07-2 Device Driver Basics

Using kernel modules

Free Electrons

Loadable kernel modules

- Modules: add a given functionality to the kernel (drivers, filesystem support, and many others)
- Can be loaded and unloaded at any time, only when their functionality is needed
- Useful to keep the kernel image size to the minimum (essential in GNU/Linux distributions for PCs)
- Also useful to reduce boot time: you don't spent time initializing devices and kernel features that you only need later
- Caution: once loaded, have full access to the whole kernel address space. No particular protection

Minimal Device Driver (Listing 8-1)

```
/* Example Minimal Character Device Driver */
#include <linux/module.h>
static int __init hello_init(void) {
  printk(KERN_INFO "Hello Example Init\n");
  return 0;
}
static void __exit hello_exit(void) {
  printk("Hello Example Exit\n");
}
module_init(hello_init);
module_exit(hello_exit);

MODULE_AUTHOR("Chris Hallinan");
MODULE_DESCRIPTION("Hello World Example");
MODULE_LICENSE("GPL");
```

Module Build Infrastructure

1. Starting from the top-level Linux source directory, create a directory under `.../drivers/char` called `examples`.
2. Add a menu item to the kernel configuration to enable building `examples` and to specify built-in or loadable kernel module.
3. Add the new examples subdirectory to the `.../drivers/char/Makefile` conditional on the menu item created in step 2.
4. Create a `Makefile` for the new `examples` directory, and add the `hello1.o` module object to be compiled conditional on the menu item created in step 2.
5. Finally, create the driver `hello1.c` source file from Listing 8-1.

See Section 8.1.4, page 205

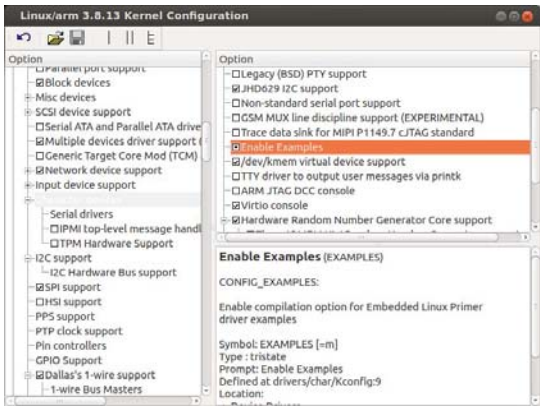
Typo page 206

```
diff --git a/drivers/char/Kconfig b/drivers/char/Kconfig
index 6f31c94..0805290 100644
--- a/drivers/char/Kconfig
+++ b/drivers/char/Kconfig
@@ -4,6 +4,13 @@

 menu "Character devices"

+config EXAMPLES
+       tristate "Enable Examples"
+       default m
+       ---help---
+         Enable compilation option for Embedded Linux Primer
+         driver examples
+
config DEVKMEM
          bool "/dev/kmem virtual device support"
          default y
          help
```

Must be lower case

In new kernel

Check Config



Module Build Output – In tree

```
host$ cd bb-kernel/KERNEL
host$ time make modules
 CHK     include/generated/uapi/linux/version.h
 CHK     include/generated/utsrelease.h
make[1]: `include/generated/mach-types.h' is up to date.
 CALL    scripts/checksyscalls.sh
  CC [M]  drivers/char/examples/hello1.o
  Building modules, stage 2.
 MODPOST 1326 modules
  LD [M]  drivers/char/examples/hello1.ko

real    0m33.706s
user    0m31.462s
sys     0m4.824s
```

