Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 9

Дисциплина Операционные системы.
Тема Обработчики прерываний.

Студент Сиденко А.Г.
Группа ИУ7-63Б
Оценка (баллы)
Преподаватель Рязанова Н.Ю.

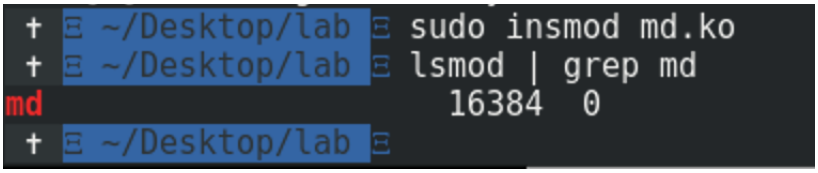
Москва, 2020 г.

Тасклеты

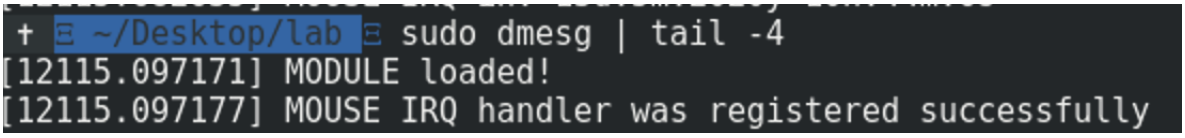
```c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/interrupt.h>
#include <linux/time.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Sidenko");
MODULE_DESCRIPTION("Lab9");

#define MOUSEIRQ 12

char my_tasklet_data[] = "MOUSE_IRQ";

// Bottom Half Function
void my_tasklet_function(unsigned long data)
{
  struct timeval t;
  struct tm broken;
  do_gettimeofday(&t);
  time_to_tm(t.tv_sec, 0, &broken);
  printk("%s in: %dd.%dm.%dy %dh:%dm:%ds\n", (char *)data,
                        broken.tm_mday, broken.tm_mon + 1,
                        broken.tm_year + 1900, broken.tm_hour + 3,
                        broken.tm_min, broken.tm_sec);
}

// Регистрация тасклета
DECLARE_TASKLET(my_tasklet, my_tasklet_function, (unsigned long)
                                                  &my_tasklet_data);

// Обработчик прерывания
irqreturn_t irq_handler(int irq, void *dev, struct pt_regs *regs)
{
  // Проверка, что произошло именно нужное 12-е прерывание
  if(irq == MOUSEIRQ)
  {
    // Постановка тасклета в очередь на выполнение
    tasklet_schedule(&my_tasklet);
    return IRQ_HANDLED; // прерывание обработано
  }
  else
    return IRQ_NONE; // прерывание не обработано
}

// Инициализация модуля
static int __init my_module_init(void)
{
  printk(KERN_DEBUG "MODULE loaded!\n");
  // Разделение(совместное использование) линии IRQ с другими устройствами
```

```
50    int ret = request_irq(MOUSEIRQ, (irq_handler_t)irq_handler, IRQF_SHARED,
51                          "my_irq_handler", (void *)(irq_handler));
52    if (ret != 0)
53    {
54      printk(KERN_ERR "MOUSE_IRQ_handler_wasn't_registered");
55      return ret;
56    }
57    printk(KERN_INFO "MOUSE_IRQ_handler_was_registered_successfully");
58    return ret;
59  }
60
61  // Выход загружаемого модуля
62  static void __exit my_module_exit(void)
63  {
64    // Освобождение линии прерывания
65    free_irq(MOUSEIRQ, (void *)(irq_handler));
66    // Удаление тасклета
67    tasklet_disable(&my_tasklet);
68    tasklet_kill(&my_tasklet);
69    printk(KERN_DEBUG "MODULE_unloaded!\n");
70  }
71
72  module_init(my_module_init);
73  module_exit(my_module_exit);
```

1. Загрузим модуль ядра и проверим в списке загруженных модулей ядра



2. Вывод буфера сообщений ядра в стандартный поток вывода



3. Посмотрим содержимое файла /proc/interrupts, который предоставляет таблицу о количестве прерываний на каждом из процессоров

4. Вывод буфера сообщений ядра в стандартный поток вывода, смотрим на обработку прерываний от мыши

```
 + ~/Desktop/lab  sudo dmesg | tail -8
[12353.359712] MODULE loaded!
[12353.359718] MOUSE IRQ handler was registered successfully
[12354.655226] MOUSE IRQ in: 13d.5m.2020y 10h:48m:5s
[12354.742889] MOUSE IRQ in: 13d.5m.2020y 10h:48m:5s
[12356.767153] MOUSE IRQ in: 13d.5m.2020y 10h:48m:7s
[12356.847626] MOUSE IRQ in: 13d.5m.2020y 10h:48m:8s
[12373.578828] MOUSE IRQ in: 13d.5m.2020y 10h:48m:24s
[12373.673005] MOUSE IRQ in: 13d.5m.2020y 10h:48m:24s
```

