VE482 — Introduction to Operating Systems Lab 9

Tasks

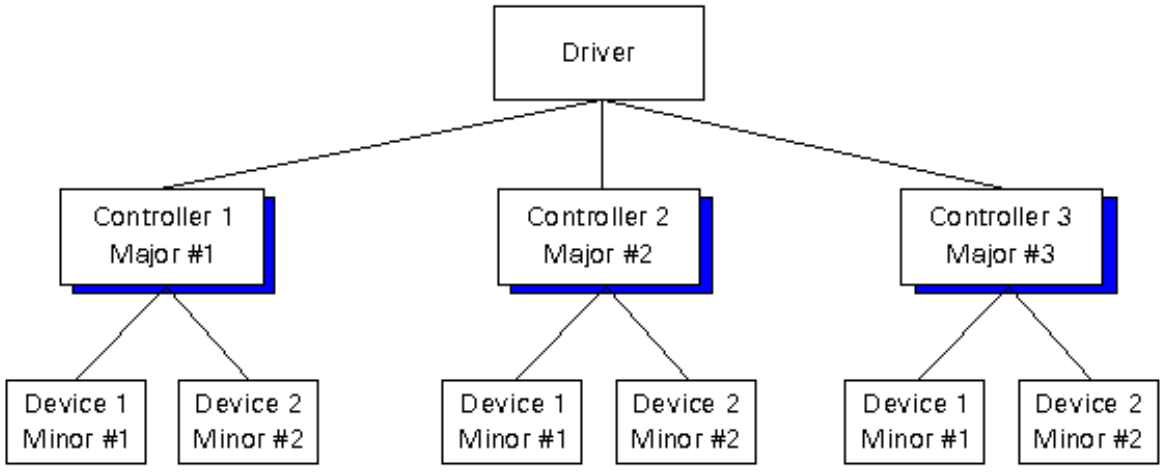
1. **What needs to be returned by read and write file operations for a character device?**

   The value returned by read or write can be:

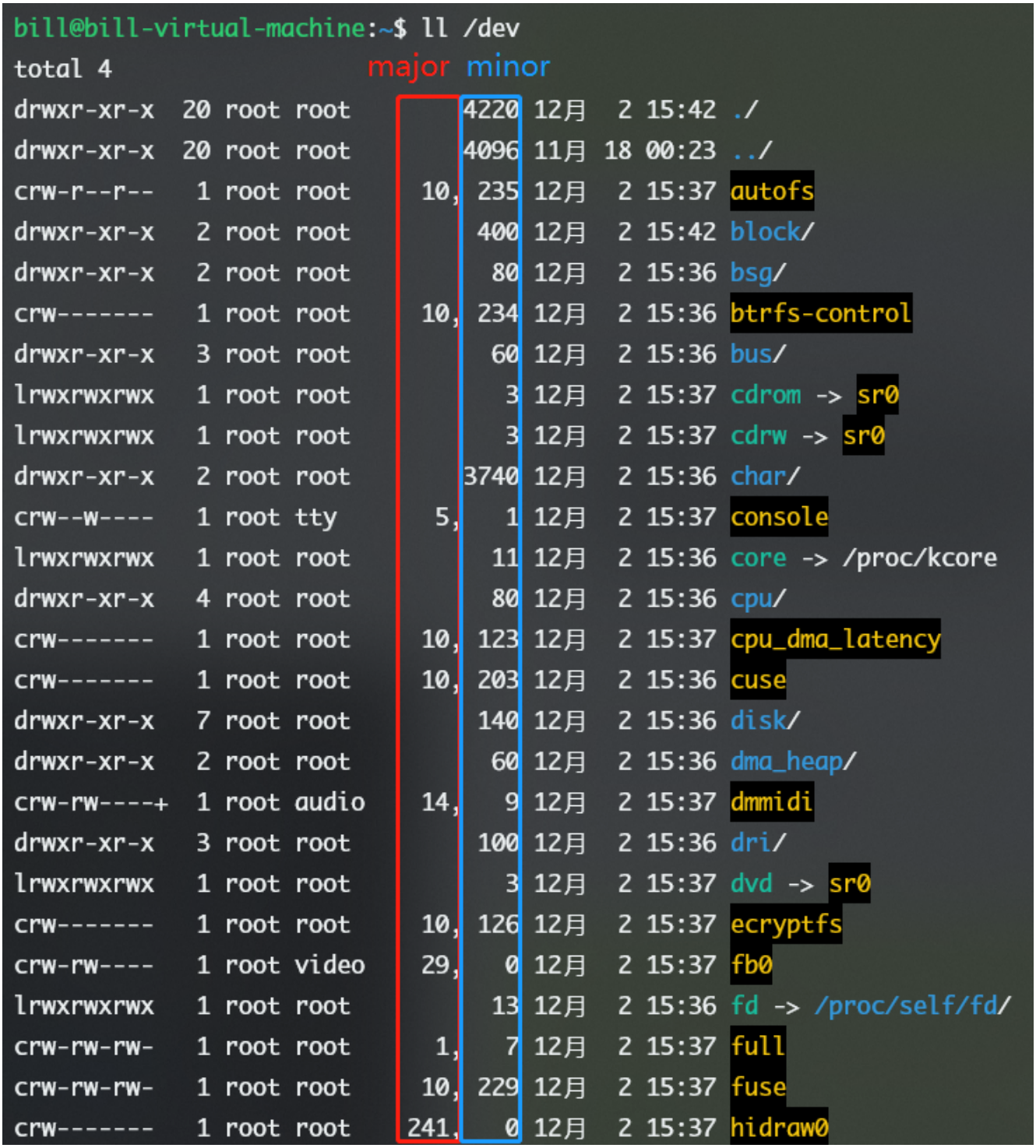
   > - the number of bytes transferred; if the returned value is less than the size parameter (the number of bytes requested), then it means that a partial transfer was made. Most of the time, the user-space app calls the system call (read or write) function until the required data number is transferred.
   > - 0 to mark the end of the file in the case of read ; if write returns the value 0 then it means that no byte has been written and that no error has occurred; In this case, the user-space application retries the write call.
   > - a negative value indicating an error code.

