# Connecting to Userspace

**Advanced Embedded Linux Development**

with **Dan Walkes**

University of Colorado **Boulder**

**Learning objectives:**

Creating and using device nodes

Registering and Unregistering Devices

# Linux Device Driver Examples

- “scull” driver overview
  - “Simple Character Utility for Loading Localities”
  - Acts on memory as if it were a device
- Benefits
  - Portable across architectures
  - Not hardware dependent
- Drawbacks
  - Not a “real” device driver
- See https://github.com/cu-ecen-5013/ldd3

# Types of “scull” drivers

- scull0 to scull3
  - Four devices with memory array global and persistent
    - Global: data shared with all file descriptors
    - Persistent: Not lost if device is closed and reopened.
- scullpipe0 to scullpipe3
  - FIFO pipes
  - Implement blocking and non blocking read and write.

# Types of “scull” drivers

- scullsingle
  - scull but only allows use for a single process
- scullpriv
  - scull private to each virtual console session
- sculluid and scullwuid
  - Opened only by one user at a time.

# /dev Directory Setup

- How do devices in the /dev directory map to appropriate kernel modules?
  - Device numbers/mknod
  - From Assignment 3:

- mknod <name> <type> <major> <minor>
  - Null device is a known major 1 minor 3
  - Console device is known major 5 minor 1

```
sudo mknod -m 666 dev/null c 1 3
```

# Device Numbers

- ls -l on the /dev directory shows two numbers
  - These are the “major” and “minor” device number
  - major is type - identifies the driver
  - minor identifies a specific device

```
crw-rw-rw-   1 root     root       1,   3 Apr 11  2002 null
crw-------   1 root     root      10,   1 Apr 11  2002 psaux
crw-------   1 root     root       4,   1 Oct 28 03:04 tty1
crw-rw-rw-   1 root     tty        4,  64 Apr 11  2002 ttys0
crw-rw----   1 root     uucp       4,  65 Apr 11  2002 ttyS1
crw--w----   1 vcsa     tty        7,   1 Apr 11  2002 vcs1
crw--w----   1 vcsa     tty        7, 129 Apr 11  2002 vcsa1
crw-rw-rw-   1 root     root       1,   5 Apr 11  2002 zero
```

# Common major/minor numbers

- See https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/devices.txt
  - Common device drivers use have dedicated major numbers (and sometimes minor numbers)
  - Everyone else allocates

```
8 block       SCSI disk devices (0-15)
                0 = /dev/sda          First SCSI disk whole disk
               16 = /dev/sdb          Second SCSI disk whole disk
               32 = /dev/sdc          Third SCSI disk whole disk
                  ...
              240 = /dev/sdp          Sixteenth SCSI disk whole disk
```

```
ecen5013@ecen5013-VirtualBox:~$ ls -la /dev/sd*
brw-rw---- 1 root disk 8,  0 Sep  2 08:21 /dev/sda
brw-rw---- 1 root disk 8,  1 Sep  2 08:21 /dev/sda1
brw-rw---- 1 root disk 8,  2 Sep  2 08:21 /dev/sda2
brw-rw---- 1 root disk 8,  5 Sep  2 08:21 /dev/sda5
brw-rw---- 1 root disk 8, 16 Sep  2 08:21 /dev/sdb
brw-rw---- 1 root disk 8, 17 Sep  2 08:21 /dev/sdb1
```

Linux Device Drivers 3rd Edition Chapter 3
https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/devices.txt

# Register/Unregister Example

- See the “sleepy” module for a simple example

```
int sleepy_init(void)
{
    int result;

    /*
     * Register your major, and accept a dynamic number
     */
    result = register_chrdev(sleepy_major, "sleepy", &sleepy_fops);
    if (result < 0)
        return result;
    if (sleepy_major == 0)
        sleepy_major = result; /* dynamic */
    return 0;
}

void sleepy_cleanup(void)
{
    unregister_chrdev(sleepy_major, "sleepy");
}

module_init(sleepy_init);
module_exit(sleepy_cleanup);
```

Linux Device Drivers 3rd Edition Chapter 3
https://github.com/cu-ecen-5013/ldd3/blob/master/misc-modules/sleepy.c

# unregister_chrdev

- What happens to a chrdev I register when my module exits?
  - You are responsible for unregistering
  - Use unregister_chrdev for this

```
/**
 * __unregister_chrdev - unregister and destroy a cdev
 * @major: major device number
 * @baseminor: first of the range of minor numbers
 * @count: the number of minor numbers this cdev is occupying
 * @name: name of this range of devices
 *
 * Unregister and destroy the cdev occupying the region described by
 * @major, @baseminor and @count.  This function undoes what
 * __register_chrdev() did.
 */
void __unregister_chrdev(unsigned int major, unsigned int baseminor,
                         unsigned int count, const char *name)
```

```
static inline void unregister_chrdev(unsigned int major, const char *name)
{
        __unregister_chrdev(major, 0, 256, name);
}
```

Linux Device Drivers 3rd Edition Chapter 3

https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h#L2684

# Device Number Representation

- dev_t type holds both, individual major or minor values accessed with macros
- Just a typedef for a 32 bit integer

```
#ifndef u32
#define u32 unsigned int
#endif
```

```
typedef u32 __kernel_dev_t;

typedef __kernel_fd_set         fd_set;
typedef __kernel_dev_t          dev_t;
```

