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

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

Лабораторная работа № 9
Тема: Обработчики прерываний
Дисциплина: Операционные системы

Студент Лучина Е.Д

Группа ИУ7-61Б

Преподаватель Рязанова Н.Ю.

Москва.
2020 г.

Задание1:

- Написать загружаемый модуль ядра, в котором зарегистрировать обработчик аппаратного прерывания с флагом IRQF_SHARED.
- Инициализировать тасклет.
- В обработчике прерывания запланировать тасклет на выполнение.
- Вывести информацию о тасклете используя, или printk(), или seq_file interface - <linux/seq_file.h> (Jonathan Corber: http://lwn.net//Articales//driver-porting/).

Листинг программы

tmod.c

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Luchina");

static int my_irq = 1;
static int my_dev_id = 0;

char tasklet_data[] = "Tasklet func was called";
void tasklet_function(unsigned long data);
DECLARE_TASKLET(my_tasklet, tasklet_function, (unsigned
long)&tasklet_data);

void tasklet_function(unsigned long data)
{
	printk(KERN_INFO "TMOD : %s ; my_tasklet.state = %s\n", (char *)data,
(my_tasklet.state) ? "RUN" : "SCHED");
```

```
}

irqreturn_t my_irq_handler(int irq, void *dev)
{
	printk(KERN_INFO "TMOD : IRQ handler was called\n");
	if ((int *)dev != &my_dev_id)
		return IRQ_NONE;
	tasklet_schedule(&my_tasklet);
	printk(KERN_INFO "TMOD : my_tasklet was sheduled\n");
	return IRQ_HANDLED;
}

static int __init tmodule_init(void)
{
	int ret = request_irq(my_irq, my_irq_handler, IRQF_SHARED, "my_dev",
(void *)&my_dev_id);
	if(ret)
	{
		printk(KERN_INFO "TMOD : cannot register interrupt handler\n");
		return ret;
	}
	printk(KERN_INFO "TMOD : interrupt handler was registered\n");
	printk(KERN_INFO "TMOD : module is loaded\n");
	return 0;
}

static void __exit tmodule_exit(void)
{
	tasklet_kill(&my_tasklet);
	printk(KERN_INFO "TMOD : my_tasklet was killed\n");
	free_irq(my_irq, &my_dev_id);
	printk(KERN_INFO "TMOD : interrupt handler was unregistered\n");
	printk(KERN_INFO "TMOD : module is unloaded\n");
}

module_init(tmodule_init);
module_exit(tmodule_exit);
```

Makefile

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

```
    PWD = $(shell pwd)

default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
    @rm -f *.o .*.cmd .*.flags *.mod.c *.order
    @rm -f .*.*.cmd *~ *.*~ TODO.*
    @rm -fR .tmp*
    @rm -rf .tmp_versions

