```c
/*
 *    ENSICAEN
 *    LLD
 */

// Defining __KERNEL__ and MODULE allows us to access kernel-level code not usually
available to userspace programs.
#undef __KERNEL__
#define __KERNEL__
#undef MODULE
#define MODULE
#include <linux/module.h>    // included for all kernel modules
#include <linux/kernel.h>    // included for KERN_INFO
#include <linux/init.h>      // included for __init and __exit macros when compiled in
kernel
#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/interrupt.h>
#include <linux/ioctl.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include "tuxit.h"

static struct cdev char_dev;
dev_t dev_num;
struct {
   unsigned char scancode;
   int count;
   int stat[256];
   unsigned char *buffer;
} logger;

/* Spin lock */
DEFINE_SPINLOCK(mr_lock);
/* Tasklets for BH */
void keyboard_tasklet_bh(unsigned long);
DECLARE_TASKLET(keyboard_tasklet, keyboard_tasklet_bh, 0);


static int param_verbose=0;
module_param_named(verbose,param_verbose,int,S_IRUGO);
MODULE_PARM_DESC(verbose, "0: silent, 1: verbose");

void keyboard_tasklet_bh(unsigned long hits) {
   u8 scode;

   if (logger.count < LOGGER_SIZE) {
      /* save the scan code */
      spin_lock(&mr_lock);
      scode = logger.scancode;
      spin_unlock(&mr_lock);

      /* stat */
      logger.count++;
      logger.stat[scode]++;
   }

   return;
}

/* This function services keyboard interrupts */
irq_handler_t irq_handler (int irq, void *dev_id, struct pt_regs *regs) {

   /*
```

```c
    * Read keyboard status.
    * Obtain a spin lock and update scancode.
    */
   spin_lock(&mr_lock);
   logger.scancode = inb (0x60);
   spin_unlock(&mr_lock);

   /* schedule the tasklet */
   tasklet_schedule(&keyboard_tasklet);

   return (irq_handler_t) IRQ_HANDLED;
}

static int cdev_open(struct inode *pInode, struct file *pFile)
{
   printk(KERN_INFO "DBG %s\n",__func__);
   return 0;
}

static int cdev_release(struct inode *pInode, struct file *pFile)
{
   return 0;
}

static int cdev_read(struct file *fp, char __user *buffer, size_t len, loff_t *offset)
{
   printk(KERN_INFO "DBG %s got %d event\n",__func__,logger.count);

   if (len > logger.count)
      len = logger.count;

   if (len) {
      sprintf(logger.buffer, "%d events\n", len);
      copy_to_user(buffer, logger.buffer, strlen(logger.buffer));
      logger.count = 0;
      return strlen(logger.buffer);
   }
   else {
      return 0;
   }
}

static struct file_operations cdev_fops = {
 owner:     THIS_MODULE,
 open:      cdev_open,
 release:   cdev_release,
 read:      cdev_read
};


static int tuxit_init(void)
{
   int result = -1;
   int err=0;

   err = alloc_chrdev_region(&dev_num, 0, 1, "EnsiCaenModule");
   if (err){
      goto init_out;
   }

   if (param_verbose) {
      printk(KERN_INFO "EnsiCaen-3A Init (%d,%d)\n",MAJOR(dev_num), MINOR(dev_num));
      printk(KERN_INFO "compiled: %s %s\n",__DATE__, __TIME__);
   }
   cdev_init(&char_dev, &cdev_fops);
   char_dev.owner = THIS_MODULE;

   err = cdev_add(&char_dev, dev_num, 1);
   if (err){
      cdev_del(&char_dev);
      unregister_chrdev_region(dev_num, 1);
      goto init_out;
   }

   /* Request IRQ 1, the keyboard IRQ */
   result = request_irq (1, (irq_handler_t) irq_handler, IRQF_SHARED, "keyboard_stats",
(void *)(irq_handler));
   if (result)
      printk(KERN_INFO "can't get shared interrupt for keyboard\n");

   /* init scan logger */
   logger.count = 0;
   logger.buffer = kmalloc(GFP_KERNEL, LOGGER_SIZE);

   printk(KERN_INFO "OK\n");
   return 0;    // Non-zero return means that the module couldn't be loaded.

 init_out:
   printk(KERN_INFO "error:%d %s\n",err,ERR_TO_STR(err));
   return err;
}

static void tuxit_cleanup(void)
{
   if (param_verbose) {
      printk(KERN_INFO "EnsiCaen-3A Exit (%d,%d)\n", MAJOR(dev_num), MINOR(dev_num));
   }

   free_irq(1, (void *)(irq_handler));

   /* unregister cdevice */
   cdev_del(&char_dev);
   unregister_chrdev_region(dev_num, 1);
}

module_init(tuxit_init);
module_exit(tuxit_cleanup);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("www.ensicaen.com");
MODULE_DESCRIPTION("tuxit" " 3A LDD training");
```