



New GPIO interface for linux user space

FOSDEM '18

Bartosz Golaszewski

About us

- Baylibre
 - Embedded Linux Engineering Firm
 - ~20 senior engineers, coming from the semiconductor world
 - HW and SW products: from concept to manufacturing
 - Upstream Linux kernel development and maintenance
 - Founding developers of kernelCI.org project



GPIO – overview

- General-purpose input/output
- Generic pin
- Can be configured at run time
 - Input (readable)/output (writable)
 - Enabled/disabled
 - IRQs
- Simple applications:
 - Buttons, LEDs, buzzers, power-switches/relays, stepper motors, level detectors, thermostats, etc...
- Provided by SoCs, expanders or multifunction devices (I2C, SPI, ...)



GPIO in the kernel

- Provider-consumer model
- Two co-existing interfaces
 - Based on GPIO numbers (legacy, deprecated)
 - Based on GPIO descriptors (recommended)
 - Easy access to GPIOs associated with devices
 - More fine-grained control
- Support for devres
- GPIO chip drivers in drivers/gpio
- Consumers all over the place
 - Writing drivers for devices using GPIOs is encouraged wherever possible



GPIO in user space

- Needed when no kernel device drivers provided/possible
 - Power switches
 - Relays
 - GPS
 - Bluetooth
- Certain users prefer to toggle GPIOs from user space
 - Intelligent home systems
 - Robotics



/sys/class/gpio – legacy user API

- d8f388d8 (“gpio: sysfs interface”)
- State not tied to process
 - Concurrent access to sysfs attributes
 - If process crashes, the GPIOs remain exported
- Cumbersome API
 - Multiple attributes per GPIO: value, direction, active_low, edge
 - Single sequence of GPIO numbers representing a two-level hierarchy - necessary to calculate the number of the GPIO, numbers not stable
 - Polling possible but complicated: need to lseek() or reopen ‘value’ on events, need to open ‘value’ separately for every GPIO, events not queued



Character device – new user API

- Merged in linux v4.8
- One device file per gpiochip
 - /dev/gpiochip0, /dev/gpiochip1, /dev/gpiochipX...
- Similar to other kernel interfaces: open() + ioctl() + poll() + read() + close()
- Possible to request multiple lines at once (for reading/setting values)
- Possible to find GPIO lines and chips by name
- Open-source and open-drain flags
- User/consumer strings
- Uevents
- Reliable polling



Character device – user API (linux/gpio.h)

- Chip info
- Line info
- Line request for values
- Reading values
- Setting values
- Line request for events
- Polling for events
- Reading events



Character device – chip info

```
struct gpiochip_info {  
    char name[32];  
    char label[32];  
    __u32 lines;  
};  
  
void get_chip_info(void)  
{  
    struct gpiochip_info info;  
    int fd, rv;  
  
    fd = open("/dev/gpiochip0", O_RDWR);  
    rv = ioctl(fd, GPIO_GET_CHIPINFO_IOCTL, info);  
}
```



Character device – line info

```
struct gpioline_info {
    __u32 line_offset;
    __u32 flags;
    char name[32];
    char consumer[32];
};

#define GPIOLINE_FLAG_KERNEL      (1UL << 0)
#define GPIOLINE_FLAG_IS_OUT     (1UL << 1)
#define GPIOLINE_FLAG_ACTIVE_LOW (1UL << 2)
#define GPIOLINE_FLAG_OPEN_DRAIN (1UL << 3)
#define GPIOLINE_FLAG_OPEN_SOURCE (1UL << 4)

void get_line_info(void)
{
    struct gpioline_info info;

    memset(&info, 0, sizeof(info));
    info.line_offset = 3;

    rv = ioctl(fd, GPIO_GET_LINEINFO_IOCTL, &info);
}
```