disclean: clean
    @rm *.ko *.symvers

endif
```

Пояснения к коду

1) Подключаем заголовочные файлы
2) Определим лицензию и автора модуля
3) Будем работать с прерыванием IRQ1 - клавиатура. Идентификатор устройства примем равным нулю
4) Инициализируем сообщение для тасклета и объявим функцию обработчик тасклета. Статически зарегистрируем тасклет с помощью макроса DECLARE_TASKLET.
5) Определим функцию-обработчик тасклета. Она будет выводить data(сообщение) и state(статус) тасклета.
6) Определим обработчик прерывания, внутри которого планируется тасклет.
7) Определим функцию инициализации модуля при загрузки. Регистрируем обработчик аппаратного прерывания и разрешаем определенную линию irq посредством функции request_irq. Выводим в журнал соответствующие сообщения.
8) В функции выгрузки модуля функция tasklet_kill ждет завершения тасклета и удаляет уго из очереди на выполнение в контексте процесса. А также free_irq отключает обработчик прерывания.
9) Сообщаем ядру какие функции использовать при загрузки и выгрузки ядра с помощью макросов module_init, module_exit.

Демонстрация работы программы

Файл /proc/interrupts предоставляет таблицу о количестве прерываний на каждом из процессоров в следующем виде:

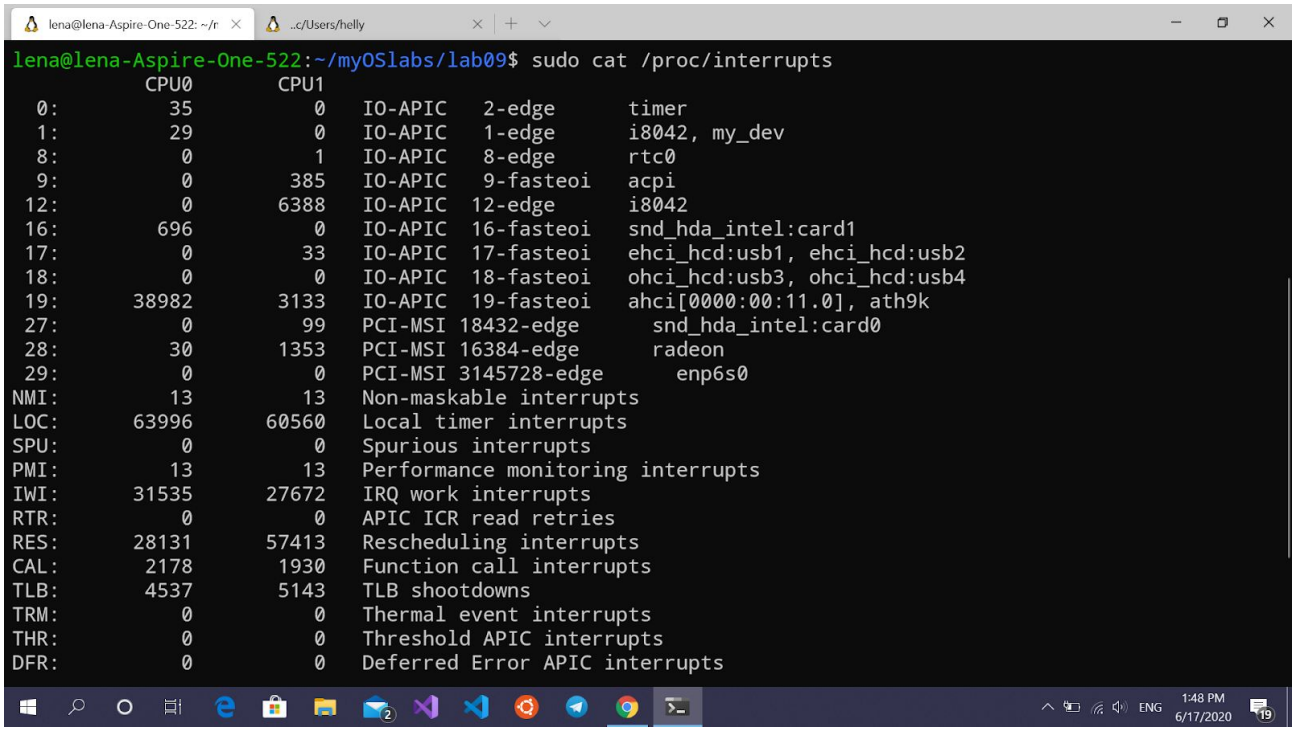
- Первая колонка: номер прерывания
- Колонки CPUx: счётчики прерываний на каждом из процессоров
- Следующая колонка: вид прерывания:
    - IO-APIC-edge — прерывание по фронту на контроллер I/O APIC
    - IO-APIC-fasteoi — прерывание по уровню на контроллер I/O APIC
    - PCI-MSI-edge — MSI прерывание
    - XT-PIC-XT-PIC — прерывание на PIC контроллер
- Последняя колонка: устройство, ассоциированное с данным прерыванием

