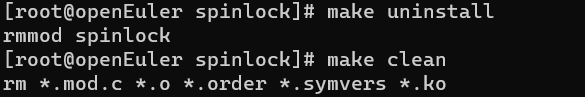


2、进行模块加载、查看、卸载。









# 实训3： 使用顺序锁保护变量(60min)

* 1. **相关知识**

同上

* 1. **任务描述**

1、在实训1的基础上，分别用函数setMyVar和getMyVar实现变量的写和读，并且打印出读和写的具体值。在线程while循环中调用set和get函数，每次对变量加1，可以从output.txt中查看结果。

2、对上文的变量用顺序锁进行保护，调用两个线程对变量进行读取，一个线程从0开始，一个线程从100开始，每次加1，要求在加锁、解锁和获取锁冲突时候都要输出状态。

* 1. **审核要求**

1. 正确编写满足功能的源文件，正确编译。

2. 正常加载、卸载内核模块；且内核模块功能满足任务所述。

1. 提交相关源码与运行截图。

## 参考答案

一、参考答案源码

1、seqlock.c

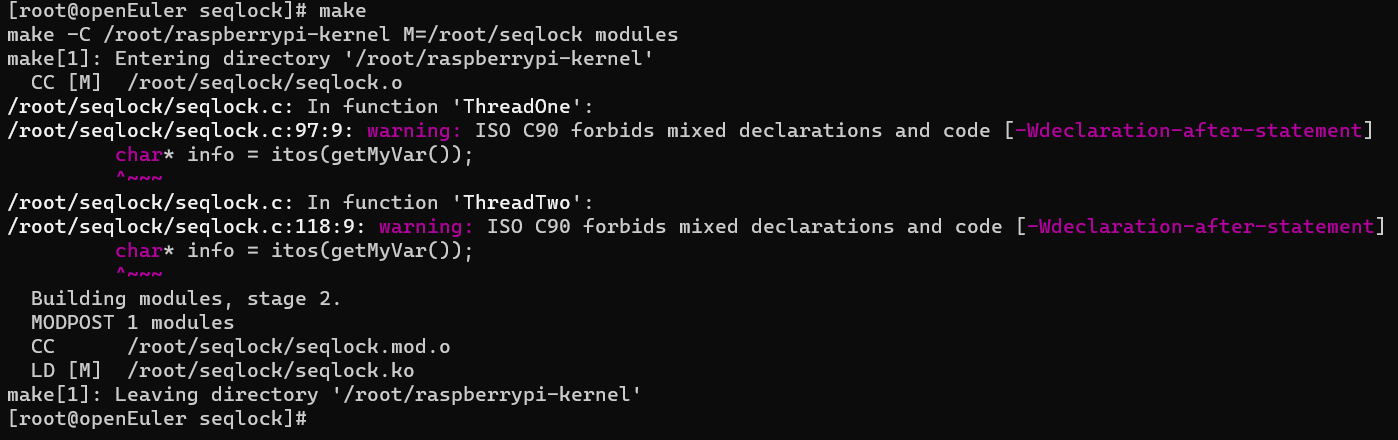
|  |
| --- |
| #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <linux/string.h>  #include <linux/slab.h>  #include <linux/delay.h>  #include <linux/kthread.h>  #include <linux/err.h>  #include <linux/spinlock.h>  #include <linux/sched.h>  #include <linux/seqlock.h>  static struct task\_struct \*Thread1 = NULL;  static struct task\_struct \*Thread2 = NULL;  static seqlock\_t lock;  static int count = 0;  static int myVar = 0;  static void setMyVar(int input){  write\_seqlock(&lock);  printk("try to lock setmyvar\n");  if(count){  printk("busy setmyvar\n");  }  //临界区  printk("lock setmyvar\n");  count++;  myVar = input;  printk("setmyvar is %d \n",myVar);    write\_sequnlock(&lock);  printk("unlock setmyvar\n");  count--;  }  static int getMyVar(void){  unsigned long seq;  int ret;  do{  seq= read\_seqbegin(&lock);  ret=myVar;  }while(read\_seqretry(&lock,seq));  return ret;  }  static int length(int value)  {  int len = 1;  value /= 10;  while (value)  {  len++;  value /= 10;  }  return len;  }  static char\* itos(int value)  {  int len = length(value);  char\* info = (char\*) kmalloc(len + 2, GFP\_ATOMIC);  int index = len - 1;  info[index--] = value % 10 + '0';  value /= 10;  while(value)  {  info[index--] = value % 10 + '0';  value /= 10;  }  info[len] = '\n';  info[len + 1] = '\0';  return info;  }  struct file \* open\_file(char\* filename)  {  struct file \* fp = filp\_open(filename, O\_RDWR | O\_CREAT | O\_APPEND, 0644);  if (IS\_ERR(fp))  {  printk("Create file error\n");  return NULL;  }  return fp;  }  static int write\_file(char\* info, char\* filename)  {  struct file \* fp = open\_file(filename);  