5. Выгружаем модуль ядра и выводим буфер сообщений ядра

```
 + ~/Desktop/lab  sudo rmmod md
 + ~/Desktop/lab  sudo dmesg | tail -8
[12356.767153] MOUSE IRQ in: 13d.5m.2020y 10h:48m:7s
[12356.847626] MOUSE IRQ in: 13d.5m.2020y 10h:48m:8s
[12373.578828] MOUSE IRQ in: 13d.5m.2020y 10h:48m:24s
[12373.673005] MOUSE IRQ in: 13d.5m.2020y 10h:48m:24s
[12404.737099] MOUSE IRQ in: 13d.5m.2020y 10h:48m:55s
[12404.750830] MOUSE IRQ in: 13d.5m.2020y 10h:48m:55s
[12431.814622] MOUSE IRQ in: 13d.5m.2020y 10h:49m:23s
[12435.131142] MODULE unloaded!
```

Очереди работ

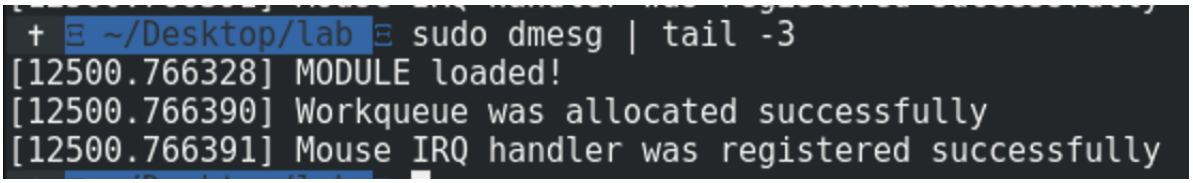
```
1  #include <linux/kernel.h>
2  #include <linux/module.h>
3  #include <linux/interrupt.h>
4  #include <linux/time.h>
5  #include <linux/workqueue.h>
6  #include <linux/slab.h>
7
8  MODULE_LICENSE("GPL");
9  MODULE_AUTHOR("Sidenko");
10 MODULE_DESCRIPTION("Lab9");
11
12 #define MOUSEIRQ 12
13
14 char my_workqueue_data[] = "MOUSE_IRQ";
15 // Очередь работ
16 static struct workqueue_struct *my_wq;
17 struct work_struct *my_work;
18
19 // Bottom Half Function
20 void my_workqueue_function(struct work_struct *my_work)
21 {
22   struct timeval t;
23   struct tm broken;
24   do_gettimeofday(&t);
25   time_to_tm(t.tv_sec, 0, &broken);
26   printk("%s in: %dd.%dm.%dy %dh:%dm:%ds\n", (char *)my_workqueue_data,
```

```
27                              broken.tm_mday, broken.tm_mon + 1,
28                              broken.tm_year + 1900, broken.tm_hour + 3,
29                              broken.tm_min, broken.tm_sec);
30    kfree(my_work);
31  }
32
33  irqreturn_t irq_handler(int irq, void *dev, struct pt_regs *regs)
34  {
35    // Проверка, что произошло именно нужное 12-е прерывание
36    if(irq == MOUSEIRQ)
37    {
38      my_work = (struct work_struct*)kmalloc(sizeof(struct work_struct),
39                                                          GFP_KERNEL);
40      if (my_work)
41      {
42          INIT_WORK(my_work, my_workqueue_function);
43          queue_work(my_wq, my_work);
44      }
45
46      return IRQ_HANDLED; // прерывание обработано
47    }
48    else
49      return IRQ_NONE; // прерывание не обработано
50  }
51
52  // Инициализация модуля
53  static int __init my_module_init(void)
54  {
55    printk(KERN_DEBUG "MODULE loaded!\n");
56    // Разделение(совместное использование) линии IRQ с другими устройствами
57    int ret = request_irq(MOUSEIRQ, (irq_handler_t)irq_handler, IRQF_SHARED,
58                                    "my_irq_handler", (void *)(irq_handler));
59    if (ret != 0)
60    {
61      printk(KERN_ERR "Mouse IRQ handler wasn't registered");
62      return -ENOMEM;
63    }
64
65    // Создание очереди работ
66    my_wq = create_workqueue("my_queue");
67    if (my_wq)
68      printk(KERN_INFO "Workqueue was allocated successfully");
69    else
70    {
71      free_irq(MOUSEIRQ, (void *)(irq_handler));
72      printk(KERN_ERR "Workqueue wasn't allocated");
73      return -ENOMEM;
74    }
75
76    printk(KERN_INFO "Mouse IRQ handler was registered successfully");
77    return ret;
78  }
```

