|  |  |
| --- | --- |
| 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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 9 |

**Дисциплина:** Операционные системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-62Б |  |  | Н.А. Гарасев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Н.Ю. Рязанова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**Задание 1:**

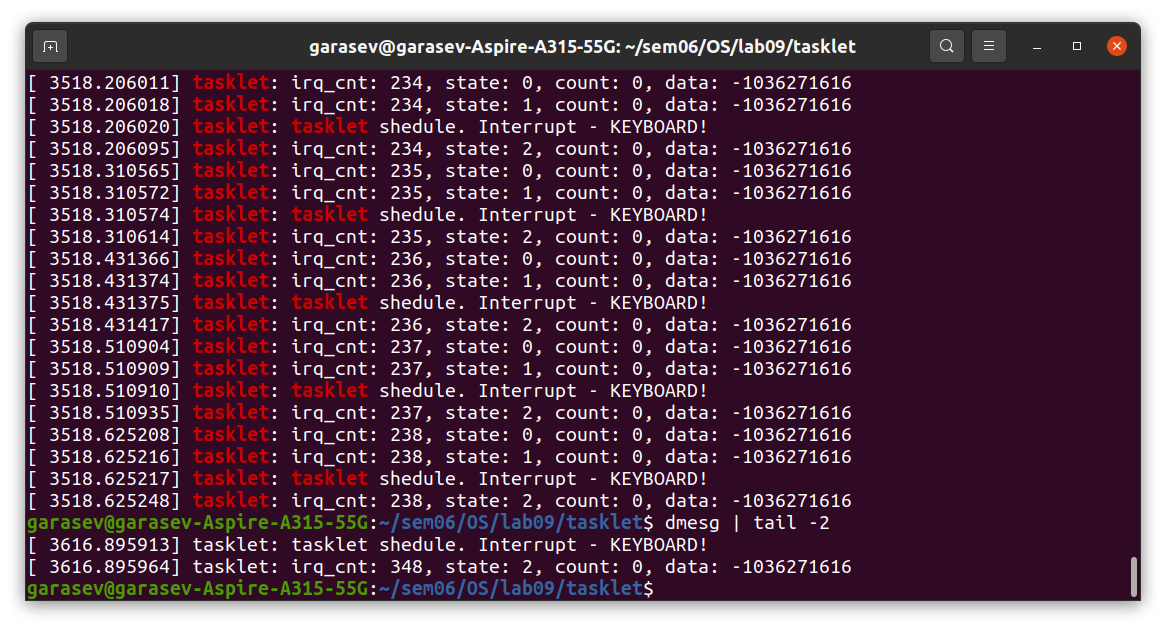
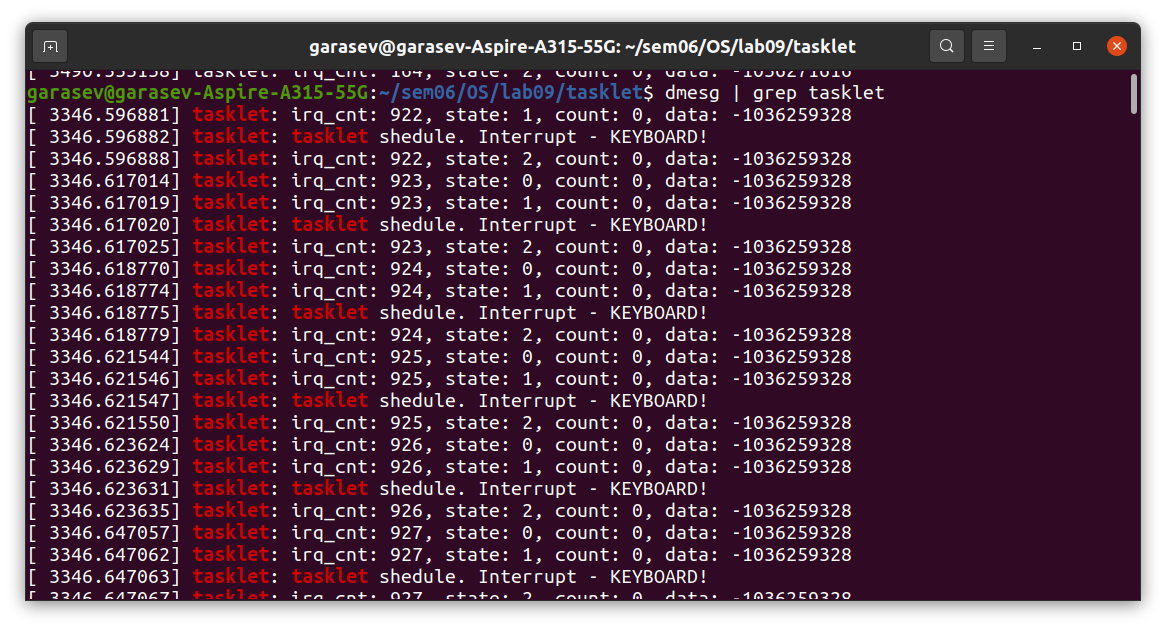
* Написать загружаемый модуль ядра, в котором зарегистрировать обработчик аппаратного прерывания с флагом IRQF\_SHARED.
* Инициализировать тасклет.
* В обработчике прерывания запланировать тасклет на выполнение.
* Вывести информацию о тасклете используя, или printk(), или seq\_file interface - <linux/seq\_file.h>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

