# GPIO access from kernel space

Asked 7 years, 11 months ago Active 2 years, 11 months ago Viewed 13k times

3

1

After attempting to write a simple char device driver I now wish to write a char device driver to access the GPIO pins on a embedded linux board such as a Beagleboard. I am interested in writing a module `"mygpiomodule"` which when loaded must appear in `/dev/mygpiomodule` such that read, write from user space accesses the GPIO. I do not wish to access GPIO from user space. I want to do it as my first attempt towards writing a module that can interface with some peripheral. I want to stick some LEDs into the port and attempt to turn them on or off.

How should I attempt doing this?

Thanks
Mir

linux-kernel linux-device-driver

edited Jan 14 '12 at 0:10

asked Jan 12 '12 at 8:54

Mir
602 4 8 20

## 3 Answers

0

If you only want to toggle a few GPIO bits from a kernel driver, then you probably don't want to use the gpio_xxx APIs that user 'omotto' mentioned, because that is intended mostly to allow other driver to access GPIOs by name, as well as provide access to userspace. The quick-and-dirty solution from davroslyrad would be more appropriate for you.

answered Jan 17 '17 at 21:50

user5613277

4

Linux kernel allows you to "play" with GPIOs easily.

There is an include file for working with GPIOs:

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

GPIOs must be allocated before use, though the current implementation does not enforce this requirement. The basic allocation function is:

The gpio parameter indicates which GPIO is required, while label associates a string with it that can later appear in sysfs. The usual convention applies: a zero return code indicates success; otherwise the return value will be a negative error number. A GPIO can be returned to the system with:

```
void gpio_free(unsigned int gpio);
```

Some GPIOs are used for output, others for input. A suitably-wired GPIO can be used in either mode, though only one direction is active at any given time. Kernel code must inform the GPIO core of how a line is to be used; that is done with these functions:

```
int gpio_direction_input(unsigned int gpio);
int gpio_direction_output(unsigned int gpio, int value);
```

In either case, gpio is the GPIO number. In the output case, the value of the GPIO (zero or one) must also be specified; the GPIO will be set accordingly as part of the call. For both functions, the return value is again zero or a negative error number. The direction of (suitably capable) GPIOs can be changed at any time.

For input GPIOs, the current value can be read with:

```
int gpio_get_value(unsigned int gpio);
```

This function returns the value of the provided gpio; it has no provision for returning an error code. It is assumed (correctly in almost all cases) that any errors will be found when gpio_direction_input() is called, so checking the return value from that function is important.

Setting the value of output GPIOs can always be done using gpio_direction_output(), but, if the GPIO is known to be in output mode already, gpio_set_value() may be a bit more efficient:

```
void gpio_set_value(unsigned int gpio, int value);
```

For more information check this link: enter link description here

answered Jan 17 '17 at 15:49

omotto

**827** 11 15

---

6

Fortunately writing Linux drivers is not overly complicated, although somewhat more than can be taught within this forum. However the good news is that there are many sources available on the web that does a very good job on explaining exactly what you are wanting to do. Here are just few that I have used (even to make GPIO/LED control drivers), and they do provide source code that works and can be used as a base for your driver.

Free Software Magazine article

Linux Documentation Project article

Linux Journal article

The very "TinkerToy" nature of Linux provides a lot of flexibility. For instance, you might consider having your driver attach to the "/proc" file system as well (using create_proc_entry() API), this way you can access your driver without needing to write a dedicated application by simply "echo YOURSTRING > /proc/mygpiomodule" strings to your driver. To read from the driver would use "cat /proc/mygpiomodule". The links above also include examples on doing this as well. This can help with testing, and accessing your driver via startup scripts if you need to do that.

Take some time to review the articles/books I listed above, and certainly Google for more if you need to. Go ahead dive right in, and compile up and run some of the examples, and you will come up to speed rather quickly.

answered Jan 23 '12 at 15:45



davroslyrad
**399** 2 8