2009-12-21 Linux kernel driver that handles IRQ

Plan

- make minimalistic driver that requests IRQ on load and release it on unload;
- pass IRQ no. as a module parameter;
- (variation) make ISR lock spinlock and not release it; see how it looks;
- (extra) modify qemu to allow emitting IRQ from monitor;

Worklog

- get your future rootfs: build this directory tree, put busybox for i386 into bin, make symlinks to it, copy device nodes from system /dev

```
├── bin/
│   ├── busybox*
│   ├── cat -> busybox*
│   ├── df -> busybox*
│   ├── du -> busybox*
│   ├── insmod -> busybox*
│   ├── less -> busybox*
│   ├── ln -> busybox*
│   ├── ls -> busybox*
│   ├── lsmod -> busybox*
│   ├── mount -> busybox*
│   ├── rmmod -> busybox*
│   ├── sh -> busybox*
│   ├── top -> busybox*
│   └── vi -> busybox*
├── dev/
│   ├── pts/
│   ├── console
│   ├── kmsg
│   ├── loop0
│   ├── loop1
│   ├── loop2
│   ├── loop3
│   ├── loop4
│   ├── loop5
│   ├── loop6
│   ├── loop7
│   ├── mem
│   ├── null
│   ├── ptmx
│   ├── random
```

```
│   ├── systty
│   ├── tty
│   ├── tty0
│   ├── tty1
│   ├── tty2
│   ├── urandom
│   └── zero
├── lib/
│   └── modules/
│       └── 2.6.33-rc1/
├── proc/
├── sbin/
├── sys/
├── tmp/
├── init*
└── test.ko
```

- make /init like this:

```
#! /bin/sh

mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devpts devpts /dev/pts

while :
do
  /bin/sh
done
```

- get a linux, configure and compile it:

```
git clone
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
make ARCH=i386 defconfig
# turn off networking as we don't need it
# turn on initramfs:
# General setup --->
#   [*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
#   (/home/dumb/ws/ii/games-build/games/src/lkm/rootfs) Initramfs source
file(s)
make ARCH=i386 menuconfig
make ARCH=i386 -j3 bzImage
```

- make test.c, module source:

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

```
static irqreturn_t test_irq_handler(int i, void *dev)
{
    static spinlock_t lock = __SPIN_LOCK_UNLOCKED(lock);
    unsigned long flags;

    printk(KERN_INFO "%s(%d, %p)\n", __func__, i, dev);
    spin_lock_irqsave(&lock, flags);
    printk(KERN_INFO "flags: %lx\n", flags);
    spin_unlock_irqrestore(&lock, flags);
    return IRQ_HANDLED;
}

static int test_irq;
static int test_irq_no;
#define IRQ_NO test_irq_no

module_param(test_irq_no, int, 13);
MODULE_PARM_DESC(test_irq_no, "requested IRQ line");

static int __init test_init(void)
{
        int rc;
        printk(KERN_INFO "%s\n", __func__);
        rc = request_irq(IRQ_NO, test_irq_handler, IRQF_SHARED,
"test_device", &test_irq);
        printk(KERN_INFO "request_irq(%d, %p): %d\n", IRQ_NO, &test_irq,
rc);
        if (!rc)
            test_irq = IRQ_NO;
        return 0;
}

static void __exit test_exit(void)
{
        printk(KERN_INFO "%s\n", __func__);
        if (test_irq)
            free_irq(test_irq, &test_irq);
}

module_init(test_init);
module_exit(test_exit);

MODULE_AUTHOR("jcmvbkbc");
MODULE_LICENSE("GPL");
```

- make a Makefile for it:

```
PWD = $(shell pwd)
EXTRA_CFLAGS += -Wall

obj-m += test.o
```

```
default: modules

modules clean:
        make -C $(KDIR) M=$(PWD) $@
```

- compile it like this: make -C “$PWD/linux-2.6" ARCH=i386 M="$PWD” modules
- copy test.ko to the rootfs / and rebuild bzImage
- run it in qemu: qemu -kernel bzImage
- load module inside the shell in qemu: insmod test.ko test_irq_no=1
- press and release keys (irq 1 is keyboard interrupt)
- see it there: cat /proc/interrupts
- unload it: rmmod test

Code: ftp://kkv.spb.su/pub/home/dumb/ws/lted/20091221/

Conclusion

- initramfs is fun: bzImage holds rootfs and re-builds it even if kernel code is not changed;
- spinlock re-acquisition locks up hard;
- qemu internal hardware objects are not that easily navigable – to be continued;

linux, lkm, qemu, lted