kernel\_write(fp, info, strlen(info), &fp->f\_pos);  filp\_close(fp, NULL);  return 0;  }  static int ThreadOne(void \*data)  {  int i = 100;  while(!kthread\_should\_stop())  {  printk("enter threadone\n");  write\_file("enter threadone\n", "/root/seqlock/output");  char\* info = itos(getMyVar());  write\_file("Thread one get: ", "/root/seqlock/output");  write\_file(info, "/root/seqlock/output");  kfree(info);  setMyVar(i);  info = itos(getMyVar());  write\_file("Thread one set: ", "/root/seqlock/output");  write\_file(info, "/root/seqlock/output");  kfree(info);  write\_file("leave threadone\n", "/root/seqlock/output");  ssleep(1);  i++;  }  return 0;  }  static int ThreadTwo(void \*data)  {  int j = 0;  while(!kthread\_should\_stop())  {  printk("enter threadtwo\n");  char\* info = itos(getMyVar());  write\_file("enter threadtwo\n", "/root/spinlock/output");  write\_file("Thread two get: ", "/root/spinlock/output");  write\_file(info, "/root/spinlock/output");  kfree(info);  setMyVar(j);  info = itos(getMyVar());  write\_file("Thread two set: ", "/root/spinlock/output");  write\_file(info, "/root/spinlock/output");  kfree(info);  write\_file("leave threadtwo\n", "/root/spinlock/output");  ssleep(1);  j++;  }  return 0;  }  int init\_lock(void ){  seqlock\_init(&lock);  return 0;  }  static int \_\_init init\_mytask(void)  {  init\_lock();  Thread1 = kthread\_run(ThreadOne, NULL, "thread1");  Thread2 = kthread\_run(ThreadTwo, NULL, "thread2");  return 0;  }  static void \_\_exit exit\_mytask(void)  {  if(Thread1){  printk("stop thread1");  kthread\_stop(Thread1);  }  if(Thread2){  printk("stop thread2");  kthread\_stop(Thread2);  }  }  module\_init(init\_mytask);  module\_exit(exit\_mytask);  MODULE\_LICENSE("GPL"); |

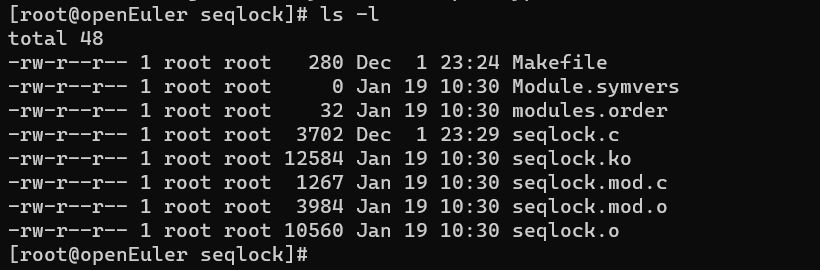
2、Makefile

|  |
| --- |
| ifneq ($(KERNELRELEASE),)  obj-m :=seqlock.o  else  KERNELDIR ?=/root/raspberrypi-kernel  PWD := $(shell pwd)  default:  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules  endif  .PHONY:clean  install:  insmod seqlock.ko  uninstall:  rmmod seqlock  clean:  -rm \*.mod.c \*.o \*.order \*.symvers \*.ko |