Character device – requesting lines

```
#define GPIOHANDLES_MAX          64
#define GPIOHANDLE_REQUEST_INPUT (1UL << 0)
#define GPIOHANDLE_REQUEST_OUTPUT (1UL << 1)
#define GPIOHANDLE_REQUEST_ACTIVE_LOW (1UL << 2)
#define GPIOHANDLE_REQUEST_OPEN_DRAIN (1UL << 3)
#define GPIOHANDLE_REQUEST_OPEN_SOURCE (1UL << 4)

struct gpiohandle_request {
    __u32 lineoffsets[GPIOHANDLES_MAX];
    __u32 flags;
    __u8 default_values[GPIOHANDLES_MAX];
    char consumer_label[32];
    __u32 lines;
    int fd;
};

void request_output(void)
{
    struct gpiohandle_request req;
    int rv;

    req.flags |= GPIOHANDLE_REQUEST_OUTPUT;
    req.lines = 2;
    req.lineoffsets[0] = 3;
    req.lineoffsets[1] = 5;
    req.default_values[0] = 1;
    req.default_values[1] = 0;
    strcpy(req.consumer_label, "foobar");

    rv = ioctl(fd, GPIO_GET_LINEHANDLE_IOCTL, &req);
}
```



Character device – reading/setting values

```
#define GPIOHANDLE_GET_LINE_VALUES_IOCTL _IOWR(0xB4, 0x08, struct gpiohandle_data)
#define GPIOHANDLE_SET_LINE_VALUES_IOCTL _IOWR(0xB4, 0x09, struct gpiohandle_data)

struct gpiohandle_data {
    __u8 values[GPIOHANDLES_MAX];
};

void set_values(void)
{
    struct gpiohandle_data data;
    int rv;

    data.values[0] = 0;
    data.values[1] = 1;

    rv = ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
}

void get_values(void)
{
    struct gpiohandle_data data;
    int rv;

    memset(&data, 0, sizeof(data));

    rv = ioctl(req.fd, GPIOHANDLE_GET_LINE_VALUES_IOCTL, &data);
}
```



Character device – event requests

```
#define GPIOEVENT_REQUEST_RISING_EDGE    (1UL << 0)
#define GPIOEVENT_REQUEST_FALLING_EDGE  (1UL << 1)
#define GPIOEVENT_REQUEST_BOTH_EDGES    ((1UL << 0) | (1UL << 1))

struct gpioevent_request {
    __u32 lineoffset;
    __u32 handleflags;
    __u32 eventflags;
    char consumer_label[32];
    int fd;
};

void request_event(void)
{
    struct gpioevent_request req;
    int rv;

    req.lineoffset = 4;
    req.handleflags = GPIOHANDLE_REQUEST_INPUT;
    req.eventflags = GPIOEVENT_REQUEST_BOTH_EDGES;
    strcpy(req.consumer_label, "foobar");

    rv = ioctl(fd, GPIO_GET_LINEEVENT_IOCTL, &req);
}
```



Character device – polling & reading events

```
#define GPIOEVENT_EVENT_RISING_EDGE 0x01
#define GPIOEVENT_EVENT_FALLING_EDGE 0x02

struct gpioevent_data {
    __u64 timestamp;
    __u32 id;
};

void recv_event(void)
{
    struct gpioevent_data event;
    struct pollfd pfd;
    ssize_t rd;
    int rv;

    pfd.fd = req.fd;
    pfd.events = POLLIN | POLLPRI;

    rv = poll(&pfd, 1, 1000);
    if (rv > 0)
        rd = read(req.fd, &event, sizeof(event));
}
```



libgpiod – C library & tools for GPIO chardev

- History
 - Needed a solution for toggling power switches on BayLibre ACME
 - ~~IIO attributes~~
 - ~~Regulators controlled from user space~~
 - GPIO character device
 - Version 0.1 released on January 18th
 - Current stable version is 0.3.2
 - 1.0 release is work-in-progress, API needs review