**Код программы.**

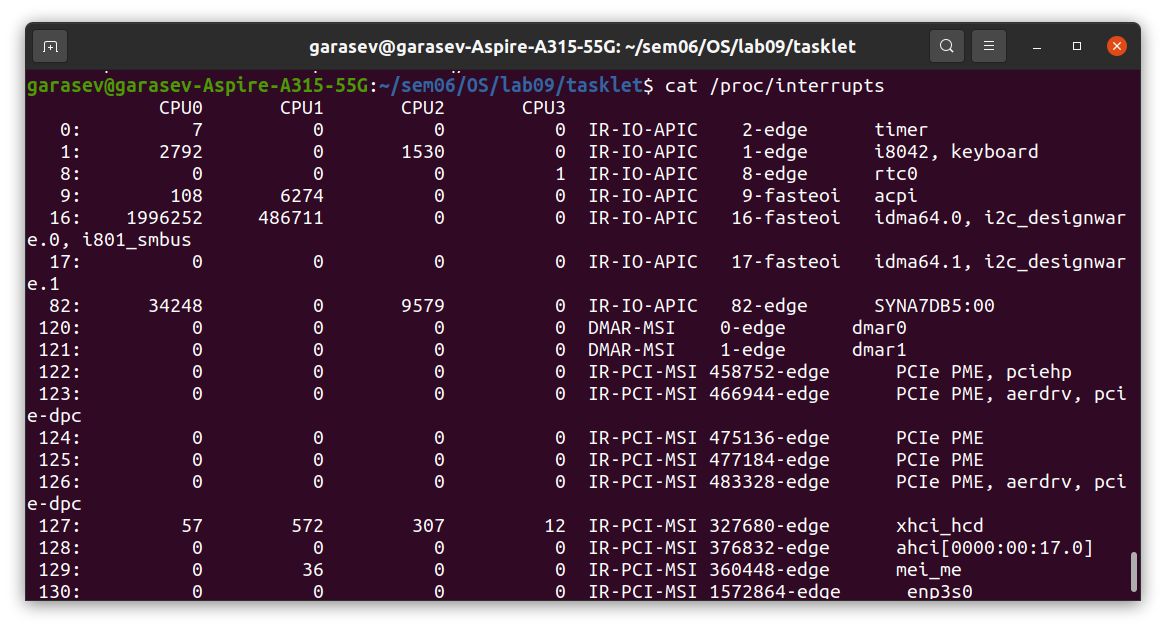
|  |
| --- |
| #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/interrupt.h>  #include <linux/proc\_fs.h>  #include <linux/seq\_file.h>  #include <linux/time.h>  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Garasev");  #define KEYB\_IRQ 1  #define PROC\_FILE\_NAME "tasklet"  static int irq\_cnt = 0;  static struct proc\_dir\_entry \*proc\_file;  char my\_tasklet\_data[] = "KEYBOARD IRQ";  struct tasklet\_struct my\_tasklet;  // не логируется в struct file\_operations, но она указывается в single\_open (там логируется)  static int proc\_tasklet\_show(struct seq\_file \*m, void \*v)  {  printk(KERN\_INFO "tasklet: call proc\_hello\_show\n");  int error = 0;  seq\_printf(m, "tasklet: irq\_cnt: %d, state: %ld, count: %d, data: %ld\n",  irq\_cnt, my\_tasklet.state, my\_tasklet.count, my\_tasklet.data);  return error;  }  // В качестве аргументы две структуры для работы с файлами, потому что файлы в ОС линукс/юникс идентифицируется номер inode и описываются struct inode. А открытые файлы( файлы с которыми работает процесс, когда процесс вызывает open) в системе описываются в системе struct file. В системе имеется одна таблица открытых файлов. То есть это не структура которая описывает файлы открытым процессом, это указатель на системную таблицу открытых файлов.  static int proc\_tasklet\_open(struct inode \*inode, struct file \*file)  {  printk(KERN\_INFO "tasklet: call proc\_open\n");  return single\_open(file, proc\_tasklet\_show, NULL);  }  static const struct file\_operations proc\_tasklet\_fops=  {  .open = proc\_tasklet\_open, // функцию, которую мы сами определили  .release = single\_release,  .read = seq\_read,  };  void my\_tasklet\_func(unsigned long data)  {  // state - состояние тасклета  // счетчик ссылок  // data - аргумент ф-ии обработчика тасклета  printk(KERN\_INFO "tasklet: irq\_cnt: %d, state: %ld, count: %d, data: %ld\n",  irq\_cnt, my\_tasklet.state, my\_tasklet.count, my\_tasklet.data);  }  // Обработчик прерывания  irqreturn\_t irq\_handler(int irq, void \*dev)  {  // Проверка, что произошло именно 