```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/interrupt.h>
#include <linux/slab.h>
#include <asm/io.h>
#include <linux/stddef.h>
#include <linux/workqueue.h>
#include <linux/delay.h>

#include "ascii.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Maslova Marina");

typedef struct
{
    struct work_struct work;
    int code;
} my_work_struct_t;

static struct workqueue_struct *my_wq;
static my_work_struct_t *work1;
static struct work_struct *work2;
int keyboard_irq = 1;

void work1_func(struct work_struct *work)
{
    my_work_struct_t *my_work = (my_work_struct_t *)work;
    int code = my_work->code;

    printk(KERN_INFO "MyWorkQueue: work1 begin");
    printk(KERN_INFO "MyWorkQueue: key code is %d", code);

    if (code < 84)
        printk(KERN_INFO "MyWorkQueue: the key is %s", ascii[code]);

    printk(KERN_INFO "MyWorkQueue: work1 end");
}

void work2_func(struct work_struct *work)
{
    printk(KERN_INFO "MyWorkQueue: work2 sleep begin");
    msleep(10);
    printk(KERN_INFO "MyWorkQueue: work2 sleep end");
}

irqreturn_t my_irq_handler(int irq, void *dev)
{
    int code;
    printk(KERN_INFO "MyWorkQueue: my_irq_handler");
    if (irq == keyboard_irq)
    {
        printk(KERN_INFO "MyWorkQueue: called by keyboard_irq");

        code = inb(0x60);
        work1->code = code;

        queue_work(my_wq, (struct work_struct *)work1);
        queue_work(my_wq, work2);

        return IRQ_HANDLED;
    }
    printk(KERN_INFO "MyWorkQueue: called not by keyboard_irq");

    return IRQ_NONE;
}

static int __init my_workqueue_init(void)
{
    int ret;
    printk(KERN_INFO "MyWorkQueue: init");

    my_wq = create_workqueue("my_wq");
    if (my_wq == NULL)
    {
        printk(KERN_ERR "MyWorkQueue: create queue error");
        return -1;
    }

    work1 = kmalloc(sizeof(my_work_struct_t), GFP_KERNEL);
    if (work1 == NULL)
    {
        printk(KERN_ERR "MyWorkQueue: work1 alloc error");
        destroy_workqueue(my_wq);
        return -1;
    }

    work2 = kmalloc(sizeof(struct work_struct), GFP_KERNEL);
    if (work2 == NULL)
    {
        printk(KERN_ERR "MyWorkQueue: work2 alloc error");
        destroy_workqueue(my_wq);
        kfree(work1);
        return -1;
    }

    INIT_WORK((struct work_struct *)work1, work1_func);
    INIT_WORK(work2, work2_func);

    ret = request_irq(keyboard_irq, my_irq_handler, IRQF_SHARED,
                      "test_my_irq_handler", (void *) my_irq_handler);
    if (ret)
    {
        printk(KERN_ERR "MyWorkQueue: request_irq error");
        destroy_workqueue(my_wq);
        kfree(work1);
        kfree(work2);
    }
    else
        printk(KERN_ERR "MyWorkQueue: loaded");

    return ret;
}

static void __exit my_workqueue_exit(void)
{
    printk(KERN_INFO "MyWorkQueue: exit");
    synchronize_irq(keyboard_irq);
    free_irq(keyboard_irq, my_irq_handler);
    flush_workqueue(my_wq);
    destroy_workqueue(my_wq);
    kfree(work1);
    kfree(work2);
    printk(KERN_INFO "MyWorkQueue: unloaded");
}

module_init(my_workqueue_init);
module_exit(my_workqueue_exit);
```