```
lena@lena-Aspire-One-522:~$ sudo cat /proc/interrupts
[sudo] password for lena:
           CPU0       CPU1
  0:         34          0   IO-APIC   2-edge      timer
  1:          9          0   IO-APIC   1-edge      i8042
  8:          0          1   IO-APIC   8-edge      rtc0
  9:          0        774   IO-APIC   9-fasteoi   acpi
 12:          0        178   IO-APIC  12-edge      i8042
 16:        707          0   IO-APIC  16-fasteoi   snd_hda_intel:card1
 17:          0         33   IO-APIC  17-fasteoi   ehci_hcd:usb1, ehci_hcd:usb2
 18:          0          0   IO-APIC  18-fasteoi   ohci_hcd:usb3, ohci_hcd:usb4
 19:      88997       3389   IO-APIC  19-fasteoi   ahci[0000:00:11.0], ath9k
 27:          0         99   PCI-MSI 18432-edge      snd_hda_intel:card0
 28:          0       1146   PCI-MSI 16384-edge      radeon
 29:          0          0   PCI-MSI 3145728-edge      enp6s0
NMI:         19         20   Non-maskable interrupts
LOC:     113977     102085   Local timer interrupts
SPU:          0          0   Spurious interrupts
```

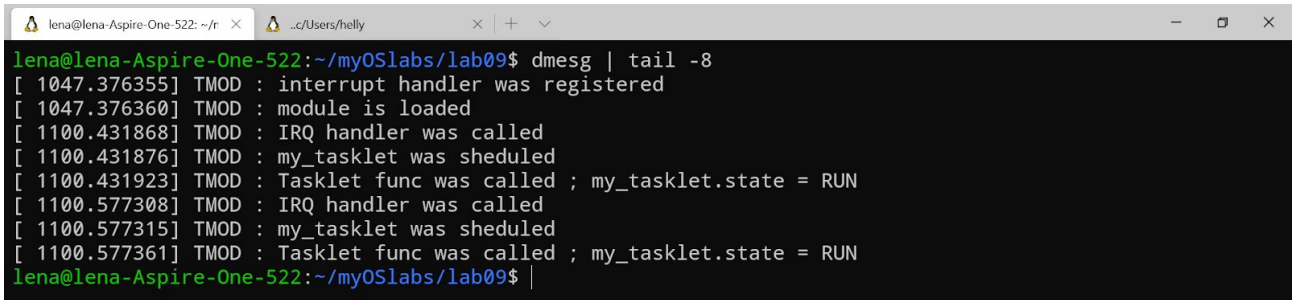
Скомпилируем (make) и загрузим (sudo insmod tmod.ko) модуль. Убедимся в успешности загрузки (lsmod, dmesg). В листинге загруженных модулей ядра увидим tmod, в журнале увидим сообщения из функции tmodule_init().

