

```

1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/interrupt.h>
4 #include <linux/slab.h>
5 #include <asm/io.h>
6 #include <linux/stddef.h>
7 #include <linux/workqueue.h>
8 #include <linux/delay.h>
9
10 #include "ascii.h"
11
12 MODULE_LICENSE("GPL");
13 MODULE_AUTHOR("Maslova Marina");
14
15 typedef struct
16 {
17     struct work_struct work;
18     int code;
19 } my_work_struct_t;
20
21 static struct workqueue_struct *my_wq;
22 static my_work_struct_t *work1;
23 static struct work_struct *work2;
24 int keyboard_irq = 1;
25
26 void work1_func(struct work_struct *work)
27 {
28     my_work_struct_t *my_work = (my_work_struct_t *)work;
29     int code = my_work->code;
30
31     printk(KERN_INFO "MyWorkQueue: work1 begin");
32     printk(KERN_INFO "MyWorkQueue: key code is %d", code);
33
34     if (code < 84)
35         printk(KERN_INFO "MyWorkQueue: the key is %s", ascii[code]);
36
37     printk(KERN_INFO "MyWorkQueue: work1 end");
38 }
39
40 void work2_func(struct work_struct *work)
41 {
42     printk(KERN_INFO "MyWorkQueue: work2 sleep begin");
43     msleep(10);
44     printk(KERN_INFO "MyWorkQueue: work2 sleep end");
45 }
46
47 irqreturn_t my_irq_handler(int irq, void *dev)
48 {
49     int code;
50     printk(KERN_INFO "MyWorkQueue: my_irq_handler");
51     if (irq == keyboard_irq)
52     {
53         printk(KERN_INFO "MyWorkQueue: called by keyboard_irq");
54
55         code = inb(0x60);
56         work1->code = code;
57
58         queue_work(my_wq, (struct work_struct *)work1);
59         queue_work(my_wq, work2);
60
61         return IRQ_HANDLED;
62     }
63     printk(KERN_INFO "MyWorkQueue: called not by keyboard_irq");
64
65     return IRQ_NONE;

```

```

66 }
67
68 static int __init my_workqueue_init(void)
69 {
70     int ret;
71     printk(KERN_INFO "MyWorkQueue: init");
72
73     my_wq = create_workqueue("my_wq");
74     if (my_wq == NULL)
75     {
76         printk(KERN_ERR "MyWorkQueue: create queue error");
77         return -1;
78     }
79
80     work1 = kcalloc(sizeof(my_work_struct_t), GFP_KERNEL);
81     if (work1 == NULL)
82     {
83         printk(KERN_ERR "MyWorkQueue: work1 alloc error");
84         destroy_workqueue(my_wq);
85         return -1;
86     }
87
88     work2 = kcalloc(sizeof(struct work_struct), GFP_KERNEL);
89     if (work2 == NULL)
90     {
91         printk(KERN_ERR "MyWorkQueue: work2 alloc error");
92         destroy_workqueue(my_wq);
93         kfree(work1);
94         return -1;
95     }
96
97     INIT_WORK((struct work_struct *)work1, work1_func);
98     INIT_WORK(work2, work2_func);
99
100     ret = request_irq(keyboard_irq, my_irq_handler, IRQF_SHARED,
101                      "test_my_irq_handler", (void *) my_irq_handler);
102     if (ret)
103     {
104         printk(KERN_ERR "MyWorkQueue: request_irq error");
105         destroy_workqueue(my_wq);
106         kfree(work1);
107         kfree(work2);
108     }
109     else
110         printk(KERN_ERR "MyWorkQueue: loaded");
111
112     return ret;
113 }
114
115 static void __exit my_workqueue_exit(void)
116 {
117     printk(KERN_INFO "MyWorkQueue: exit");
118     synchronize_irq(keyboard_irq);
119     free_irq(keyboard_irq, my_irq_handler);
120     flush_workqueue(my_wq);
121     destroy_workqueue(my_wq);
122     kfree(work1);
123     kfree(work2);
124     printk(KERN_INFO "MyWorkQueue: unloaded");
125 }
126
127 module_init(my_workqueue_init);
128 module_exit(my_workqueue_exit);

```