2. **How are exactly those major and minor numbers working? You vaguely remember that you can display them using `ls -l /dev`.**

   Major and **minor** numbers are associated with the device special files in the `/dev` directory and are used by the operating system to determine the actual driver and device to be accessed by the user-level request for the special device file.



In `/dev`, we can see the major and minor numbers of different devices



3. **Knowing the major number and minor numbers of a device, how to add a character device to `/dev`?**

```
# (Create a new device directly)
mknod /dev/$device_name c $major_id $minor_id

# (add a character device to /dev)
cdev_add(struct cdev *dev, dev_t num, unsigned int count);
```

4. **Where are the following terms located in Linux source code?**

term	location
module_init	include/linux/module.h
module_exit	include/linux/module.h
printk	include/linux/printk.h
container_of	include/linux/kernel.h
dev_t	include/linux/types.h
MAJOR	include/linux/kdev_t.h
MINOR	include/linux/kdev_t.h
MKDEV	include/linux/kdev_t.h
alloc_chrdev_region	include/linux/fs.h
module_param	include/linux/moduleparam.h
cdev_init	include/linux/cdev.h
cdev_add	include/linux/cdev.h
cdev_del	include/linux/cdev.h
THIS_MODULE	include/linux/export.h

5. **How to generate random numbers when working inside the Linux kernel? You think that a while back you read something about getting the current time.**

   The primary kernel interface is

   ```
   void get_random_bytes(void *buf, int nbytes)
   ```

   This interface will return the requested number of random bytes, and place it in the requested buffer. This is equivalent to a read from `/dev/urandom`.

   For less critical applications, there are the functions:

   `u32 get_random_u32()`

   `u64 get_random_u64()`

   `unsigned int get_random_int()`

   `unsigned long get_random_long()`

These are produced by a cryptographic RNG seeded from get_random_bytes, and so do not deplete the entropy pool as much. These are recommended for most in-kernel operations *if the result is going to be stored in the kernel*.