```
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ ls
Makefile  tmod.c
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ make
make -C /lib/modules/5.3.0-28-generic/build M=/home/lena/myOSlabs/lab09 modules
make[1]: Entering directory '/usr/src/linux-headers-5.3.0-28-generic'
  CC [M]  /home/lena/myOSlabs/lab09/tmod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/lena/myOSlabs/lab09/tmod.mod.o
  LD [M]  /home/lena/myOSlabs/lab09/tmod.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.3.0-28-generic'
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ ls
Makefile  modules.order  Module.symvers  tmod.c  tmod.ko  tmod.mod  tmod.mod.c  tmod.mod.o  tmod.o
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ sudo insmod tmod.ko
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ lsmod | grep tmod
tmod                   16384  0
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ dmesg | tail -2
[ 1047.376355] TMOD : interrupt handler was registered
[ 1047.376360] TMOD : module is loaded
lena@lena-Aspire-One-522:~/myOSlabs/lab09$
```

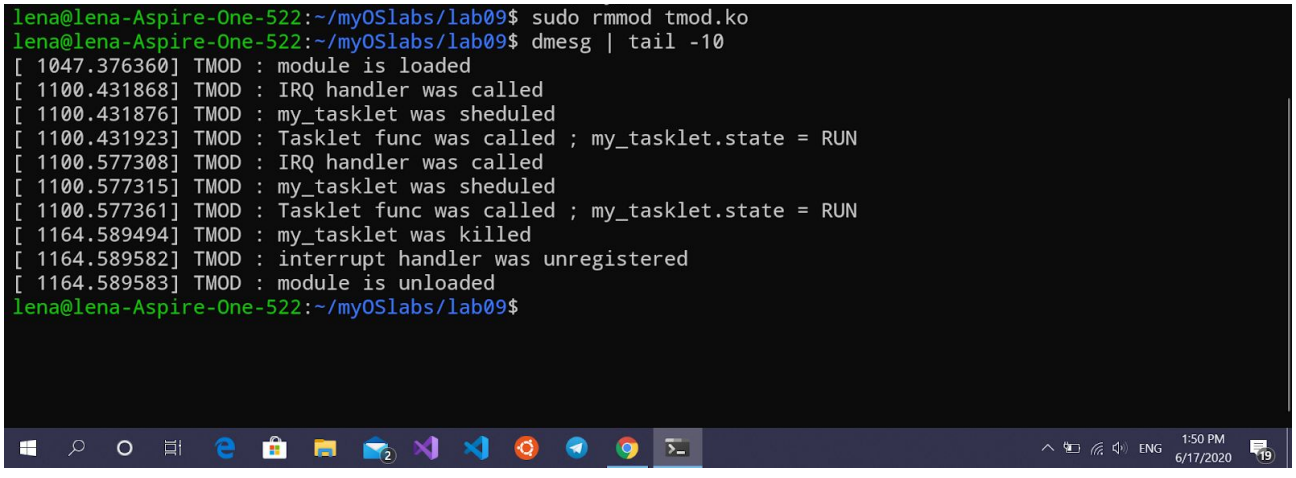
После того, как модуль загружен, в /proc/interrupts среди устройств, ассоциированных с прерыванием 1, найдем “my_dev”.



Если нажать и отпустить, например, пробел, в dmesg увидим два сообщения от обработчика. (Отмечу, что в терминале, представленном на скриншотах, удаленно через ssh подключена машина с ubuntu. Рассматриваемые прерывания клавиатуры - это прерывания с клавиатуры компьютера, на котором запущена ос ubuntu. Печатание команд в данной ситуации не вызывает обработчик прерывания)



Выгрузим модуль с помощью sudo rmmod и dmesg увидим сообщения об успешной выгрузке.



Задание2

- Написать загружаемый модуль ядра, в котором зарегистрировать обработчик аппаратного прерывания с флагом IRQF_SHARED.
- Инициализировать очередь работ.
- В обработчике прерывания запланировать очередь работ на выполнение.
- Вывести информацию об очереди работ используя, или printk(), или seq_file interface - <linux/seq_file.h> (Jonathan Corber: http://lwn.net//Articales//driver-porting/)

Листинг программы

wqmod.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <linux/workqueue.h>
#include <linux/slab.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Luchina");

static int my_irq = 1;
static int my_dev_id = 0;

char w_data[] = "work data";
static struct workqueue_struct *my_wq;

typedef struct
{
	struct work_struct my_work;
	char *text;
} my_work_t;

my_work_t *cur_work;

void my_work_function(struct work_struct *work)
{
	my_work_t *my_work = (my_work_t *)work;
	printk(KERN_INFO "WQMOD : work is processing, data = %s\n",
my_work->text);
	kfree(work);
}
```

```
irqreturn_t my_irq_handler(int irq, void *dev)
{
	int ret;
	printk(KERN_INFO "WQMOD : IRQ handler was called\n");
	if ((int *)dev != &my_dev_id)
		return IRQ_NONE;
	ret = queue_work(my_wq, (struct work_struct *)cur_work);
	if (ret)
		printk(KERN_INFO "WQMOD : work is already on queue\n");
	else
		printk(KERN_INFO "WQMOD : work was scheduled\n");
	return IRQ_HANDLED;
}

static int __init wqmodule_init(void)
{
	int ret = request_irq(my_irq, my_irq_handler, IRQF_SHARED, "my_dev",
(void *)&my_dev_id);
	if(ret)
	{
		printk(KERN_INFO "WQMOD : cannot register interrupt handler\n");
		return ret;
	}
	printk(KERN_INFO "WQMOD : interrupt handler was registered\n");
	my_wq = create_workqueue("my_workqueue");
	if (!my_wq)
	{
		free_irq(my_irq, (void *)&my_dev_id);
		printk(KERN_INFO "WQMOD : cannot create work_queue\n");
		return -1;
	}
	printk(KERN_INFO "WQMOD : work queue was created\n");

	cur_work = (my_work_t *)kmalloc(sizeof(my_work_t), GFP_KERNEL);
	if (!cur_work)
	{
		destroy_workqueue(my_wq);
		free_irq(my_irq, (void *)&my_dev_id);
		printk(KERN_INFO "WQMOD : cannot allocate work item\n");
		return -1;
	}
	INIT_WORK((struct work_struct *)cur_work, my_work_function);
	cur_work->text = w_data;
	printk(KERN_INFO "WQMOD : work item was created\n");
	printk(KERN_INFO "WQMOD : module is loaded\n");
	return 0;
```

```
}

static void __exit wqmodule_exit(void)
{
	kfree((void *)cur_work);
	flush_workqueue(my_wq);
	destroy_workqueue(my_wq);
	printk(KERN_INFO "WQMOD : workqueue was killed\n");
	free_irq(my_irq, &my_dev_id);
	printk(KERN_INFO "WQMOD : interrupt handler was unregistered\n");
	printk(KERN_INFO "WQMOD : module is unloaded\n");
}

module_init(wqmodule_init);
module_exit(wqmodule_exit);
```

Makefile

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

default:
	$(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
	@rm -f *.o .*.cmd .*.flags *.mod.c *.order
	@rm -f .*.*.cmd *~ *.*~ TODO.*
	@rm -fR .tmp*
	@rm -rf .tmp_versions

