|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **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**

**По дисциплине: “Операционные системы”**

|  |  |
| --- | --- |
| **Тема:** “Обработчики прерываний ”  **Студент:** Мередова Айджахан  **Группа:** ИУ7-66Б  **Преподаватель:** Рязанова Н.Ю. |  |

Москва.

2021 г.

**Задание1:**

**Листинг кода.**

tasklet.c

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/interrupt.h>

#include <linux/time.h>

#include <linux/workqueue.h>

#include <linux/lockdep.h>

#include <linux/mutex.h>

#include <linux/proc\_fs.h>

#include <linux/sched.h>

#include <linux/uaccess.h>

#include <linux/fs.h>

#include <linux/seq\_file.h>

#include <linux/unistd.h>

#include<linux/module.h>

#include<linux/init.h>

#include<linux/proc\_fs.h>

#include<linux/sched.h>

#include<linux/uaccess.h>

#include<linux/fs.h>

#include<linux/seq\_file.h>

#include<linux/slab.h>

#include<linux/vmalloc.h>

#define SHARED\_IRQ 1

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_AUTHOR("Meredova");

static int my\_dev\_id;

char tasklet\_data[] = "tasklet data";

void tasklet\_handler(unsigned long data);

DECLARE\_TASKLET(my\_tasklet, tasklet\_handler, (unsigned long) &tasklet\_data);

static int my\_proc\_show(struct seq\_file \*m, void \*v)

{

// pending - выяснить, приостановлен ли эл-т work (еще не обработан обработчиком))

seq\_printf(m, "=== In user mode tasklet: state - %ld, count - %d, data - %s\n", my\_tasklet.state, my\_tasklet.count, my\_tasklet.data);

return 0;

}

static int my\_proc\_open(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "=== Called my\_proc\_open\n");

return single\_open(file, my\_proc\_show, NULL);

}

static struct file\_operations props={

.open = my\_proc\_open,

.release = single\_release,

.read = seq\_read,

};

void tasklet\_handler(unsigned long data)

{

printk(KERN\_INFO "Tasklet: state - %ld, count - %d, data - %s\n", my\_tasklet.state, my\_tasklet.count, my\_tasklet.data);

}

// Обработчик прерывания

static irqreturn\_t my\_interrupt(int irq, void \*dev\_id)

{

if (irq == SHARED\_IRQ)

{

printk(KERN\_INFO "Tasklet scheduled\n");

tasklet\_schedule(&my\_tasklet);

return IRQ\_HANDLED;

}

else

return IRQ\_NONE;

}

static int \_\_init my\_tasklet\_init(void)

{ struct proc\_dir\_entry \*entry;

entry = proc\_create("tasklet", S\_IRUGO | S\_IWUGO, NULL, &props); if (!entry)

{ printk(KERN\_INFO "=== Error: can't create seq file\n"); return -ENOMEM; } printk(KERN\_INFO "=== Seq file created\n");

if (request\_irq(SHARED\_IRQ, my\_interrupt, IRQF\_SHARED, "my\_interrupt", &my\_dev\_id))

{

printk(KERN\_ERR "Error on request\_irq\n");

return -1;

}

printk(KERN\_INFO "Module loaded!\n");return 0;}

static void \_\_exit my\_tasklet\_exit(void)

{

tasklet\_kill(&my\_tasklet);

free\_irq(SHARED\_IRQ, &my\_dev\_id); // Освобождение линии прерывания

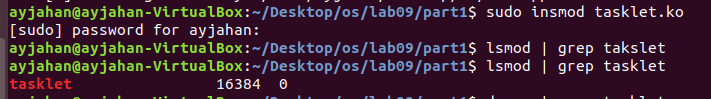
printk(KERN\_INFO "Module unloaded!\n");

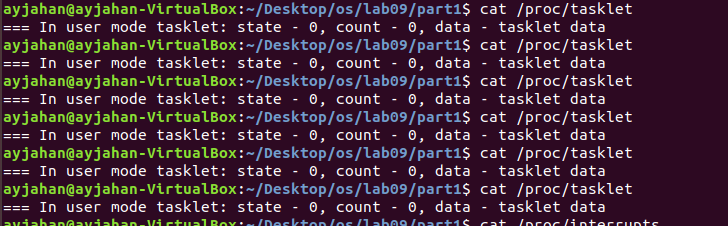
}

module\_init(my\_tasklet\_init);

