chardev.c Makefile

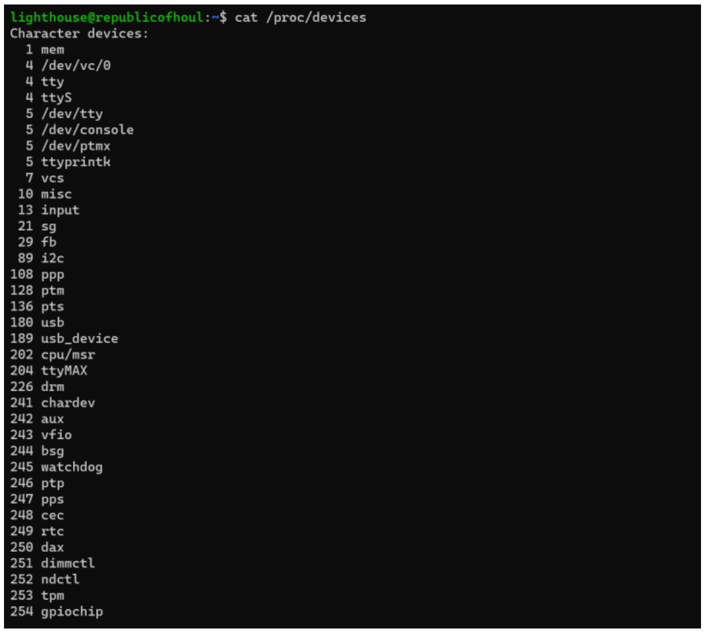
chardev.c

```c
/*
 * chardev.c: Creates a read-only char device that says how many times
 * you have read from the dev file
 */
#include <linux/cdev.h>
#include <linux/delay.h>
#include <linux/device.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/irq.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/poll.h>

/* Prototypes - this would normally go in a .h file */
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char __user *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char __user *, size_t, loff_t *);

#define SUCCESS 0
#define DEVICE_NAME "chardev" /* Dev name as it appears in /proc/devices */
#define BUF_LEN 80 /* Max length of the message from the device */

/* Global variables are declared as static, so are global within the file. */

static int major; /* major number assigned to our device driver */

enum {
    CDEV_NOT_USED = 0,
    CDEV_EXCLUSIVE_OPEN = 1,
};

/* Is device open? Used to prevent multiple access to device */
static atomic_t already_open = ATOMIC_INIT(CDEV_NOT_USED);

static char msg[BUF_LEN + 1]; /* The msg the device will give when asked */

static struct class *cls;

static struct file_operations chardev_fops = {
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release,
};

static int __init chardev_init(void)
{
    major = register_chrdev(0, DEVICE_NAME, &chardev_fops);

    if (major < 0) {
        pr_alert("Registering char device failed with %d\n", major);
        return major;
    }
    pr_info("I was assigned major number %d.\n", major);
    cls = class_create(THIS_MODULE, DEVICE_NAME);
    device_create(cls, NULL, MKDEV(major, 0), NULL, DEVICE_NAME);

    pr_info("Device created on /dev/%s\n", DEVICE_NAME);

    return SUCCESS;
}

static void __exit chardev_exit(void)
{
    device_destroy(cls, MKDEV(major, 0));
    class_destroy(cls);

    /* Unregister the device */
    unregister_chrdev(major, DEVICE_NAME);
}

/* Methods */

/* Called when a process tries to open the device file, like
 * "sudo cat /dev/chardev"
*/
static int device_open(struct inode *inode, struct file *file)
{
    static int counter = 0;

    if (atomic_cmpxchg(&already_open, CDEV_NOT_USED, CDEV_EXCLUSIVE_OPEN))
        return -EBUSY;

    sprintf(msg, "I already told you %d times Hello Hou!\n", counter++);
    try_module_get(THIS_MODULE);

    return SUCCESS;
}

/* Called when a process closes the device file. */
static int device_release(struct inode *inode, struct file *file)
{
/* We're now ready for our next caller */
    atomic_set(&already_open, CDEV_NOT_USED);

/* Decrement the usage count, or else once you opened the file, you will
* never get rid of the module.
*/
    module_put(THIS_MODULE);

    return SUCCESS;
}

 /* Called when a process, which already opened the dev file, attempts to
 * read from it.
 */
static ssize_t device_read(struct file *filp, /* see include/linux/fs.h */
    char __user *buffer, /* buffer to fill with data */
    size_t length, /* length of the buffer */
    loff_t *offset)
{
/* Number of bytes actually written to the buffer */
    int bytes_read = 0;
    const char *msg_ptr = msg;

    if (!*(msg_ptr + *offset)) { /* we are at the end of message */
        *offset = 0; /* reset the offset */
        return 0; /* signify end of file */
    }

    msg_ptr += *offset;

    /* Actually put the data into the buffer */
    while (length && *msg_ptr) {
    /* The buffer is in the user data segment, not the kernel
    * segment so "*" assignment won't work. We have to use
    * put_user which copies data from the kernel data segment to
    * the user data segment.
    */
    put_user(*(msg_ptr++), buffer++);
    length--;
    bytes_read++;
}

    *offset += bytes_read;

    /* Most read functions return the number of bytes put into the buffer. */
    return bytes_read;
}

    /* Called when a process writes to dev file: echo "hi" > /dev/hello */
    static ssize_t device_write(struct file *filp, const char __user *buff,
    size_t len, loff_t *off)
    {
        pr_alert("Sorry, this operation is not supported.\n");
        return -EINVAL;
    }

module_init(chardev_init);
module_exit(chardev_exit);

MODULE_LICENSE("GPL");
```

Makefile

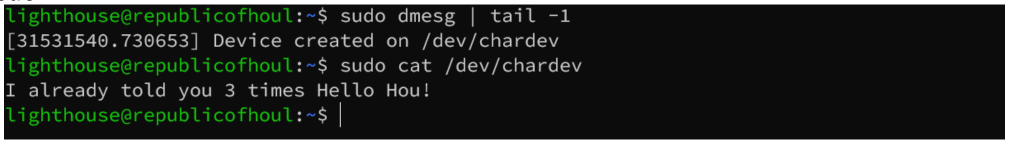
```
obj-m += chardev.o

PWD := $(CURDIR)

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
cat /proc/devices
```



```
lighthouse@republicofhoul:~$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  5 ttyprintk
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
 89 i2c
108 ppp
128 ptm
136 pts
180 usb
189 usb_device
202 cpu/msr
204 ttyMAX
226 drm
241 chardev
242 aux
243 vfio
244 bsg
245 watchdog
246 ptp
247 pps
248 cec
249 rtc
250 dax
251 dimmctl
252 ndctl
253 tpm
254 gpiochip
```

dmseg output



```
lighthouse@republicofhoul:~$ sudo dmesg | tail -1
[31531540.730653] Device created on /dev/chardev
lighthouse@republicofhoul:~$ sudo cat /dev/chardev
I already told you 3 times Hello Hou!
lighthouse@republicofhoul:~$
```