Device Driver

CS453: Operating systems

Overview

In this assignment, we will write a simple character driver called **booga**. Please do a **git pull --rebase** in your git repo to get the **booga** project folder that contains the starter code and scripts for this project. Note that, you will only be able to test this assignment on your own VM as you will need to have *root* access to complete this assignment.

Important notes

You **MUST** build against the Kernel version installed on **onyx**. If you are unclear what version to build against please ask do NOT just assume. The only time you will need to use root in this project is to load and unload the drivers. You should not use the root account for building, test, etc.

Specification

The main driver

The **booga** driver is a simple character driver that supports the **open**, **read** and **write** and **close** operations. The driver supports four minor numbers: 0, 1, 2, and 3. The device files are: `/dev/booga0`, `/dev/booga1`, `/dev/booga2`, `/dev/booga3`. We will also create a link from `/dev/booga` to `/dev/booga0`, so that acts as the default device when someone accesses `/dev/booga`. On reading from `/dev/booga0`, `/dev/booga1`, `/dev/booga2` and `/dev/booga3` the driver gives a stream of one of the following phrases:

- `booga! booga!`
- `googoo! gaagaa!`
- `neka! maka!`
- `wooga! wooga!`

Note that the driver may only output part of a phrase if the number of bytes requested by the user is not a multiple of the length of the chosen phrase. Also each new phrase is separated from the previous one by a single space.

The driver is unpredictable as to what phrase it says. In order to simulate this behavior, we will use a random number generator. However, we cannot call any standard C library

functions in the kernel (e.g. random()). Linux actually has a device driver that generates strong random numbers by observing hardware noise, interrupts and other system activity. In the user space, the random device driver can be accessed via `/dev/random`. In your device driver code, you will need to include the header file `<linux/random.h>` and the function that you will call has the following prototype.

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

Here is a sample code on how to use the above function.

```
char randval;
get_random_bytes(&randval, 1);
choice = (randval & 0x7F) % 4; /* bitwise AND is to guarantee
positive numbers */
```

On writing to booga devices /dev/booga0, /dev/booga1, /dev/booga2, the booga device driver works like /dev/null (so it pretends to write the buffer but doesn't actually write to any device). However if a process tries to write to /dev/booga3, it suffers from sudden death! You cannot call the system call `kill()` from inside the kernel space so you will have to figure out how to terminate a process from inside the kernel. Search the kernel sources for ideas. Use the **SIGTERM** signal instead of **SIGKILL** for terminating the process.

Get booga stats from /proc

Create **/proc** entries for the booga driver. It reports on the number of bytes read/written since the driver was loaded, number of times it was opened from each supported minor number, and the number of times each of the four strings was selected. So if we output 1000 characters with the string "booga! booga!" in it, then we count that as one more instance of the phrase for this purpose. If the driver outputs an incomplete phrase, we will still count it as another instance of that phrase being selected. Here is a sample output.

```
[user@localhost booga]# cat /proc/driver/booga
bytes read = 300
bytes written = 200
number of opens:
  /dev/booga0 = 4 times
  /dev/booga1 = 0 times
  /dev/booga2 = 1 times
  /dev/booga3 = 1 times
strings output:
  booga! booga!  = 0 times
  googoo! gaagaa!  = 1 times
  neka! maka!  = 1 times
  wooga! wooga!    = 1 times
```

Thread-safety

Protect the updating of structure used to track the statistics using semaphores. Search the kernel code base (or the `examples/device-management/linux-device-drivers/example4` under the class examples repository) for an example on how to use semaphores.

Notes

Here are the prototypes of the functions that your driver would need to implement.

```
static int booga_open (struct inode *inode, struct file *filp);
static int booga_release (struct inode *inode, struct file *filp);
static ssize_t booga_read (struct file *filp, char *buf, size_t count, loff_t *f_pos);
static ssize_t booga_write (struct file *filp, const char *buf, size_t count, loff_t *f_pos);
static int __init booga_init(void);
static void __exit booga_exit(void);
```

Remember that you need to use the `__copy_to_user(...)` kernel function to copy the data to user space.

Note that the print messages will not show on the screen if you are testing under X Windows. The messages are, however, logged in the file /var/log/messages. You can open another terminal and watch the output to the system messages file with the command:

```
tail -f /var/log/messages
```

In some versions, you may not have this file setup. Alternatively, you can use the command `dmesg --follow` to watch for kernel log messages.