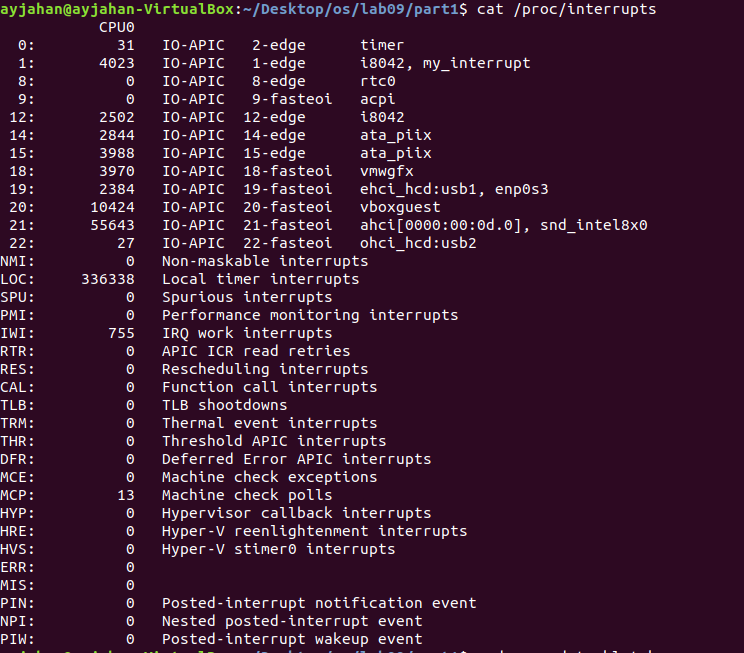
module\_exit(my\_tasklet\_exit);

**Результат работы:**

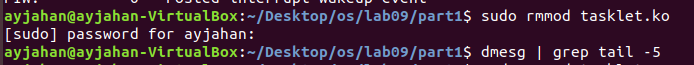


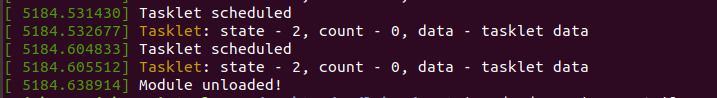


Загрузка модуля



/proc/interrupts





Выгрузка модуля.

**Задание2:**

**Листинг кода:**

wqueue.c

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/interrupt.h>

#include <linux/time.h>

#include <linux/workqueue.h>

#include <linux/lockdep.h>

#include <linux/mutex.h>

#include <linux/proc\_fs.h>

#include <linux/sched.h>

#include <linux/uaccess.h>

#include <linux/fs.h>

#include <linux/seq\_file.h>

#include <linux/unistd.h>

#include<linux/module.h>

#include<linux/init.h>

#include<linux/proc\_fs.h>

#include<linux/sched.h>

#include<linux/uaccess.h>

#include<linux/fs.h>

#include<linux/seq\_file.h>

#include<linux/slab.h>

#include<linux/vmalloc.h>

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Meredova");

#define IRQ\_NUM 1

#define IRQ\_NAME "keyboard"

#define WQ\_NAME "keyboard\_wq"

#define ITER\_COUNT 1000000

// Структура, описывающая очередь работ

static struct workqueue\_struct \*my\_wq;

typedef struct

{

struct work\_struct work;

int work\_num; // номер работы

} my\_work\_t;

// Структуры, описывающие две работы

// (обработчики нижней половины прерывания в очереди работ)

static my\_work\_t \*work\_1;

static my\_work\_t \*work\_2;

// Разделяемая переменная

static int shared\_counter = 0;

// Mutex

struct mutex my\_mutex;

static int my\_proc\_show(struct seq\_file \*m, void \*v)

{

// pending - выяснить, приостановлен ли эл-т work (еще не обработан обработчиком))

seq\_printf(m, "=== In user mode: is work\_1 pending - %d, is work\_2 pending - %d, shared counter - %d\n",

work\_pending(&(work\_1->work)), work\_pending(&(work\_2->work)), shared\_counter);

return 0;

}

static int my\_proc\_open(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "=== Called my\_proc\_open\n");

return single\_open(file, my\_proc\_show, NULL);