二、运行结果

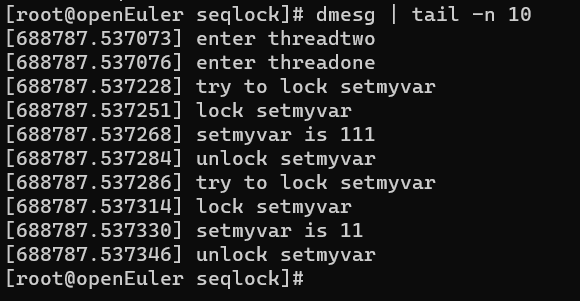
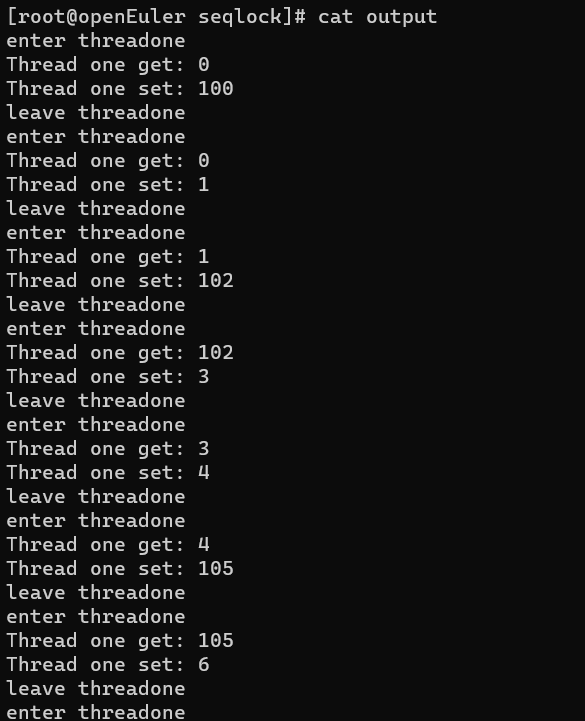
1、加载内核模块，查看文件夹。

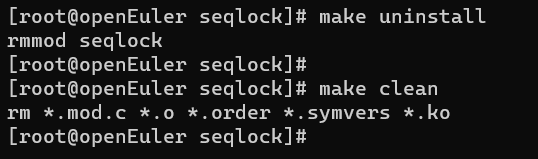




2、进行模块加载、查看、卸载。





1. **实训4： 使用读写锁保护变量(60min)**
   1. **相关知识**

同上

* 1. **任务描述**

1、在实训1的基础上，分别用函数setMyVar和getMyVar实现变量的写和读，并且打印出读和写的具体值。在线程while循环中调用set和get函数，每次对变量加1，可以从output.txt中查看结果。

* 1. 对上文的变量用读写锁进行保护，调用两个线程对变量进行读取，一个线程从0开始，一个线程从100开始，每次加1，要求在加锁、解锁和获取锁冲突时候都要输出状态。
  2. **审核要求**