1-е прерывание  if(irq == KEYB\_IRQ)  {  // Постановка тасклета в очередь на выполнение  ++irq\_cnt;  printk(KERN\_INFO "tasklet: irq\_cnt: %d, state: %ld, count: %d, data: %ld\n",  irq\_cnt, my\_tasklet.state, my\_tasklet.count, my\_tasklet.data);  tasklet\_schedule(&my\_tasklet);  printk(KERN\_INFO "tasklet: irq\_cnt: %d, state: %ld, count: %d, data: %ld\n",  irq\_cnt, my\_tasklet.state, my\_tasklet.count, my\_tasklet.data);  printk(KERN\_INFO "tasklet: tasklet shedule. Interrupt - KEYBOARD!\n");  return IRQ\_HANDLED; // прерывание обработано  }  else  return IRQ\_NONE; // прерывание не обработано  }  // Инициализация модуля  static int \_\_init my\_module\_init(void)  {  printk(KERN\_INFO "tasklet: MODULE loaded!\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 ===  int ret = request\_irq(KEYB\_IRQ, (irq\_handler\_t)irq\_handler, IRQF\_SHARED,  "keyboard", (void \*)(irq\_handler));  if (ret != 0)  {  printk(KERN\_ERR "tasklet: KEYBOARD IRQ handler wasn't registered");  return ret;  }  // инициализирует структуру tasklet\_struct в соответствие с данными, предоставленными пользователем.  // my\_tasklet\_func - обработчик тасклета  // знач-е 3его пар-ра - ф-я обработчик  tasklet\_init(&my\_tasklet, my\_tasklet\_func, (void \*)(irq\_handler));  printk(KERN\_INFO "tasklet: KEYBOARD IRQ handler was registered successfully");  proc\_file = proc\_create\_data(PROC\_FILE\_NAME, S\_IRUGO | S\_IWUGO, NULL, &proc\_tasklet\_fops, NULL);  if (!proc\_file)  return -ENOMEM;  return ret;  }  // Выход загружаемого модуля  static void \_\_exit my\_module\_exit(void)  {  // Освобождение линии irq от указанного обработчика  // Данные по указателю dev (2ой пар-р) требуются для удаления только конкретного устройства.  free\_irq(KEYB\_IRQ, (void \*)(irq\_handler));  // Удаление тасклета  tasklet\_disable(&my\_tasklet);  // ждет заверш-я тасклета и удаляет тасклет из очереди на выполнение  tasklet\_kill(&my\_tasklet);  if (proc\_file)  remove\_proc\_entry(PROC\_FILE\_NAME, NULL);  printk(KERN\_INFO "tasklet: MODULE unloaded!\n");  }  module\_init(my\_module\_init);  module\_exit(my\_module\_exit); |

