CEng 536 Advanced Unix
Fall 2018
Kernel Programming Assignment 2
Due: 21/1/2019

1 Description

In this project you are going to implement a pseudo character device driver for Linux kernel ressembling a multi-channel reminder service. Readers of the character device can subscribe channels from 0 to 15 and their read() calls will read only when a message with one of the subscribed channel is written. Writers can change their channels and all writes go to that specific channel. However they can describe a delay factor in milliseconds so that their write will be effective after the delay.

The device will be named reminder and all minor devices will work as an isolated service with same functionality.

Channel sets are defined by a 16 bits integer. Least significant bit is the channel 0 and most significant bit is the channel 15. Existence of a bit means that that channel is included in the read. This set cannot be set to 0 which means nothing to read. But if it is set to `0xffff`, all messages will be read.

Writers have first character in message reserved for the channel number between 0 to 15, so they write on channel at a time. If this number is not in the range, it is simply bitwise anded by `0xff` to truncate without error. When writers write `n` bytes, readers can read `n-1` bytes message. If there are multiple readers of a channel, data is duplicated so all of them can read it. The message will be duplicated at time of the write (in other words after the delay). The readers subscribed to a channel can read messages that are written after the subscription. If they have unread messages when they unsubscribed, they are discarded.

Writes will not block, the buffer of the channels grow indefinitely. Reads block until requested number of bytes are read. There is no message boundary, each read will continue from where the previous read is left.

A typical write is:

```
#include<stdint.h>

struct writebuf {
        char channel;
        char content[];
} *wbuf;

/* user space write n bytes message on write set 0xaaaa */
fd = open("/dev/reminder0", O_RDWR);
wbuf = malloc(4098);
memcpy(wbuf->content, message, n);
wbuf->channel = 10;
write(fd, wbuf, n+1);
```

The largest buffer to write in a single write is bounded by 4096 bytes including channel number. Similarly reads are bounded by 4095. Larger reads read first 4095 bytes, larger writes are truncated and writes only first 4096 byte. Your device should implement `ioctl()` and `poll()` (only for read) functions. Following ioctl calls needs to be implemented:

```
#include<linux/ioctl.h>

#define REMIND_IOC_MAGIC  'd'

#define REMIND_IOCTSETSUBS _IO(REMIND_IOC_MAGIC, 1)
#define REMIND_IOCQGETSUBS _IO(REMIND_IOC_MAGIC, 2)
#define REMIND_IOCTSETDELAY _IO(REMIND_IOC_MAGIC, 3)
#define REMIND_IOCQGETDELAY _IO(REMIND_IOC_MAGIC, 4)
#define REMIND_IOCQGETPENDING _IO(REMIND_IOC_MAGIC, 5)
```

```
#define REMIND_IOC_MAXNR 5
```

REMIND_IOCTSETSUBS
: It sets the subscription set of a reader. Least significant 16 bits of the `ioctl()` parameter is taken as the new set.

REMIND_IOCTGETSUBS
: It gets the current subscriptions set of a reader as return value of `ioctl()`. Default subscription value is 1 for subscription of channel 0 only.

REMIND_IOCTSETDELAY
: It sets the delay for a writer in milliseconds (1/1000 secs). Default value is 0, no delay. If this value is negative, an `EINVAL` error is returned.

REMIND_IOCQGETDELAY
: It gets the delay for a writer.

REMIND_IOCQGETPENDING
: It computes and returns the number of messages to be delivered to a reader in future. It it sum of all delayed writes on the channels that are subscribed.

Some of the operations require device is opened for reading or writing. In case an invalid operation is send (like `REMIND_IOCQGETDELAY` for a read only opened device), `EACCESS` should be returned.
All kernel structures should be protected against race conditions.

2 Implementation

Download and change one of the scull devices under:
`https://github.com/jesstess/ldd4.git` You can use ideas from scullc and scull/pipe.c.
Only module parameter you need is `numminors` which has 8 as default:
`modprobe reminder numminors=4`
For executing a function with a delay, you need `struct delayed_work` structure and `INIT_DELAYED_WORK()` macro. This will register a function to be called after given number of time. The function should get a `struct work_struct *` value which can be converted into `delayed_work` by `to_delayed_work(wspointer)`. Using `container_of` trick, you can associate function with data.
You can use your favorite Linux distribution as long as kernel is new ($>$ 4.0). Install `linux-headers` package, compile and test your device.
Your module should allocate a character device (dynamic major number) and register the following operations:
`open(), read(), write(), release(), unlocked_ioctl(), poll`

3 Submission and External libraries

You need to submit a 'tar.gz' archive (no zip) containing scull like Makefile and your source. Makefile, a .c file, a .h file, load and unload scripts are sufficient.
Please ask all questions to:
`news://news.ceng.metu.edu.tr:2050/metu.ceng.course.536/`
`https://cow.ceng.metu.edu.tr/News/thread.php?group=metu.ceng.course.536`