

Practical 4 - Minix I/O and Device drivers

This prac will let you explore how Minix IO works, as well as how device drivers are structured. This prac will require you to search the code in more places than just those explicitly mentioned - don't be afraid to do some looking!

Interrupt Handling

Interrupts are handled initially by the operating system and passed on to appropriate software for other handling. The lowest level of interrupt handling is in `/kernel/mpx386.s`, an assembly file.

- Find where the handlers are defined and implemented.
- What tasks are done here in the assembly handler definition before control is handed over to the higher-level software?
- What is done in `/kernel/i8259.c:intr_handle` - what function is called from here? *This is impossible to work out just from the function - you will first need to find out which functions are assigned to handle each interrupt (put_irq_handler registers a function to handle an interrupt - try finding out where this function is called from, and with what parameters).*
- Examine the keyboard interrupt handling code. Which function claims to handle the interrupt... how does this function get assigned as the handler - what system call is used to perform this function?

Hopefully you will have explored a sizeable chunk of code in attempting to understand how all the pieces fit together to perform the interrupt handling. It's not a simple task!

Device drivers

Now that we've examined something about the interrupt handlers, we will have a look at how a device driver works. As part of this, we will implement a new device in the system. The memory driver is the simplest of the drivers, so this is what we will work with. The memory driver is used for devices such as `/dev/zero` - we will create a new device, `/dev/number`, which produces a number between 0-255 when read, and is specified using the `ioctl` system call

Exploring the code

- The driver is part of the boot image - have a look at `/kernel/main.c` to see where it is started up when the system boots. Again, this might take a little work to find out where it happens.
- When the driver first runs, the main function is run (within `/drivers/memory/memory.c`). This includes an initialisation function. What does this initialisation do?
- Following initialisation, the driver jumps to `/drivers/libdriver/driver.c:driver_task`. This is the common file for all device drivers - examine the code and work out what the main loop in this function does. Make sure you can see where messages are handled and replied to.
- What functions does the memory driver support? How does it perform them? What system calls (and associated kernel calls) are needed to support these functions?

Adding /dev/number

In order to add a new device there are a few steps to be taken:

- `/include/minix/dmap.h` defines the major device numbers for a few of the drivers, along with minor device numbers for the memory driver devices. You will need to add an appropriate entry here.
- `/include/sys/ioc_memory.h` defines the `ioctl` command codes for the memory driver, add an appropriate line here
- `/drivers/memory/memory.c` is the main file for the driver. There are a few places to add code here:
 - `NR_DEVS` defines the number of minor devices - update this.

- `ZERO_BUF_SIZE` is a buffer for the `/dev/zero` device. You will need to create a similar buffer for `/dev/number`
- `m_transfer` is the function which performs the data transfer from the driver to the user process (via a kernel call). You will need to add some code in the switch statement here (have a look at how `ZERO_DEV` works for hints).
- `m_ioctl` is the function which performs the command upon the open file. You will need to add some code in the switch statement here (have a look at how `MIOCRAMSIZE` works)
- Initialize your buffer in `m_init`
- Build the new image and boot into it
- You should now be able to create the new device at the command line - "`mknod /dev/number c 1 7`" is the command to use.
 - `mknod` makes a new special file in the filesystem at the specified location,
 - `c` specifies the new device to be a character device
 - `1` and `7` specify the major and minor device numbers.
- Write a simple application which opens your new device and reads from it.

Summary

If you've answered the questions above and completed the code for `/dev/number`, you should have examined a sizeable chunk of the source code related to IO and one driver in particular. Working out how this code works and how the pieces of the operating system fit together can be difficult, but should give you a deeper understanding of how the system functions.