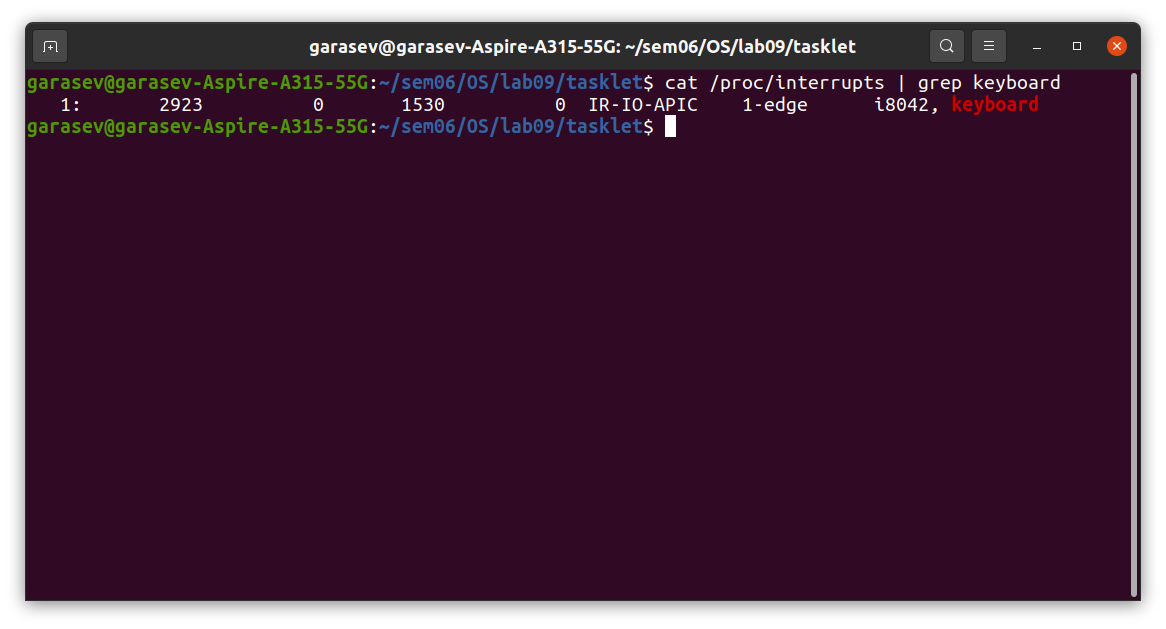
**Демонстрация работы программы.**

При каждом прерывании от клавиатуры вызывается обработчик прерывания my\_irq\_handler. В нем вызывается tasklet\_schedule(). При обработке тасклета выводится информация о нем.