First time takes much longer
Try make –j*X* modules

Module First Build Output

```
host$ time make modules
 HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf -s arch/arm/Kconfig
*
* Restart config...
*
*
* Character devices
*
Enable Examples (EXAMPLES) [M/n/y/?] (NEW)
Virtual terminal (VT) [Y/n/?] y
  Enable character translations in console (CONSOLE_TRANSLATIONS) [Y/n/?] y
  Support for console on virtual terminal (VT_CONSOLE) [Y/n/?] y
  Support for binding and unbinding console drivers (VT_HW_CONSOLE_BINDING) [Y/n/?] y
/dev/kmem virtual device support (DEVKMEM) [Y/n/?] y
Non-standard serial port support (SERIAL_NONSTANDARD) [N/y/?] n
Unix98 PTY support (UNIX98_PTYS) [Y/n/?] y
  Support multiple instances of devpts (DEVPTS_MULTIPLE_INSTANCES) [N/y/?] n
Legacy (BSD) PTY support (LEGACY_PTYS) [N/y/?] n
Hardware Random Number Generator Core support (HW_RANDOM) [Y/n/m/?] y
  Timer IOMEM HW Random Number Generator support (HW_RANDOM_TIMERIOMEM) [N/m/y/?] n
Siemens R3964 line discipline (R3964) [N/m/y/?] n
RAW driver (/dev/raw/rawN) (RAW_DRIVER) [N/m/y/?] n
```

If you get this, run
`make xconfig`
and save

Module Build Output – Out-of-tree

```
host$ cd …/drivers/char/examples
host$ vi Makefile
ifneq ($(KERNELRELEASE),)
obj-$(CONFIG_EXAMPLES) += hello1.o
else
KDIR := ~/BeagleBoard/bb-kernel/KERNEL
all:
	$(MAKE) -C $(KDIR) M=$$PWD
endif
```

- **Compile with**
- `host$ make`

Once built… Option 1

On the Beagle…Two choices….

- Option 1:
  `make INSTALL_MOD_PATH=~/BeagleBoard modules_install`
- Will create **`lib`** directory in **`~/BeagleBoard`** with everything that goes in **`/lib`** on the Beagle

```
host$ ls -F ~/BeagleBoard/lib/modules/3.8.13+/
build@              modules.dep.bin     modules.seriomap
kernel/             modules.devname     modules.softdep
modules.alias       modules.ieee1394map modules.symbols
modules.alias.bin   modules.inputmap    modules.symbols.bin
modules.builtin     modules.isapnpmap   modules.usbmap
modules.builtin.bin modules.ofmap       source@
modules.ccwmap      modules.order
modules.dep         modules.pcimap
```

- Then

```
host$ rm build source
host$ scp -r ~/BeagleBoard/lib root@beagle:/lib
```

- Could take a while to transfer

Once built… Option 2

- Just copy the new file you created

```
host$ scp …/drivers/char/examples/hello1.ko root@beagle:.
```

- On the Beagle

```
bone$ uname -r
3.8.13-bone67
bone$ cd /lib/modules/`uname -r`/kernel/drivers/char/
bone $ mkdir examples
bone $ cd examples
bone $ mv ~/hello1.ko  .
```

Note backquotes

- **Now build a new dependencies file**

```
beagle$ depmod -a
```

Loading and Unloading a Module

```
beagle$ modprob hello1
beagle$ dmesg | tail -4
[    9.106206] snd-usb-audio 1-1:1.0: usb_probe_interface
[    9.106244] snd-usb-audio 1-1:1.0: usb_probe_interface - got id
[    9.813239] usbcore: registered new interface driver snd-usb-
  audio
[  109.308551] Hello Example Init
beagle$ rmmod hello1
beagle$ dmesg | tail -4
[    9.106244] snd-usb-audio 1-1:1.0: usb_probe_interface - got id
[    9.813239] usbcore: registered new interface driver snd-usb-
  audio
[  109.308551] Hello Example Init
[  241.037368] Hello Example Exit
```

Module Utilities

```
$ insmod /lib/modules/`uname -r`/kernel/drivers/char/examples/hello1.ko
```

- **No need build dependencies file**

Example Driver with Parameter

```
/* Example Minimal Character Device Driver */
#include <linux/module.h>

static int debug_enable = 0;
module_param(debug_enable, int, 0);

MODULE_PARM_DESC(debug_enable, "Enable module debug mode.");

static int __init hello_init(void) {
    /* Now print value of new module parameter */
    printk("Hello Example Init - debug mode is %s\n",
           debug_enable ? "enabled" : "disabled");
    return 0;
}
```

```
static void __exit hello_exit(void) {
    printk("Hello Example Exit\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_AUTHOR("Chris Hallinan");
MODULE_DESCRIPTION("Hello World
    Example");
MODULE_LICENSE("GPL");
```

