

 **Update 3 files**
Khaled Rami Omar authored 1 minute ago

1c630402

README.md 8.95 KiB

Group 3

Members

- John Nguyen
- Ibrahim Merhi
- Khaled Rami Omar

Link to repository

<https://gitlab.au.dk/au-ece-sw3hal/e2024/Group-3.git>

Øvelse: Linux Device Driver med GPIO (Rpi)

Formål

Denne øvelse giver erfaring med at skrive, compilere og indsætte en basal Linux Device Driver.

Du forventes med øvelsen at opfylde følgende mål:

Mål	Delmål 1	Delmål 2	Delmål 3
Initialisere af driver	init gpio interfaces	Init major/minor numre og cdev	Lave fejlhåndtering
Implementere filoperationer	Impl. og test open/release	Impl. read	Impl.write
Anvende driver	Indsætte driver og oprette major/minor numre	Anvende file operations	Anvende kernel log til debugging/test

Øvelsen

Lav 2 drivere, der hhv. håndterer læsning og skrivning.

- Driver 1: Læsning fra devicet (/dev/sw2) returnerer hvorvidt SW2 er trykket ned eller ej.
- Driver 2: Skrivning til devicet (/dev/led3) tænder og slukker LED3.

NB! 2 Drivere medfører 2x makefile + 2x driver .c fil, navnene er desuden givet ovenfor. Placér hver driver i sin egen folder.

Vi anvender SW2 og LED3. Du kan finde deres GPIO numre i diagrammet, ligesom i tidligere øvelse.

GPIO LDD driverne i steps

Nedenfor er vist en overfladisk skabelon som kan bruges som inspiration når I skal lave de 2 drivere. Bemærk at driver 1 ikke har en `write()` funktion, da den kun bør kunne læse fra SW2. Den gpio som svarer til SW2 skal altså være sat til input, som er default. Derimod skal driver 2 have både en `write()` og en `read()` funktion. `write()` funktionen er naturligvis for at kunne tænde/slukke for LED3, mens `read()` er til for at kunne få at vide om den er tændt eller slukket. I det sidste tilfælde (LED3 driveren) skal gpio'en være et output.

HUSK! At udfylde `module_init`, `module_exit`, samt `licens`, `author` og `description`.

a) Implementer "`init`" og "`exit`" funktionerne. Det er her I skal requeste og free gpio's, samt sætte gpio_direction til at styre pin'ens retning. Samtidigt er det afgørende at du/I laver en ordentlig fejlhåndtering i `init()`.

```
static int mygpio_init(void) {
    int err = 0;

    // Request GPIO for SW2
    err = gpio_request(SW2, "SW2");
```

```

if (err) {
    pr_err("Failed to request GPIO for SW2\n");
    return err;
}

// Set GPIO direction (input)
err = gpio_direction_input(SW2);
if (err) {
    pr_err("Failed to set GPIO direction for SW2\n");
    goto err_free_gpio;
}

// Allocate Major/Minor numbers
err = alloc_chrdev_region(&devno, first_minor, max_devices, "switch-driver");
if (err < 0) {
    pr_err("Failed to allocate char dev region\n");
    goto err_free_gpio;
}

pr_info("Switch driver got Major %i\n", MAJOR(devno));

// Create class
switch_class = class_create(THIS_MODULE, "switch_class");
if (IS_ERR(switch_class)) {
    pr_err("Failed to create class for SW2\n");
    err = PTR_ERR(switch_class);
    goto err_unregister_chrdev;
}

// Initialize and add cdev
cdev_init(&switch_cdev, &switch_fops);
err = cdev_add(&switch_cdev, devno, 1); // Register one device only
if (err < 0) {
    pr_err("Failed to add cdev for SW2\n");
    goto err_destroy_class;
}

return 0;

err_destroy_class:
    class_destroy(switch_class);
err_unregister_chrdev:
    unregister_chrdev_region(devno, max_devices);
err_free_gpio:
    gpio_free(SW2);

    return err;
}

```

Funktionen mygpio_init initialiserer en driver til GPIO-enheden SW2. Den anmoder først om adgang til GPIO-pinen ved hjælp af gpio_request; hvis dette mislykkes, logges en fejl og funktionen returnerer. Derefter indstilles GPIO-pinen til input-tilstand med gpio_direction_input. Hvis dette også fejler, frigives GPIO-ressourcen. Funktionen allokere herefter major og minor numre via alloc_chrdev_region, som logges, hvis det lykkes. En klasse oprettes med class_create, og ved fejl frigives tidligere ressourcer. Endelig initialiseres og tilføjes den karakterbaserede enhed (cdev) med cdev_add. Fejl under nogen af disse processer håndteres for at sikre korrekt frigivelse af ressourcer. Ved succesfuld initialisering returneres 0.

```

static void mygpio_exit(void) {
    cdev_del(&switch_cdev);
    unregister_chrdev_region(devno, max_devices);
    class_destroy(switch_class);
    gpio_free(SW2);
}

```

Funktionen mygpio_exit rydder op ved afinstallation af GPIO-driveren. Den fjerner den karakterbaserede enhed med cdev_del, unregisterer major og minor numre, destruerer klassen med class_destroy, og frigiver GPIO-pinen SW2 med gpio_free. Dette sikrer, at alle ressourcer ryddes korrekt op og undgår hukommelseslækager.