disclean: clean
	@rm *.ko *.symvers

endif
```

Пояснения к коду

1) Подключаем заголовочные файлы
2) Определим лицензию и автора модуля
3) Будем работать с прерыванием IRQ1 - клавиатура. Идентификатор устройства примем равным нулю

4) Инициализируем сообщение для задачи, объявим очередь работ и структуру my_work_t.
5) Вызываемая функция my_work_function выводит сообщение в журнал и освобождает память из-под объекта struct work_struct.
6) Определим обработчик прерывания, внутри которого добавим объект work в очередь работ с помощью функции queue_work (она назначает работу текущему процессору)
7) Определим функцию инициализации модуля при загрузки. Регистрируем обработчик аппаратного прерывания и разрешим линию irq посредством функции request_irq. С помощью create_workqueue создадим очередь работ. В случае неуспеха отключим обработчик прерывания (free_irq). Иначе выделяем в пространстве ядра память под структуру my_work_t. Если память выделилась успешно, динамически инициализируем структуру для объекта work с помощью макроса INIT_WORK. А при неудаче destroy_workqueue удаляет рабочую очередь, free_iqr отключает обработчик.
8) В функции выгрузки модуля освобождаем память, принудительно завершаем все работы в очереди и удаляем очередь, отключаем обработчик. Сопровождаем эти действия выводом сообщений в журнал.
9) Сообщаем ядру какие функции использовать при загрузки и выгрузки ядра с помощью макросов module_init, module_exit.

Демонстрация работы программы

Скомпилируем модуль и загрузим его. Удостоверимся, что загрузка прошла успешно. В /proc/interrupts увидим что первое прерывание теперь также ассоциируется с устройством my_dev.

```
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ make
make -C /lib/modules/5.3.0-28-generic/build M=/home/lena/myOSlabs/lab09 modules
make[1]: Entering directory '/usr/src/linux-headers-5.3.0-28-generic'
  CC [M]  /home/lena/myOSlabs/lab09/wqmod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/lena/myOSlabs/lab09/wqmod.mod.o
  LD [M]  /home/lena/myOSlabs/lab09/wqmod.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.3.0-28-generic'
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ clear
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ sudo insmod wqmod.ko
[sudo] password for lena:
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ dmesg | tail
[   53.948262] ath: Regpair used: 0x3d
[   53.948266] ath: regdomain 0x8283 dynamically updated by country element
[   54.190877] IPv6: ADDRCONF(NETDEV_CHANGE): wlp7s0: link becomes ready
[   76.671643] rfkill: input handler disabled
[ 2213.040964] wqmod: loading out-of-tree module taints kernel.
[ 2213.041048] wqmod: module verification failed: signature and/or required key missing - tainting kernel
[ 2213.043216] WQMOD : interrupt handler was registered
[ 2213.044378] WQMOD : work queue was created
[ 2213.044387] WQMOD : work item was created
[ 2213.044388] WQMOD : module is loaded
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ sudo cat /proc/interrupts
           CPU0       CPU1
  0:         34          0   IO-APIC   2-edge      timer
  1:         52          0   IO-APIC   1-edge      i8042, my_dev
  8:          0          1   IO-APIC   8-edge      rtc0
  9:          0        612   IO-APIC   9-fasteoi   acpi
```

Проверим работу обработчика прерывания и очереди работ - нажмем на клавиши клавиатуры.

```
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ dmesg | tail
[ 2213.043216] WQMOD : interrupt handler was registered
[ 2213.044378] WQMOD : work queue was created
[ 2213.044387] WQMOD : work item was created
[ 2213.044388] WQMOD : module is loaded
[ 2247.581703] WQMOD : IRQ handler was called
[ 2247.581712] WQMOD : work is already on queue
[ 2247.582392] WQMOD : work is processing, data = work data
[ 2247.727456] WQMOD : IRQ handler was called
[ 2247.727464] WQMOD : work is already on queue
[ 2247.727735] WQMOD : work is processing, data = work data
```

Выгрузим модуль

```
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ sudo rmmod wqmod.ko
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ dmesg | tail
[ 2213.044388] WQMOD : module is loaded
[ 2247.581703] WQMOD : IRQ handler was called
[ 2247.581712] WQMOD : work is already on queue
[ 2247.582392] WQMOD : work is processing, data = work data
[ 2247.727456] WQMOD : IRQ handler was called
[ 2247.727464] WQMOD : work is already on queue
[ 2247.727735] WQMOD : work is processing, data = work data
[ 2293.537750] WQMOD : workqueue was killed
[ 2293.537850] WQMOD : interrupt handler was unregistered
[ 2293.537852] WQMOD : module is unloaded
lena@lena-Aspire-One-522:~/myOSlabs/lab09$ cat /proc/interrupts
           CPU0       CPU1
  0:         34          0   IO-APIC   2-edge      timer
  1:         54          0   IO-APIC   1-edge      i8042
  8:          0          1   IO-APIC   8-edge      rtc0
  9:          0        652   IO-APIC   9-fasteoi   acpi
```