Passing Parameters to a Module

```
beagle$ modprobe hello2 debug_enable=1
Hello Example Init – debug mode is enabled
beagle$ rmmod hello2
beagle$ modprobe hello2
Hello Example Init – debug mode is disabled
```

Other module commands

```
# /sbin/lsmod
# /sbin/modinfo hello1
# /sbin/rmmod hello1
# /sbin/depmod (creates modules.dep.bin)
```

- Go play with them

Adding File System Ops to Hello.c

- Section 8.3, page 217 has a long example about adding file system operations to **`hello.c`**
- Look it over
- Creates a new device (**`/dev/hello1`**)
- You can read and write it
- Do it

Driver File System Operations

- Once a device driver is loaded into the live kernel…
  - **`open()`** is used to prepare it for subsequent operations
  - **`release()`** is used to clean up
  - **`ioctl()`** is used for nonstandard communication
- Think in terms of reading and writing a file…

```
fd = open("file", …
read(fd, …
close(fd)
```

open/release additions to hello.c

```
#include <linux/fs.h>

#define  HELLO_MAJOR 234
…
struct file_operations hello_fops;

static int hello_open(struct inode *inode, struct file *file) {
    printk("hello_open: successful\n");
    return 0;
}

static int hello_release(struct inode *inode, struct file *file) {
    printk("hello_release: successful\n");
    return 0;
}
```

read/write additions to hello.c

```
static ssize_t hello_read(struct file *file,
      char *buf, size_t count, loff_t *ptr) {
    printk("hello_read: returning zero bytes\n");
    return 0;
}

static ssize_t hello_write(struct file *file,
    const char *buf, size_t count, loff_t * ppos)
{
    printk("hello_read: accepting zero bytes\n");
    return 0;
}
```

ioctl additions to hello.c

```
static long hello_ioctl(struct inode *inode,
      struct file *file, unsigned int cmd,
      unsigned long arg) {
    printk("hello_ioctl: cmd=%ld, arg=%ld\n",
           cmd, arg);
    return 0;
}
```

init additions to hello.c

```
#define  HELLO_MAJOR 234
static int __init hello_init(void)
{
    int ret;
    printk("Hello Example Init - debug mode is %s\n",
           debug_enable ? "enabled" : "disabled");
    ret = register_chrdev(HELLO_MAJOR, "hello1", &hello_fops);
        if (ret < 0) {
            printk("Error registering hello device\n");
            goto hello_fail1;
        }
    printk("Hello: registered module successfully!\n");

    /* Init processing here... */

    return 0;

hello_fail1:
    return ret;
}
```

Major number for device driver

- Every device has a major and minor number

```
$ ls -ls /dev/console
0 crw------- 1 yoder root 5, 1 2011-02-06 17:57 /dev/console
```

- Device numbers *used* to be statically assigned
- See **…/Documentation/devices.txt**

```
 5 char  Alternate TTY devices
                   0 = /dev/tty             Current TTY device
                   1 = /dev/console System console
                   2 = /dev/ptmx            PTY master multiplex
                  64 = /dev/cua0            Callout device for ttyS0
```

- The text uses static assignment

```
234-239         UNASSIGNED
240-254 char    LOCAL/EXPERIMENTAL USE
```

Registering our functions

- Struct file_operations is used bind our functions to the requests from the file system.

```
struct file_operations hello_fops = {
    owner:    THIS_MODULE,
    read:     hello_read,
    write:    hello_write,
    ioctl_ioctl:    hello_ioctl,
    open:     hello_open,
    release: hello_release,
};
```

