Report on exercise #2

Matteo Corain S256654

Laboratory #5 – System and device programming – A.Y. 2018-19

The given `chardev_SDP_lab` module implements a driver for a simple character-oriented device which internally buffers the last characters written on the device and returns them when performing a read operation. The original version of the driver is such that, when accessed using the provided `test_chardev` program, the results are (sort of) correct (some remainders of the previous writes on the buffer can be present when reading from the device anyway), while if we try to access the device by using a succession of `echo` and `cat` operations, this last one fails entering in an infinite loop.

To understand this behavior, we have to analyze what actions are actually performed by the `cat` program when our device is passed to it. There are essentially two problems with the provided version of the driver, that prevent it from working with the `cat` utility:

- First, when `cat` accesses the device associated with our driver, it performs read operations in blocks of 128 kB (131072 bytes); this dimension, passed to the `device_read()` function of the driver and then to the `copy_to_user()` system call, is much higher than the length of the internal buffer of the driver (`BUF_LEN`, set to 128 bytes) and, as a consequence, when we call `copy_to_user()` a buffer overflow error is arisen by the kernel;
- Second, `cat` continues to perform read operations on the given file until the returned number of read bytes is `0`, meaning the end of the file is reached; in the current implementation, however, the internal buffer is always returned from the beginning (the returned length is always equal to the original `count`), and therefore an infinite number of reads are performed.

As a consequence of these two problems, the cat utility does not show anything on video and the following message is printed on the kernel log, together with the call trace which generated the error:

```
[  274.038599] Buffer overflow detected (128 < 131072)!
```

Those problems do not verify when using the `test_chardev` program, for which the length passed to the read operations is limited to the length of the user-space buffer allocated by the program (`BUFFER_SIZE`, set to 100 bytes), which is shorter than the internal buffer of the device and therefore no buffer overflow ever happens; additionally, there is no continuous loop which performs reads until the “end of file” (a single read is performed on the device), so no infinite looping is performed. However, also with the `test_chardev` program, we have the problem that, if we write on our device a string which is shorter than the one we have previously written, the last characters of the returned buffer reflect the ones of the longest string (they are not overwritten by subsequent writes).

To solve the aforementioned problems, a number of modifications have been introduced to the given module, principally focusing on the `device_read()` and `device_write()` functions called when the device is respectively accessed for read or write operations.

For avoiding the buffer overflow problem, a check on the number of bytes to read has been placed at the beginning of both the `device_read()` and `device_write()` functions. This is used to limit the number of bytes to be written to the internal buffer to a maximum of `BUF_LEN` and the number of bytes to be read from the internal buffer to the length of the string stored in the internal buffer, retrieved via the macro `strlen()` defined in the header file `linux/string.h`. In this way, when the subsequent copy operation is performed, there is no risk that a buffer overflow error could arise.

After having corrected the first problem, if we use `cat` on our device, we can see that the content of the buffer is correctly accessed, but the problem is that a value higher than zero is always returned by the `read()` operations performed by the utility. This is interpreted as if the end of the file is never reached, and therefore the operation is repeated indefinitely (the content of the buffer is printed multiple times on screen).

In order to make the behavior of the driver compliant with the behavior of the `cat` program (read until `read()` returns zero), the read operation has been modified to return the content of the buffer not from the beginning of the buffer, but instead from the last position on which the read operation has been performed. This has been achieved by introducing a variable `buf_ptr`, of type `char*`, that is used in the following way:

- Whenever a write operation is performed, `buf_ptr` is set to the beginning address of the internal buffer (`buf_ptr = char_dev_buf`), so that the following read operations will start reading the buffer from the beginning;
- Whenever a read operation is performed, the `copy_to_user()` system call is executed on `buf_ptr` rather than on the address of the beginning of the buffer and for a number of bytes equal to the length of the string contained in the address given by `buf_ptr`; finally, after each read operation, `buf_ptr` is incremented by the length of the string returned to the user.

In this way, after each write operation, the buffer pointer is advanced until, at some point, it will point to a location in the buffer where no string is actually present; in this case, the `read()` will return `0` and the cat utility will terminate.

Finally, in order to avoid problems when writing on the device strings which are shorter than the ones previously written, a `memset()` call has been placed at the beginning of each write operation, which sets the entire buffer's contents to `0` (string termination character). In this way, when we perform a `write()` on our device, it is assured that no remainders from the previous writes will be present in the buffer.

The functionalities of the modified driver have been tested by using the provided sequence of command, generating the output below.

```
# echo something > /dev/chardev_SDP_lab
# cat /dev/chardev_SDP_lab
something
# echo xxx > /dev/chardev_SDP_lab
# cat /dev/chardev_SDP_lab
xxx
```