1. 正确编写满足功能的源文件，正确编译。

2. 正常加载、卸载内核模块；且内核模块功能满足任务所述。

1. 提交相关源码与运行截图。
   1. **参考答案**

一、参考代码

1、rwlock.c

|  |
| --- |
| #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <linux/string.h>  #include <linux/slab.h>  #include <linux/delay.h>  #include <linux/kthread.h>  #include <linux/err.h>  #include <linux/spinlock.h>  #include <linux/sched.h>  #include <linux/seqlock.h>  static struct task\_struct \*Thread1 = NULL;  static struct task\_struct \*Thread2 = NULL;  static rwlock\_t lock;  static int count = 0;  static int myVar = 0;  static void setMyVar(int input){  write\_lock(&lock);//获得自旋锁,保护临界区  printk("try to lock setmyvar\n");  if(count){  printk("busy setmyvar\n");  }  //临界区  printk("lock setmyvar\n");  count++;  myVar = input;  printk("setmyvar is %d \n",myVar);  //释放自旋锁  write\_unlock(&lock);  printk("unlock setmyvar\n");  count--;  }  static int getMyVar(void){  int temp=0;  read\_lock(&lock);//获得锁  printk("try to lock getmyvar\n");  if(count){  printk("busy getmyvar\n");  }  printk("lock getmyvar\n");  count++;  temp=myVar;  printk("getmyvar is %d \n",myVar);  read\_unlock(&lock);  printk("unlock getmyvar\n");  count--;  return temp;  }  static int length(int value)  {  int len = 1;  value /= 10;  while (value)  {  len++;  value /= 10;  }  return len;  }  static char\* itos(int value)  {  int len = length(value);  char\* info = (char\*) kmalloc(len + 2, GFP\_ATOMIC);  int index = len - 1;  info[index--] = value % 10 + '0';  value /= 10;  while(value)  {  info[index--] = value % 10 + '0';  value /= 10;  }  info[len] = '\n';  info[len + 1] = '\0';  return info;  }  struct file \* open\_file(char\* filename)  {  struct file \* fp = filp\_open(filename, O\_RDWR | O\_CREAT | O\_APPEND, 0644);  if (IS\_ERR(fp))  {  printk("Create file error\n");  return NULL;  }  return fp;  }  static int write\_file(char\* info, char\* filename)  {  struct file \* fp = open\_file(filename);  kernel\_write(fp, info, strlen(info), &fp->f\_pos);  filp\_close(fp, NULL);  return 0;  }  static int ThreadOne(void \*data)  {  int i = 100;  while(!kthread\_should\_stop())  {  printk("enter threadone\n");  write\_file("enter threadone\n", "/root/rwspin/output");  char\* info = itos(getMyVar());  write\_file("Thread one get: ", "/root/rwspin/output");  write\_file(info, "/root/rwspin/output");  kfree(info);  setMyVar(i);  info = itos(getMyVar());  write\_file("Thread one set: ", "/root/rwspin/output");  write\_file(info, "/root/rwspin/output");  kfree(info);  write\_file("leave threadone\n", "/root/rwspin/output");  ssleep(1);  i++;  }  return 0;  }  static int ThreadTwo(void \*data)  {  int j = 0;  while(!kthread\_should\_stop())  {  printk("enter threadtwo\n");  char\* info = itos(getMyVar());  write\_file("enter threadtwo\n", "/root/rwspin/output");  write\_file("Thread two get: ", "/root/rwspin/output");  write\_file(info, "/root/rwspin/output");  kfree(info);  setMyVar(j);  info = itos(getMyVar());  write\_file("Thread two set: ", "/root/rwspin/output");  write\_file(info, "/root/rwspin/output");  kfree(info);  write\_file("leave threadtwo\n", "/root/rwspin/output");  ssleep(1);  j++;  }  return 0;  }  int init\_lock(void ){  rwlock\_init(&lock); //动态初始化读写自旋锁  return 0;  }  static int \_\_init init\_mytask(void)  {  init\_lock();  Thread1 = kthread\_run(ThreadOne, NULL, "thread1");  Thread2 = kthread\_run(ThreadTwo, NULL, "thread2");  return 0;  }  static void \_\_exit exit\_mytask(void)  {  if(Thread1){  printk("stop thread1");  kthread\_stop(Thread1);  }  if(Thread2){  printk("stop thread2");  kthread\_stop(Thread2);  }  }  module\_init(init\_mytask);  module\_exit(exit\_mytask);  MODULE\_LICENSE("GPL"); |

2、Makefile

|  |
| --- |
| ifneq ($(KERNELRELEASE),)  obj-m :=rwspin.o  else  KERNELDIR ?=/root/raspberrypi-kernel  PWD := $(shell pwd)  default:  $(MAKE) -C $(KERNELDIR) M=$(PWD) modules  endif  .PHONY:clean  install:  insmod rwspin.ko  uninstall:  rmmod rwspin  clean:  -rm \*.mod.c \*.o \*.order \*.symvers \*.ko |

二、运行结果

