OPERATING SYSTEM
TD 5: Kernel module and special files

Global instructions

Sources for part 2 and 3 are available on the web page: `http://adrien.krahenbuhl.fr/en/teaching/`.

All this TD requires administrator rights (root) to modify the Linux kernel. Check your code before loading your modules because a kernel crash can theoretically have serious consequences on your installation: disk corruption in the worst case. In any case, it is better to reboot the crashed machine every time a problem occurs in your kernel module. Use a Virtual Machine (like KVM or VirtualBox) is a good idea for this TD: you will not have to reboot the host machine.

1 Kernel modules

(1.1) Display the module list with the `lsmod` command. Load the `uio` module (or another one not yet loaded) with:

```
sudo /sbin/insmod/lib/modules/2.6.32-5-686/kernel/drivers/uio/uio.ko
```

Observe its appearing in `lsmod`. Unload it and watch it disappear in `lsmod`.

(1.2) Look at the references counter and the dependencies of the various modules in the right columns of the `lsmod` output. Find in the output of `lsmod` a module A that has no reference or dependency, and another module B that has a single reference and that has A in its dependency list. Try to unload B with `rmmod B`. Then unload A and observe the evolution of `lsmod` output. Try again to unload B.

In which direction are the dependencies indicated and what are their links with the reference counter?

(1.3) In the real life, you never need to specify the full path nor the dependencies: you can use `modprobe` (which calls `insmod` or `rmmod` internally). Run `/sbin/modinfo <name>` on modules with and without dependencies (for example the modules A and B in the previous question) to find their installation path and dependencies. Use `modprobe <name>` to load a module with its dependencies all at once (for example module A).

2 Create a kernel module

(2.1) Compile the modules provided with `make`, then load the `hello.ko` module and check with `lsmod` that it is loaded. If `sudo insmod hello.ko` fails with `Permission denied` error, pass root with `sudo su` before doing `insmod hello.ko`

(2.2) Observe the code of `hello.c`. it asks to the kernel to display a message at loading and unloading time. Observe these messages (and all other kernel messages) by running:

```
dmesg | less
```

(2.3) Add other messages in `hello.c`, recompile it then reload it to confirm that your changes appear in `dmesg`.

3 Special Files

In all these tests, we can follow the evolution of operations in `dmesg`.

(3.1) Load the `special.ko` module compiled in the same time than `hello.ko`. It will create a special file `/dev/special`. Compare this file with the information given by `modinfo`. Run `cat /dev/special` and explain what you are watching.

Run the two following commands:

OPERATING SYSTEM
TD 5: Kernel module and special files

```
echo -n toto> /dev/special
cat /dev/special
```

The `-n` option avoids line breaking at the end of `toto`. Explain again.

(3.2) Modify the `special.c` module to manage reads and writes in the buffer `buffer`. Make sure you respect `LENGTH`), you return the correct number of read or written characters, and you update the value of the progress pointer in the `*offp` file.

Create a small text file, copy it to `/dev/special` and check the file content. Try again with a long text file (longer than `LENGTH`) and make sure it is properly truncated. In order to check that the reading and writing piece by piece is working, you can use:

```
dd if=/dev/special of=myfile bs=100 count=20 # reading
dd if=myfile of=/dev/special bs=100 count=20 # writing
```

(3.3) Modify the module in order that each character read is incremented by `1`. The kernel buffer remains unchanged, but we read `bcd` if it actually contains `abc`. What more useful operation could we imagine to apply in place of this incrementation?

(3.4) Modify the `special.c` module in order that each character have the following individual behavior:

- ✓ initialization with the character `'0'`
- ✓ a writing increments the corresponding character in the array and ignores the written value
- ✓ a reading returns the character stored in the array and then reset it to `'0'`.

Example with an array of size 4:

```
Initialization                  Buffer = 0000
Writing abc at the beginning             1110
Writing ab at offset 2                   1121
Writing abcd at the beginning            2232
Reading 1 character at beginning         0232   return 2
Reading 4 characters at the beginning    0000   return 0232
```

Check that this allows to create a *special* file whose the behavior is completely different from a normal file. What could be the use of this file?

(3.5) Modify the `special.c` module to be able to dynamically adjust the length of the buffer by an `ioctl`. We will take care to protect the module from concurrent accesses during the settings of the length by using the mutex.