

CSIS0230A Principle of Operating Systems (Class A)

Assignment 4

Morse-code device driver

Tutor: {llcheng, kmcheung}@csis.hku.hk, Deadline Dec 21, 2002.

1. The device

We have a simple device which can be plugged into a parallel port of any computer possessing it. The device has an LED and a 910 ohm resistor connecting pin 2 and 25 of the parallel port in series. The end-effect is that if you output a 1-bit into bit-0 of the base-port of the parallel port, the LED will be lit, and if you output a 0-bit into it, the LED will be turned off. Your task is to write a device driver for the port. The device driver should implement two devices.

The first device, with minor number 0, is output-only. When a character is written into the device, the LED should show the international morse-code representing the character. For example, if you write a single character "m" into it using the *write* system call, the device driver should turn on the LED, wait 3 units of time, turn off the LED, wait 1 unit of time, turn on the LED, wait 3 units of time, turn off the LED, wait 3 units of time, and return. The possible codes and time durations are specified in the next section.

The second device, with minor number 1, controls the rate at which international morse-code is generated on the first device. To the user, the device looks like a text file which stores the amount of time for the LED to lit for a dot, in 10ms units (so a number 20 means the device should lit for 0.2 second for a dot). If the user reads that file, the ASCII representation of that number (together with an end-of-line character) is returned; if the user write an ASCII number to that file (whether or not with an end-of-line character), the speed of the device changes.

Here are some more technical requirements of the device:

- The device driver should be implemented in a kernel module. The kernel module should have a parameter *ioport* specifying the I/O port number of the printer port. It should check that the number is one of 0x378, 0x278 or 0x3bc, with 0x378 being the default. An invalid I/O port number should result in the error *EINVAL*.
- The I/O port should be allocated when the module is loaded, and deallocated when the module is unloaded. Failure to allocate the port should result in the error *EBUSY*.
- When the device driver is loaded, the device driver should test whether it is possible to write a bit pattern of all 1's into it, read it, output a bit pattern of all 0's into it, and read it again. If what is read is different from what is written into it, the loading of the module should fail with the error code *ENODEV*.
- By default, the major number of the device should be dynamically allocated. However, there should be a module parameter *devmajor* allowing the user to directly specify the major number for it. Failure to allocate the major number should result in the error code *EBUSY*.
- At any time, only one of the two devices can be opened. Any attempt to open the second device when the first is already opened should result in the error *EBUSY*.
- The open mode should be checked when the device is opened: the first (code generation) device should allow opening for writing only. The second (control) device should allow opening either for reading or for writing, but not for both. Failure should result in the error *EINVAL*.

- At any time, only one operation of a device can be active. E.g., if a thread of a process is writing to a device, the second thread cannot write to the device again. Any such attempt should fail with *EBUSY*.
- Since many user programs fail to call the *write* system call again in case the write is completed only partially, the write method of the morse-code generation device should output the complete buffer to the device before returning (unless a signal interrupts it, see below).
- It should be possible to interrupt a pending write to the morse-code generation device by a signal. An interrupt should be checked (using *signal_pending()*, see ULK Chapter 9 p. 256) when a complete character is generated. In the middle of character generation, the process should be in an “uninterruptable” state.
- The speed of the morse code generation is controlled by the second device. When opened for output, it responds to all write requests up to the first non-digit character. The characters received before that are converted to a number and is used to change the morse code generation speed. All data after the first non-digit character should be silently ignored.
- Initially, the length of a dot should be 200 milliseconds. This duration should be limited to between 50 milliseconds and 1000 milliseconds. An attempt to set the dot-length outside this range should instead set it to the limit. (E.g., an attempt to set it to 10 milliseconds should actually set it to 50 milliseconds.)

2. Morse-code

Your device should at least be able to handle the following characters:

A	.-	G	--.	M	--	S	...	Y	-.--	4-
B	-...	H	N	-.	T	-	Z	--..	5
C	-.-.	I	..	O	---	U	..-	0	-----	6	-.....
D	-..	J	.-.-	P	.-.-.	V	...-	1	.-----	7	-----
E	.	K	-.-	Q	--.-	W	.-.-	2	..-----	8	-----
F	...-	L	.-..	R	.-.	X	-...-	3	...--	9	-----

These are implemented by the signal of the channel (for our case, the LED) being ON and OFF. A period (“dot”) in the above table is a short ON pulse, which length is usually the base-unit of time duration of the code generated. A hyphen (“dash”) in the above table is a long ON pulse, which is of length being the same as 3 dots. After a dot or a dash, there is always an OFF period as long as a dot. After the dots and dashes of a character is completely generated, there should be an extra OFF period as long as two dots (the “inter-character space”). Finally, a space character should be generated as an OFF period as long as 4 dots (the “inter-word space”).

You may consult a more complete morse code table to implement more characters (in particular, punctuation characters), but that is not necessary for our assignment. Case of characters should be ignored (alternatively, you may generate an upper-case character without the extra OFF period after the character, which is how morse code is actually used: the SOS “save-our-ship” signal has nine pulses without any inter-character space). Unsupported characters should be silently ignored by the device driver.

3. The device

Each group is allocated one copy of the simple device used for this assignment. Please collect it from the office of our TA (llcheng@csis.hku.hk). The connections of the device is not

particularly strong, so handle it with care, and never try to remove it from the computer by pulling the LED, resistor or the metallic wire (“legs”). Instead, gently apply force on the base of the parallel port to install and remove the device.

Please also note that the parallel port is directly connected to the circuitry of the motherboard, and it can be dangerous to the motherboard if you apply voltage into the connections (or short random connectors) when the device is in use. So avoid any contact of metal to the device when it is connected to the parallel port.

4. Testing out your device

You should write programs to test your device for various conditions. However, as a preliminary test, you may use `echo` and `cat` to test your device. E.g., to output `Hello, world` to your device, you might use this:

```
/bin/echo Hello, world > morseport
```

where `morseport` is a device you make for morse-code generation. Remember to type the `/bin/`: the `bash` shell program provides an internal command called `echo` which will be invoked if you don't type the `/bin/` prefix, but the shell ignores signals so you will be unable to test when a signal can be sent to interrupt writing to the device.

5. Handin

Send the module source file. You should write user programs to test your module, but you do not need to handin those user programs.

This is a group assignment, allowing at most 2 students in a group. Please put the names, university numbers and login-ids of all group members as comment of the code you hand-in.

6. References

Tutorial 10 notes and solution. (Device driver.)

Linux Device Drivers, 2nd Edition. (<http://www.xml.com/ldd/chapter/book/>):

- Chapter 2: I/O Ports and I/O Memory; Automatic and Manual Configuration.
- Chapter 3: File operations; A Brief Introduction to Race Conditions; read and write.
- Chapter 5: Access Control on a Device File.
- Chapter 6: Delaying Execution.
- Chapter 8: Using I/O Ports, using Digital I/O Ports.