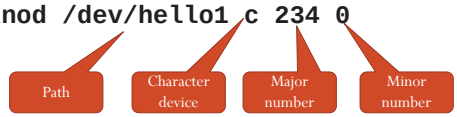
init additions to hello.c

```
#define  HELLO_MAJOR 234
static int __init hello_init(void)
{
    int ret;
    printk("Hello Example Init - debug mode is %s\n",
           debug_enable ? "enabled" : "disabled");
    ret = register_chrdev(HELLO_MAJOR, "hello1", &hello_fops);
        if (ret < 0) {
            printk("Error registering hello device\n");
            goto hello_fail1;
        }
    printk("Hello: registered module successfully!\n");
    /* Init processing here... */
    return 0;
hello_fail1:
    return ret;
}
```

Device Nodes and **mknod**

- Use **mknod** to create a new device

```
$ mknod /dev/hello1 c 234 0
```



- Then

```
$ ls -l /dev/hello1
crw-r—r--   1 root  root  234, 0 Apr 2 2011  /dev/hello1
```

Dynamic Major Number

- The above example uses the older *static* method to assign a device number
- Today dynamic allocation is preferred
- Here is how:

```
#include <linux/kdev_t.h>
dev_t dev;
```

- This declares **dev** to be a device number (both major and minor). Now assign it a value
- **`dev = MKDEV(234, 0);`**

Requesting a number

- Now request a number

```
#include <linux/fs.h>
int register_chrdev_region(dev, 4, "hello");
```

  - This requests a device number starting with 234 (previous page)
  - It asks for 4 minor numbers
  - Uses the name "hello"
- When done with the device use:

```
void unregister_chrdev_region(dev, 4);
```

Using **mknod**

- If you major number is assigned dynamically, how do you use **mknod**? Try the following

```
module="hello"
device="hello"
mode="664"
# remove stale nodes
/sbin/insmod ./$module.ko $* || exit 1
rm -f /dev/${device}0
major=`awk "\\$2==\"$module\" {print \\$1} /proc/devices`
mknod /dev/${device}0 c $major 0
```

/proc/devices

```
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 14 sound
 29 fb
 81 video4linux
 89 i2c
 90 mtd
116 alsa
128 ptm
136 pts
153 spi
161 ircomm
166 ttyACM
180 usb
189 usb_device
212 DVB
216 rfcomm
226 drm
234 hello3
244 ttyGS
245 ttySDIO
246 usbmon
247 uio
248 ttyO
249 bsg

Block devices:
  1 ramdisk
259 blkext
  7 loop
  8 sd
 31 mtdblock
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
179 mmc
254 device-mapper
```

Assignment

- See http://elinux.org/EBC_Device_Drivers

Module dependencies

- Some kernel modules can depend on other modules, which need to be loaded first
- Example: the usb-storage module depends on the scsi_mod, libusual and usbcore modules
- Dependencies are described in /lib/modules/<kernel-version>/modules.dep

/lib/modules/3.8.13/modules.dep

```
kernel/lib/raid6/raid6_pq.ko:
kernel/lib/ts_kmp.ko:
kernel/lib/ts_bm.ko:
kernel/lib/ts_fsm.ko:
kernel/lib/notifier-error-inject.ko:
kernel/lib/pm-notifier-error-inject.ko:
kernel/lib/notifier-error-inject.ko
kernel/lib/mpi/mpi.ko:
kernel/lib/asn1_decoder.ko:
kernel/lib/oid_registry.ko:
kernel/drivers/char/examples/hello2.ko:
kernel/drivers/char/examples/hello3.ko:
kernel/drivers/char/examples/hello1.ko:
```

Kernel log

When a new module is loaded,
related information is available in the kernel log

- The kernel keeps its messages in a circular buffer (so that it doesn't consume more memory with many messages)
- Kernel log messages are available through the **`dmesg`** command
  ("**di**agnostic **mes**sa**g**e")
- Kernel log messages are also displayed in the system console (messages can be filtered by level using **`/proc/sys/kernel/printk`**)

printk

- **`/proc/sys/kernel/printk`**
- The four values in this file are
  - *console_loglevel*,
  - *default_message_loglevel*,
  - *minimum_console_level* and
  - *default_console_loglevel*.
- These values influence **`printk()`** behavior when printing or logging error messages
- Messages with a higher priority than *console_loglevel* will be printed to the console
- Messages without an explicit priority will be printed with priority *default_message_level*

http://www.tin.org/bin/man.cgi?section=5&topic=proc

Kernel log levels

0 (KERN_EMERG)	The system is unusable
1 (KERN_ALERT)	Actions that must be taken care of immediately
2 (KERN_CRIT)	Critical conditions
3 (KERN_ERR)	Noncritical error conditions
4 (KERN_WARNING)	Warning conditions that should be taken care of
5 (KERN_NOTICE)	Normal, but significant events
6 (KERN_INFO)	Informational messages that require no action
7 (KERN_DEBUG)	Kernel debugging messages, output by the