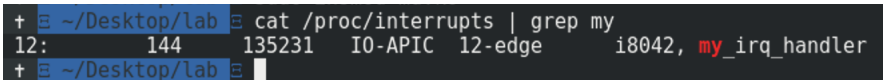
```
79
80 // Выход загружаемого модуля
81 static void __exit my_module_exit(void)
82 {
83     // Освобождение линии прерывания
84     free_irq(MOUSEIRQ, (void *)(irq_handler));
85     // Удаление очереди работ
86     flush_workqueue(my_wq);
87     destroy_workqueue(my_wq);
88     printk(KERN_DEBUG "MODULE unloaded!\n");
89 }
90
91 module_init(my_module_init);
92 module_exit(my_module_exit);
```

1. Загрузим модуль ядра и проверим в списке загруженных модулей ядра

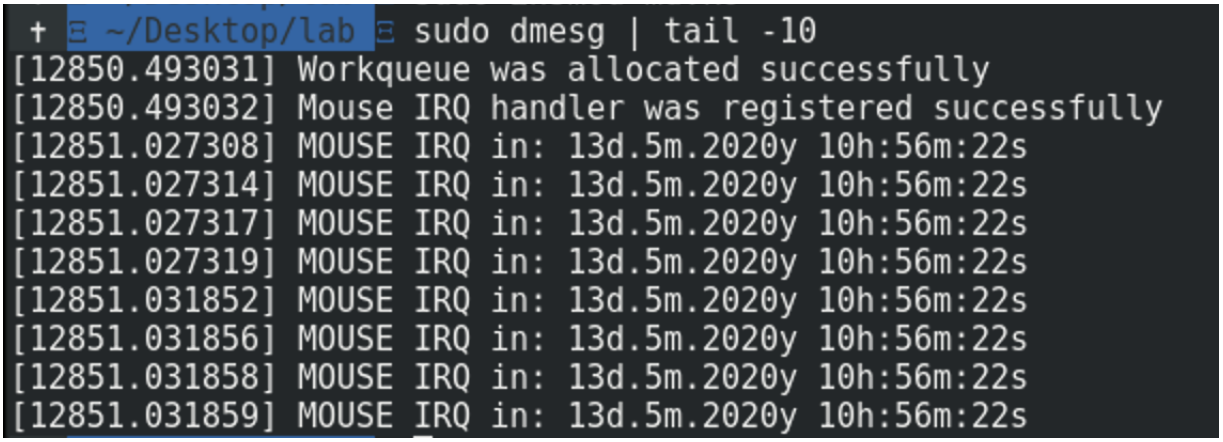
2. Вывод буфера сообщений ядра в стандартный поток вывода



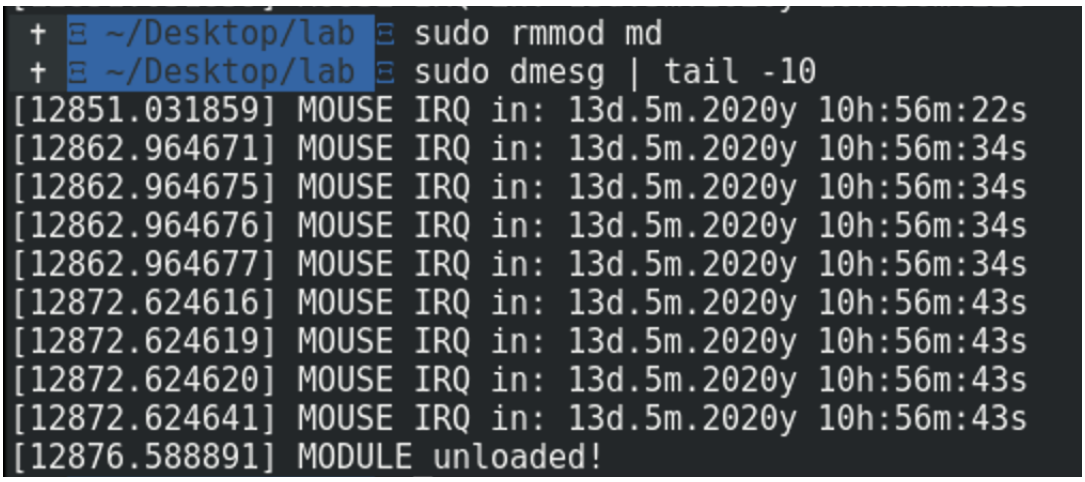
3. Посмотрим содержимое файла /proc/interrupts, который предоставляет таблицу о количестве прерываний на каждом из процессоров



4. Вывод буфера сообщений ядра в стандартный поток вывода, смотрим на обработку прерываний от мыши



5. Выгружаем модуль ядра и выводим буфер сообщений ядра



Makefile

```
ifneq ($(KERNELRELEASE),)
        obj-m := md.o
else
        CURRENT = $(shell uname -r)
        KDIR = /lib/modules/$(CURRENT)/build
        PWD = $(shell pwd)

default:
        $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
        rm -rf .tmp_versions
        rm *.ko
        rm *.o
        rm *.mod.c
        rm *.symvers
        rm *.order

endif
```