}

static struct file\_operations props={

.open = my\_proc\_open,

.release = single\_release,

.read = seq\_read,

};

void my\_bottom\_half(struct work\_struct \*work)

{

// блокирует объект мьютекса, на который ссылается my\_mutex

mutex\_lock(&my\_mutex);

printk(KERN\_INFO "=== MUTEX LOCKED ===\n");

my\_work\_t \*my\_work = (my\_work\_t \*)work;

// какая работа вошла в критческую секцию?

printk(KERN\_INFO "=== worker\_%d, status = %d\n", my\_work->work\_num, work\_pending(&(my\_work->work)));

// другая работа заблокированав это время?

if (my\_work->work\_num == 1) //

printk(KERN\_INFO "=== work\_2, status = %d\n", work\_pending(&(work\_2->work)));

else

printk(KERN\_INFO "=== work\_1, status = %d\n", work\_pending(&(work\_1->work)));

int sum = 0, i = 0;

for (; i < ITER\_COUNT; i++)

sum += i;

shared\_counter++;

printk(KERN\_INFO "Sum: %d\n", sum);

printk(KERN\_INFO "Workqueue: counter %d\n", ++shared\_counter);

mutex\_unlock(&my\_mutex);

printk(KERN\_INFO "=== MUTEX UNLOCKED ===\n");

printk(KERN\_INFO "=== AFTER UNLOCK ===: worker\_%d, status = %d\n", my\_work->work\_num, work\_pending(&(my\_work->work)));

}

// в верхней половине выполняется

// динамическая инициализация и поставновка работ в очередь работ

irqreturn\_t my\_irq\_handler(int irq\_num, void \*dev\_id)

{

// Проверка, что произошло именно нужное 1-е прерывание

if (irq\_num == IRQ\_NUM)

{

if (work\_1)

// Помещаем задачу в очередь работ (назначает работу текущему процессору)

queue\_work(my\_wq, (struct work\_struct \*)work\_1);

if (work\_2)

queue\_work(my\_wq, (struct work\_struct \*)work\_2);

return IRQ\_HANDLED; // прерывание обработано

}

return IRQ\_NONE; // прерывание не обработано

}

static int \_\_init my\_module\_init(void)

{

struct proc\_dir\_entry \*entry;

entry = proc\_create("wqueue", S\_IRUGO | S\_IWUGO, NULL, &props);

if (!entry)

{

printk(KERN\_INFO "=== Error: can't create seq file\n");

return -ENOMEM;

}

printk(KERN\_INFO "=== Seq file created\n");

// номер irq

// указ-ль на наш обработчик прерывания

// флаг разделение(совместное использование) линии IRQ с другими устройствами

// имя устройства, связ. с прерыванием

// идентификатор устройства, нужен для для разделения линии прер-я, отключения с линии прерваний с помощью free\_irq

// === регистрация обработчика прерывания ===

if (request\_irq(IRQ\_NUM, my\_irq\_handler, IRQF\_SHARED, IRQ\_NAME, my\_irq\_handler))

{

printk(KERN\_ERR "=== \*\*ERROR\*\* Failed to register IRQ handler\n");

return -ENOMEM;

}

printk(KERN\_INFO "=== IRQ handler registered\n");

// kmalloc() — выделение в ядре участков памяти размером в заданное количество байтов

work\_1 = (my\_work\_t \*)kmalloc(sizeof(my\_work\_t), GFP\_KERNEL);

work\_2 = (my\_work\_t \*)kmalloc(sizeof(my\_work\_t), GFP\_KERNEL);

if (work\_1)

{

// Инициализация структуры work12 (динамическое задание work\_struct)

// func – функция, которая вызывается из workqueue – обработчик нижней половины

INIT\_WORK((struct work\_struct \*)work\_1, my\_bottom\_half);

work\_1->work\_num = 1;

}

else

{

printk(KERN\_ERR "=== \*\*ERROR\*\* Failed kalloc\n");

return -ENOMEM;

}

if (work\_2)

{

INIT\_WORK((struct work\_struct \*)work\_2, my\_bottom\_half);

work\_2->work\_num = 2;

}

else

{

printk(KERN\_ERR "=== \*\*ERROR\*\* Failed kalloc\n");

}

// ==== Cоздание очереди работ ====

// Флаг UNBOUND - по умолчанию - непривязанная к конкретному процессору очередь

// внутри вызов alloc\_workqueue(name, flags, max\_active)