Module utilities (1)

- modinfo <module_name>
  modinfo <module_path>.ko
  Gets information about a module: parameters, license, description and dependencies.
  Very useful before deciding to load a module or not.
- sudo insmod <module_path>.ko
  Tries to load the given module. The full path to the module object file must be given.

Understanding module loading

- When loading a module fails, insmod often doesn't give you enough details!
- Details are often available in the kernel log
- Example:

```
beagle$ sudo insmod ./intr_monitor.ko
insmod: error inserting './intr_monitor.ko': -1
Device or resource busy
beagle$ dmesg
[17549774.552000] Failed to register handler for
irq channel 2
```

Module utilities (2)

- sudo modprobe <module_name>
  Most common usage of modprobe: tries to load all the modules the given module depends on, and then this module. Lots of other options are available. modprobe automatically looks in `/lib/modules/<version>/` for the object file corresponding to the given module name.
- lsmod
  Displays the list of loaded modules
  Compare its output with the contents of /proc/modules!

lsmod

```
beagle$ lsmod
Module                  Size  Used by
hello3                  1952  0
iptable_nat             2493  0
nf_conntrack_ipv4      13161  1
nf_defrag_ipv4          1378  1 nf_conntrack_ipv4
nf_nat_ipv4             3487  1 iptable_nat
nf_nat                 16075  2 nf_nat_ipv4,iptable_nat
nf_conntrack           80162  4 nf_nat,nf_nat_ipv4,iptable_nat,nf_conntrack_ipv4
ip_tables              11013  1 iptable_nat
x_tables               16848  1 ip_tables
g_multi                56699  2
libcomposite           17303  1 g_multi
rfcomm                 30945  0
ircomm_tty             17315  0
ircomm                 10198  1 ircomm_tty
irda                  108219  2 ircomm_tty,ircomm
hidp                   12626  0
bluetooth             190961  4 hidp,rfcomm
…
```

Module utilities (3)

- sudo rmmod <module_name>
  Tries to remove the given module.
  Will only be allowed if the module is no longer in use (for example, no more processes opening a device file)
- sudo modprobe -r <module_name>
  Tries to remove the given module and all dependent modules (which are no longer needed after the module removal)

Passing parameters to modules

- Find available parameters:
  modinfo snd-intel8x0m
- Through insmod:
  sudo insmod ./snd-intel8x0m.ko index=-2
- Through modprobe:
  Set parameters in /etc/modprobe.conf or in any file in /etc/modprobe.d/:
  options snd-intel8x0m index=-2
- Through the kernel command line,
  when the module is built statically into the kernel:

  snd-intel8x0m.index=-2

  module name
  module parameter name
  module parameter value

Useful reading

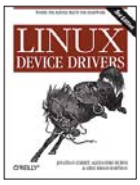
Linux Kernel in a Nutshell, Dec 2006

- By Greg Kroah-Hartman, O'Reilly
  http://www.kroah.com/lkn/
- A good reference book and guide on configuring, compiling and managing the Linux kernel sources.
- Freely available on-line!
  Great companion to the printed book for easy electronic searches!
  Available as single PDF file on
  http://free-electrons.com/community/kernel/lkn/
- In **`exercises/pptx`**

Useful reading too

Linux Device Drivers, Third Edition, February 2005

- By Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, O'Reilly
  http://lwn.net/Kernel/LDD3/
- Freely available on-line!
  Great companion to the printed book for easy electronic searches!
  Available as single PDF file



- LDD3 is current as of the 2.6.10 kernel (Old?)
- In **`exercises/pptx`**