Проверим, что my\_tasklet добавилось в список прерываний. Ниже представлено содержимое файла /proc/interrupts, что говорит о том, что my\_tasklet добавилось в список прерываний.



На рисунке видно, что добавлено прерывание с именем keyboard. 

**Задание 2:**

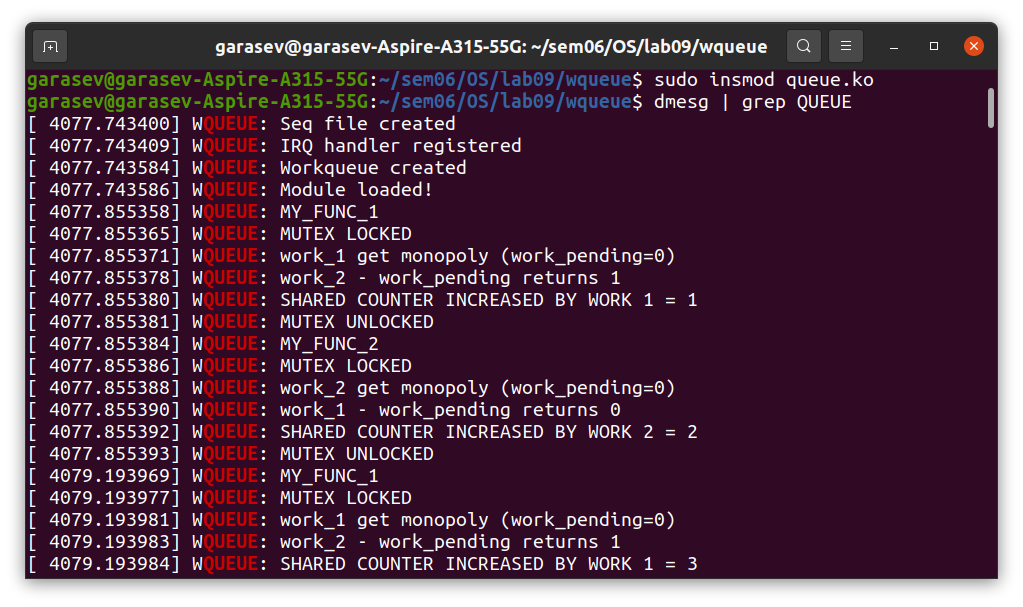
* Написать загружаемый модуль ядра, в котором зарегистрировать обработчик аппаратного прерывания с флагом IRQF\_SHARED.
* Инициализировать очередь работ.
* В обработчике прерывания запланировать очередь работ на выполнение.
* Вывести информацию об очереди работ используя, или printk(), или seq\_file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