6. **How to define and specify module options?**

   We can use the function `module_param(name, type, perm)` to pass a parameter to the module. In `linux/moduleparam.h`, the usage of it is described as follows:

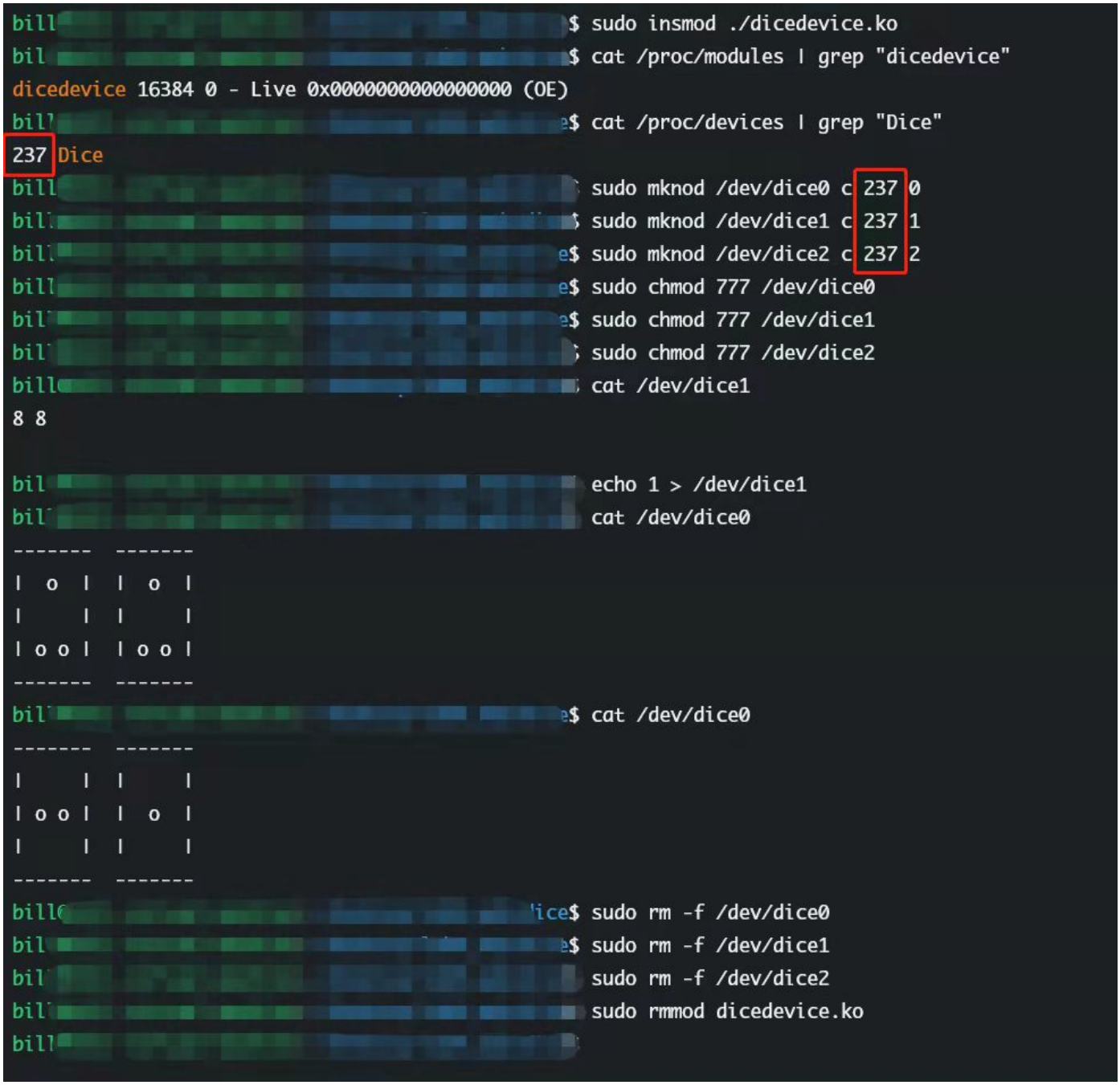
   - `name`: The variable to alter, and exposed parameter name.
   - `type`: The type of the parameter. Standard types include `byte`, `hexint`, `short`, `ushort`, `int`, `uint`, `long`, and `ulong`.
   - `perm`: The visibility in sysfs. It is 0 if the variable is not to appear in sysfs, or 0444 for world-readable, 0644 for root-writable, etc.
     Example:

```
// In dice.c
#include <linux/moduleparam.h>

int gen_sides = 20;
module_param(gen_sides, int, 0644);
```

```
# During installation
insmod dice.ko gen_sides = 8
```

Test Result



Reference

[1] return value of read and write https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html

[2] major and minor numbers http://osr600doc.sco.com/en/HDK_concepts/ddT_majmin.html

[3] Linux source code https://elixir.bootlin.com/linux/v5.15.6/source