// name - имя очереди, flags - флаги определяют как очередь работ будет выполняться

// max\_active - ограничивает число задач из данной очереди, которые могут одновременно выполняться на одном CPU.

my\_wq = create\_workqueue(WQ\_NAME);

if (!my\_wq)

{

free\_irq(IRQ\_NUM, my\_irq\_handler);

printk(KERN\_ERR "=== \*\*ERROR\*\* Failed to create work queue\n");

return -ENOMEM;

}

printk(KERN\_INFO "=== Workqueue created\n");

mutex\_init(&my\_mutex);

printk(KERN\_INFO "=== Module loaded!\n");

return 0;

}

// Выход загружаемого модуля

static void \_\_exit my\_module\_exit(void)

{

// Освобождение линии прерывания

free\_irq(IRQ\_NUM, my\_irq\_handler);

// Удаление очереди работ

flush\_workqueue(my\_wq); // Все работы в данной очереди работ принудительно завершены

destroy\_workqueue(my\_wq); // удаление очереди работ

mutex\_destroy(&my\_mutex);

if (work\_1)

kfree(work\_1);

if (work\_2)

kfree(work\_2);

remove\_proc\_entry("wqueue", NULL);

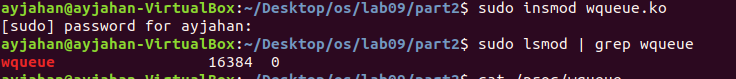
printk(KERN\_INFO "=== Module unloaded!\n");

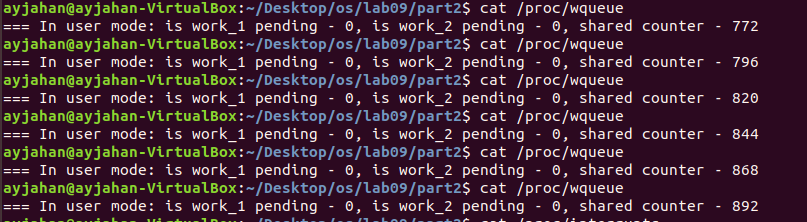
}

module\_init(my\_module\_init);

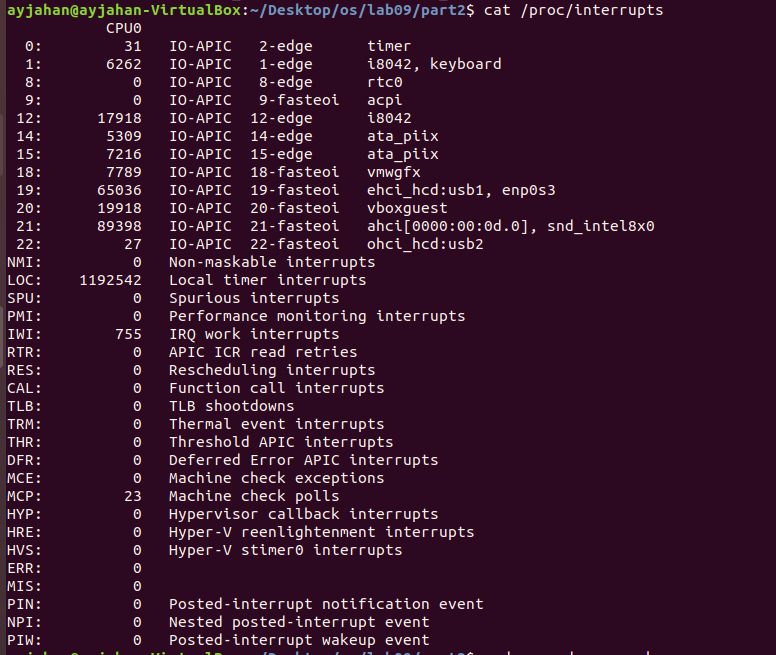
module\_exit(my\_module\_exit);

**Результат работы:**



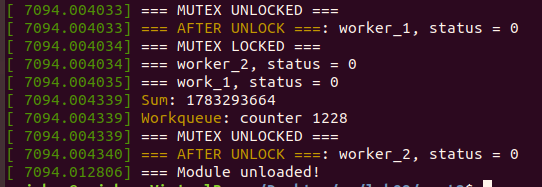


Загрузка модуля



/proc/interrups





Выгрузка модуля