Linux Device Drivers 3rd Edition Chapter 3
https://elixir.bootlin.com/linux/v5.8.7/source/include/linux/types.h#L16

# Device Number Representation

- Create a dev_t:

```
MKDEV(int major, int minor);
```

```
#define MINORBITS       20
#define MINORMASK       ((1U << MINORBITS) - 1)

#define MAJOR(dev)      ((unsigned int) ((dev) >> MINORBITS))
#define MINOR(dev)      ((unsigned int) ((dev) & MINORMASK))
#define MKDEV(ma,mi)    (((ma) << MINORBITS) | (mi))
```

- Extract major and minor numbers from dev_t:

```
MAJOR(dev_t dev);
MINOR(dev_t dev);
```

Linux Device Drivers 3rd Edition Chapter 3
https://elixir.bootlin.com/linux/v5.8.7/source/include/linux/kdev_t.h#L12

# Register/Unregister Example

- See the “scull” module for a more complex example
  - Register a range of devices
  - Handle errors after registration

```
int scull_init_module(void)
{
    int result, i;
    dev_t dev = 0;

    /*
     * Get a range of minor numbers to work with, asking for a dynamic
     * major unless directed otherwise at load time.
     */
    if (scull_major) {
        dev = MKDEV(scull_major, scull_minor);
        result = register_chrdev_region(dev, scull_nr_devs, "scull");
    } else {
        result = alloc_chrdev_region(&dev, scull_minor, scull_nr_devs,
                "scull");
        scull_major = MAJOR(dev);
    }
    if (result < 0) {
        printk(KERN_WARNING "scull: can't get major %d\n", scull_major);
        return result;
    }

    /*
     * allocate the devices -- we can't have them static, as the number
     * can be specified at load time
     */
    scull_devices = kmalloc(scull_nr_devs * sizeof(struct scull_dev), GFP_KERNEL);
    if (!scull_devices) {
        result = -ENOMEM;
        goto fail;  /* Make this more graceful */
    }

    return 0; /* succeed */

  fail:
    scull_cleanup_module();
    return result;
}
```

Linux Device Drivers 3rd Edition Chapter 3
https://github.com/cu-ecen-5013/ldd3/blob/master/scull/main.c

# /proc/devices

- Contains current list of allocated devices and associated driver
  - Dynamically registered devices will show up here with name specified in register_chrdev and allocated major number

```
ecen5013@ecen5013-VirtualBox:~$ cat /proc/devices
Character devices:
  1 mem
Block devices:
  7 loop
  8 sd
  9 md
```

# Dynamically assigning Devices

- Registering devices will allocate major/minor combinations we can use with mknod to create /dev file system entries.
- Can’t create device nodes (with mknod) in advance when using dynamic allocation.
- How can we handle dynamic allocation?
  - Parse /proc/devices after loading the module to find major number.
  - See misc-modules module_load for an example.
    - Other things to do at init time:
      - Set owner user and group
      - Set permissions on device

Linux Device Drivers 3rd Edition Chapter 3
https://github.com/cu-ecen-5013/ldd3/blob/master/misc-modules/module_load

# AWK

- Not an acronym - Abbreviation of developers names
- Works great when you have text output separated by a common field (for instance space)

```
major=$(awk "\$2==\"$module\" {print \$1}" /proc/devices)
```

```
Block devices:
  7 loop
  8 sd
```

- $2 == “sd” matches when the second field is sd
- print $1 prints the first field in this case.

https://en.wikipedia.org/wiki/AWK

# Dynamic Device Load Script Example

```
echo "Load our module, exit on failure"
insmod ./$module.ko $* || exit 1
echo "Get the major number (allocated with allocate_chrdev_region) from /proc/devices"
major=$(awk "\$2==\"$module\" {print \$1}" /proc/devices)
if [ ! -z ${major} ]; then
    echo "Remove any existing /dev node for /dev/${device}"
    rm -f /dev/${device}
    echo "Add a node for our device at /dev/${device} using mknod"
    mknod /dev/${device} c $major 0
    echo "Change group owner to ${group}"
    chgrp $group /dev/${device}
    echo "Change access mode to ${mode}"
    chmod $mode  /dev/${device}
else
    echo "No device found in /proc/devices for driver ${module} (this driver may not allocate a device)"
fi
```

```
aesd@aesd-VirtualBox:~/ldd3/misc-modules$ sudo ./module_load sleepy
Load our module, exit on failure
Get the major number (allocated with allocate_chrdev_region) from /proc/devices
Remove any existing /dev node for /dev/sleepy
Add a node for our device at /dev/sleepy using mknod
Change group owner to staff
Change access mode to 664
aesd@aesd-VirtualBox:~/ldd3/misc-modules$ cat /proc/devices  | grep sleepy
241 sleepy
aesd@aesd-VirtualBox:~/ldd3/misc-modules$ ls -la /dev | grep sleepy
crw-rw-r--   1 root    staff   241,   0 Feb 29 20:20 sleepy
```

Example Output

Allocated major 241

Device node created for 241 major, 0 minor

Linux Device Drivers 3rd Edition Chapter 3

https://github.com/cu-ecen-5013/ldd3/blob/master/misc-modules/module_load