libgpiod – C library & tools for GPIO chardev

- Features
 - C API, fully documented in doxygen
 - Command-line tools: gpiodetect, gpioinfo, gpioset, gpioget, gpiofind & gpiomon
 - Custom test suite (working together with gpiod-mockup kernel module and irq_sim)



libgpiod – C library & tools for GPIO chardev

- C API split into logical parts:
 - Simple API
 - Chip operations
 - Line operations
 - Info, requests, events
 - Iterators



libgpiod – C API examples

```
struct gpiod_chip *chip;
struct gpiod_line *line;
int rv, value;

chip = gpiod_chip_open("/dev/gpiochip0");
if (!chip)
    return -1;

line = gpiod_chip_get_line(chip, 3);
if (!line) {
    gpiod_chip_close(chip);
    return -1;
}

rv = gpiod_line_input(line, "foobar");
if (rv) {
    gpiod_chip_close(chip);
    return -1;
}

value = gpiod_line_get_value(line);

gpiod_chip_close(chip)
```

```
struct timespec ts = { 0, 1000000 };
struct gpiod_line_event event;
struct gpiod_chip *chip;
struct gpiod_line *line;
int rv, value;

chip = gpiod_chip_open("/dev/gpiochip0");
if (!chip)
    return -1;

line = gpiod_chip_get_line(chip, 3);
if (!line) {
    gpiod_chip_close(chip);
    return -1;
}

rv = gpiod_line_request_rising_edge_events(line, "foobar");
if (rv) {
    gpiod_chip_close(chip);
    return -1;
}

do {
    rv = gpiod_line_event_wait(line, &ts);
} while (rv <= 0);

rv = gpiod_line_event_read(line, &event);

gpiod_chip_close(chip)
```



libgpiod tools - examples

```
$ gpiodetect  
gpiochip2 [gpio-mockup-C] (8 lines)  
gpiochip1 [gpio-mockup-B] (8 lines)  
gpiochip0 [gpio-mockup-A] (8 lines)
```

```
$ gpioinfo gpiochip1  
gpiochip1 - 8 lines:  
  line 0: "gpio-mockup-B-0" unused output active-high  
  line 1: "gpio-mockup-B-1" unused output active-high  
  line 2: "gpio-mockup-B-2" unused output active-high  
  line 3: "gpio-mockup-B-3" unused output active-high  
  line 4: "gpio-mockup-B-4" unused output active-high  
  line 5: "gpio-mockup-B-5" unused output active-high  
  line 6: "gpio-mockup-B-6" unused output active-high  
  line 7: "gpio-mockup-B-7" unused output active-high
```



libgpiod tools - examples

```
$ gpiofind gpio-mockup-B-3  
gpiochip1 3
```

```
$ gpioget `gpiofind gpio-mockup-B-3`  
0
```

```
$ gpioset gpiochip1 3=1  
$ gpioget gpiochip1 1 2 3 4 5  
0 0 1 0 0
```

```
$ gpioset --mode=wait gpiochip2 0=1
```

```
$ gpiomon gpiochip0 2  
event: RISING EDGE offset: 2 timestamp:  
[1508094667.935877214]
```

```
$ gpiomon --format="%o %e %s.%n" gpiochip0 2  
2 1 1508094729.895930484
```



libgpod – future work

- Planned features – started or planned
 - GPIO daemon + client
 - C++ bindings (in progress)
 - Python bindings (ctypes? C extensions?)
- Potential
 - Bindings for other languages



libgpiod – C library & tools for GPIO chardev

- Where to get it:
 - Hosted at kernel.org
 - Source:
<https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/>
 - Releases: <https://www.kernel.org/pub/software/libs/libgpiod/>
 - Available in meta-openembedded & buildroot
 - Packaged in Fedora, Arch, Debian linux
- Contributions & bug reports:
 - Send e-mails to linux-gpio@vger.kernel.org
 - Use [libgpiod] prefix



Q & A

THANK YOU!