**Код программы.**

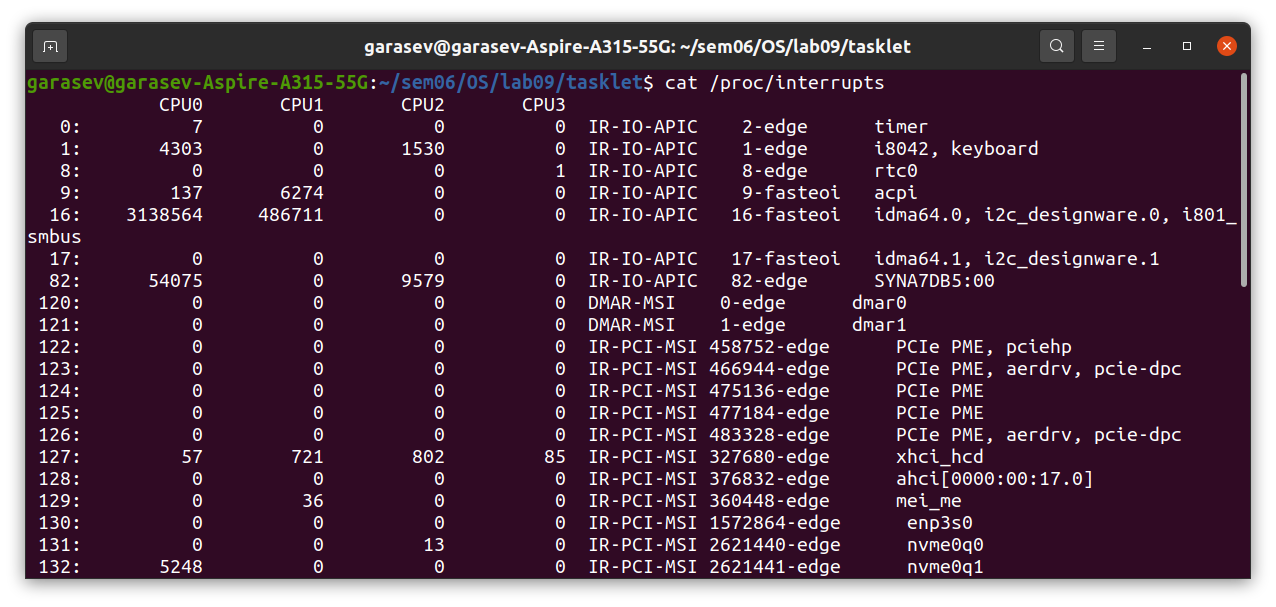
|  |
| --- |
| #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/interrupt.h>  #include <linux/workqueue.h>  #include <linux/mutex.h>  #include <linux/proc\_fs.h>  #include <linux/seq\_file.h>  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Garasev");  #define IRQ\_NUM 1  #define IRQ\_NAME "keyboard"  #define WQ\_NAME "keyboard\_wq"  #define ITER\_COUNT 100000000  static struct workqueue\_struct \*my\_wq;  typedef struct  {  struct work\_struct work;  int work\_num; // номер работы  } my\_work\_t;  // Структуры, описывающие две работы  static my\_work\_t \*work\_1;  static my\_work\_t \*work\_2;  static int shared\_counter = 0;  struct mutex my\_mutex;  static int my\_proc\_show(struct seq\_file \*m, void \*v)  {  seq\_printf(m, "WQUEUE: In user mode: is work\_1 pending - %d, is work\_1 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 "WQUEUE: 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\_func\_1(struct work\_struct \*work)  {  printk(KERN\_INFO "WQUEUE: MY\_FUNC\_1\n");  mutex\_lock(&my\_mutex);  printk(KERN\_INFO "WQUEUE: MUTEX LOCKED\n");  my\_work\_t \*my\_work = (my\_work\_t \*)work;  // первая работа (поток) вошла в крит секцию  printk(KERN\_INFO "WQUEUE: work\_1 get monopoly (work\_pending=%d)\n", work\_pending(&(my\_work->work)));  // вторая работа заблокирована в это время  printk(KERN\_INFO "WQUEUE: work\_2 - work\_pending returns %d\n", work\_pending(&(work\_2->work)));    int i, mul = 0;  for (i = 0; i < ITER\_COUNT; i++)  mul += i;  shared\_counter++;  printk(KERN\_INFO "WQUEUE: SHARED COUNTER INCREASED BY WORK 1 = %d\n",shared\_counter);  mutex\_unlock(&my\_mutex);  printk(KERN\_INFO "WQUEUE: MUTEX UNLOCKED\n");  }  void my\_func\_2(struct work\_struct \*work)  {  printk(KERN\_INFO "WQUEUE: MY\_FUNC\_2\n");  mutex\_lock(&my\_mutex);  printk(KERN\_INFO "WQUEUE: MUTEX LOCKED\n");  my\_work\_t \*my\_work = (my\_work\_t \*)work;  // вторая работа (поток) вошла в крит секцию  printk(KERN\_INFO "WQUEUE: work\_2 get monopoly (work\_pending=%d)\n", work\_pending(&(my\_work->work)));  // первая работа заблокирована в это время  printk(KERN\_INFO "WQUEUE: work\_1 - work\_pending returns %d\n", work\_pending(&(work\_1->work)));  int i, mul = 0;  for (i = 0; i < ITER\_COUNT; i++)  mul += i;  shared\_counter++;  printk(KERN\_INFO "WQUEUE: SHARED COUNTER INCREASED BY WORK 2 = %d\n", shared\_counter);  mutex\_unlock(&my\_mutex);  