Another approach would be to invoke a virtual system console. Under Linux, you can use the following keys:

```
Ctrl-Alt-F2 to use a virtual test console
Ctrl-Alt-F1 to go back to the X Windows console
```

In the virtual system console, the system messages will show up on screen.

We have provided a test program called **test-booga.c**. We will use this program to test the driver. While testing your program, the following situations may arise:

- The system hangs and you have to hit the reset button
- The system spontaneously reboots. Oops!
- The kernel prints an **oops** message (a milder version of the infamous General Protection Fault). You will get a stack trace with it that should help in narrowing down the cause

Here is a sample session with the booga driver. Note that the characters returned by the driver do not contain any newline characters. The output shown below was reformatted slightly for this document.

```
[user@localhost]$ make
make -C /lib/modules/`uname -r`/build M=`pwd` modules
make[1]: Entering directory '/usr/src/kernels/4.14.11-200.fc26.x86_64'
 CC [M]
/home/user/Documents/classes/cs453/github/CS453/projects/p5/grader/solutions/u
ser/booga.o
 Building modules, stage 2.
 MODPOST 1 modules
 CC
/home/user/Documents/classes/cs453/github/CS453/projects/p5/grader/solutions/u
ser/booga.mod.o
 LD [M]
/home/user/Documents/classes/cs453/github/CS453/projects/p5/grader/solutions/u
ser/booga.ko
make[1]: Leaving directory '/usr/src/kernels/4.14.11-200.fc26.x86_64'
cc    -c -o test-booga.o test-booga.c
cc  -o test-booga test-booga.o

[user@localhost]$ ls
booga.c   booga_load   booga.o         booga_unload   Module.symvers
test-booga.c
booga.h   booga.mod.c  booga-test1.sh  Makefile       README.md
test-booga.o
booga.ko  booga.mod.o  booga-test2.sh  modules.order  test-booga
[user@localhost ]$

[user@localhost]$ sudo ./booga_load
[sudo] password for user:

[user@localhost]$ /sbin/lsmod
Module                  Size  Used by
booga                  16384  0
bluetooth             593920  0
ecdh_generic           24576  1 bluetooth
rfkill                 28672  2 bluetooth
```

```
ipt_MASQUERADE         16384  1
nf_nat_masquerade_ipv4    16384  1 ipt_MASQUERADE
. . .

[user@localhost]$ ./test-booga 0 100 read
Read returned 100 characters
wooga! wooga! wooga! wooga! wooga! wooga! wooga! wooga! wooga! wooga! wooga!
wooga! wooga! wooga! Wo

[user@localhost]$ ./test-booga 0 100 read
Read returned 100 characters
googoo! gaga! googoo! gaga! googoo! gaga! googoo! gaga! googoo! gaga! googoo!
gaga! googoo! gaga! go

[user@localhost]$ ./test-booga 0 100 write
Attempting to write to booga device
Wrote 100 bytes.

[user@localhost]$ ./test-booga 3 100 write
Attempting to write to booga device
Terminated

[user@localhost]$ cat /proc/driver/booga
bytes read = 200
bytes written = 100
number of opens:
 /dev/simple0 = 3 times
 /dev/simple1 = 0 times
 /dev/simple2 = 0 times
 /dev/simple3 = 1 times
strings output:
 booga! booga!   = 0 times
 googoo! gaga!   = 1 times
 wooga! wooga!   = 1 times
 neka! maka!     = 0 times

[user@localhost]$ sudo ./booga_unload
```

Submission

Files committed to git (backpack)

Required files to submit through git (backpack), if you created more helper files that is fine.

1. All files that are required to build and run your project
2. README.md

Push your code to a branch for grading

Run the following commands

1. make clean
2. git add <file ...> (on each file!)
3. git commit -am “Finished project”
4. git branch booga
5. git checkout booga
6. git push origin booga
7. git checkout master

Check to make sure you have pushed correctly

- Use the command **git branch -r** and you should see your branch listed (see the example below)

```
$ git branch -r
  origin/HEAD -> origin/master
  origin/master
  origin/booga
```

Submit to Blackboard

You must submit the sha1 hash of your final commit on the correct branch. Your instructor needs this in order to troubleshoot any problems with submission that you may have.

- git rev-parse HEAD

Grading Rubric

Provided via backpack.