printk(KERN\_INFO "WQUEUE: MUTEX UNLOCKED\n");  }  irqreturn\_t my\_irq\_handler(int irq\_num, void \*dev\_id)  {  if (irq\_num == IRQ\_NUM)  {  if (work\_1)  queue\_work(my\_wq, (struct work\_struct \*)work\_1);  if (work\_2)  queue\_work(my\_wq, (struct work\_struct \*)work\_2);  return IRQ\_HANDLED;  }  return IRQ\_NONE;  }  static int \_\_init my\_init(void)  {  struct proc\_dir\_entry \*entry;  entry = proc\_create("wqueue", S\_IRUGO | S\_IWUGO, NULL, &props);  if (!entry)  {  printk(KERN\_INFO "WQUEUE: Can't create seq file\n");  return -ENOMEM;  }  printk(KERN\_INFO "WQUEUE: Seq file created\n");  if (request\_irq(IRQ\_NUM, my\_irq\_handler, IRQF\_SHARED, IRQ\_NAME, my\_irq\_handler))  {  printk(KERN\_ERR "WQUEUE: Failed to register IRQ handler\n");  return -ENOMEM;  }  printk(KERN\_INFO "WQUEUE: IRQ handler registered\n");  work\_1 = (my\_work\_t \*)kmalloc(sizeof(my\_work\_t), GFP\_KERNEL);  if (work\_1)  {  INIT\_WORK((struct work\_struct \*)work\_1, my\_func\_1);  work\_1->work\_num = 1;  }  else  {  printk(KERN\_ERR "WQUEUE: Failed alloc\n");  return -ENOMEM;  }  work\_2 = (my\_work\_t \*)kmalloc(sizeof(my\_work\_t), GFP\_KERNEL);  if (work\_2)  {  INIT\_WORK((struct work\_struct \*)work\_2, my\_func\_2);  work\_2->work\_num = 2;  }  else  {  printk(KERN\_ERR "WQUEUE: Failed alloc\n");  }  my\_wq = create\_workqueue(WQ\_NAME);  if (!my\_wq)  {  free\_irq(IRQ\_NUM, my\_irq\_handler);  printk(KERN\_ERR "WQUEUE: Failed to create work queue\n");  return -ENOMEM;  }  printk(KERN\_INFO "WQUEUE: Workqueue created\n");  mutex\_init(&my\_mutex);  printk(KERN\_INFO "WQUEUE: Module loaded!\n");  return 0;  }  static void \_\_exit my\_exit(void)  {  free\_irq(IRQ\_NUM, my\_irq\_handler);  flush\_workqueue(my\_wq);  destroy\_workqueue(my\_wq);  mutex\_destroy(&my\_mutex);  if (work\_1)  kfree(work\_1);  if (work\_2)  kfree(work\_2);  remove\_proc\_entry("wqueue", NULL);  printk(KERN\_INFO "WQUEUE: Module unloaded!\n");  }  module\_init(my\_init);  module\_exit(my\_exit); |

**Демонстрация работы программы.**

Был написан обработчик прерываний для irq 1. Загрузим модуль и проверим журнал.



Проверим, что keyboard добавилось в список прерываний. Ниже представлено содержимое файла /proc/interrupts, что говорит о том, что keyboard добавилось в список прерываний.



Покажем, что модуль удален. Ниже представлено содержание файла /proc/interrupts, что говорит о том, что keyboard удалилось из списка прерывания после выгрузки модуля.