b) Implementer `open` og `release`. Dvs den kode som eksekveres når en applikation forsøger at åbne/lukke et device (fil). Brug følgende funktioner som de er (De skriver en besked til kernel loggen om hhv open/release med hvilket major og minor nummer):

```
int mygpio_open(struct inode *inode, struct file *filep) {
    int major, minor;
    major = MAJOR(inode->i_rdev);
    minor = MINOR(inode->i_rdev);
    pr_info("Opening Switch Device [major], [minor]: %i, %i\n", major, minor);
    return 0;
}

int mygpio_release(struct inode *inode, struct file *filep) {
    int major, minor;
    major = MAJOR(inode->i_rdev);
    minor = MINOR(inode->i_rdev);
    pr_info("Closing/Releasing Switch Device [major], [minor]: %i, %i\n", major, minor);
    return 0;
}
```

Test ved at bygge-, indsætte driveren, oprette en node. Ex her med major = 239 og minor = 0:

```
mknod /dev/mygpio c 239 0
```

```
=(none) old-ses=4294967295 ses=2 res=1
[ 969.196771] audit: type=1006 audit(1581091617.070:4): pid=268 uid=0 old-auid=4294967295 auid=0 tty
=(none) old-ses=4294967295 ses=3 res=1
[ 1018.587538] sw2: no symbol version for module_layout
[ 1018.597023] sw2: loading out-of-tree module taints kernel.
[ 1018.614191] Switch driver got Major 239
root@raspberrypi0-wifi:~# mknod /dev/sw2 c 239 0
```

c) Implementer `read` (Driver 1 - læsning af SW1 + Driver 2 - Er LED3 tændt eller?). Dvs den kode som eksekveres i kernel-space når en user-space applikation forsøger at læse fra et device (fil).

```
ssize_t mygpio_read(struct file *filep, char __user *buf,
                    size_t count, loff_t *f_pos)
{
    int value;
    char kbuf[12];
    int len;

    // Read GPIO value (input)
    value = gpio_get_value(SW2);
    len = snprintf(kbuf, sizeof(kbuf), "%d\n", value);

    if (*f_pos > 0) {
        return 0; // EOF
    }

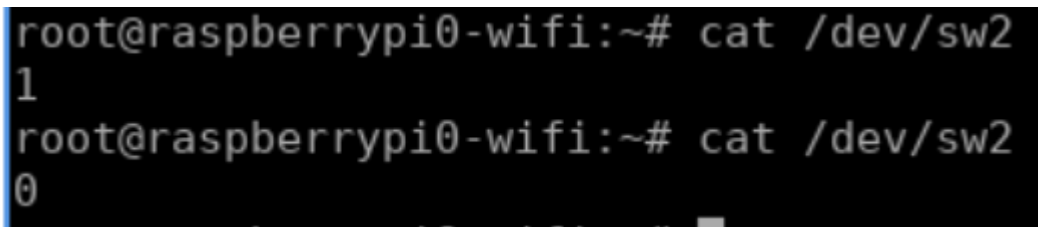
    if (copy_to_user(buf, kbuf, len)) {
        return -EFAULT;
    }

    *f_pos += len;
    return len;
}
```

Funktionen `mygpio_read` læser værdien fra GPIO-pinden (SW2) og gemmer den i `value`. Den formaterer derefter værdien som en streng i `kbuf`. Hvis læsepositionen (`*f_pos`) er større end 0, returneres 0 for at angive EOF. Hvis ikke, forsøger den at kopiere dataene til brugerens buffer (`buf`) med `copy_to_user`. Hvis kopieringen mislykkes, returneres `-EFAULT`. Hvis den er succesfuld, opdateres læsepositionen, og længden af de læste bytes returneres.

Test ved at bygge, scp og indsætte device driveren, samt oprette node. Prøv nu at læse fra noden, den læste værdi skal gerne afspejle tilstanden af trykknapperne. I kan bruge "cat" eller udvide jeres lille testapplikation. Du kan blive inspireret [her](#) og kompilere den med "arm-rpizw-gcc -o rd rd.c".

Bemærk at programmer som f.eks "cat" læser indtil de møder en End of File karakter (EOF). "cat" vil derfor læse uendeligt fra driveren og læsse bunkevis af data ud på terminalen.



```
root@raspberrypi0-wifi:~# cat /dev/sw2
1
root@raspberrypi0-wifi:~# cat /dev/sw2
0
```

d) Implementer `write` (Driver 2 - Skrivning til LED3 - altså om den er tændt eller ej.)

```
ssize_t mygpio_write(struct file *filep, const char __user *ubuf, size_t count, loff_t *f_pos) {
    int len, value;
    char kbuf[12];

    len = count < sizeof(kbuf) ? count : sizeof(kbuf) - 1;

    if (copy_from_user(kbuf, ubuf, len)) {
        return -EFAULT;
    }

    kbuf[len] = '\0'; // Null-terminate the string

    if (kstrtoint(kbuf, 10, &value)) {
        pr_err("Error converting string to int\n");
        return -EINVAL;
    }

    gpio_set_value(LED3, value); // Set GPIO value (output)

    pr_info("Wrote %i to LED3\n", value);
    *f_pos += len;
    return len;
}
```

Funktionen `mygpio_write` håndterer skrivning til en enhed ved at modtage data fra brugeren. Den kopierer data fra brugerens buffer (`ubuf`) til en intern buffer (`kbuf`), og sikrer, at den ikke overskrider størrelsen. Hvis kopieringen mislykkes, returneres en fejlkode. Herefter konverteres indholdet af `kbuf` til et heltal med `kstrtoint`. Ved fejl i konverteringen logges en fejlmeddelelse, og en fejlkode returneres. Hvis konverteringen lykkes, indstilles GPIO-pinden (LED3) til den ønskede værdi, og der logges en succesmeddelelse. Funktionen afsluttes ved at opdatere læsepositionen og returnere antallet af skrevne bytes.

Test ved at bygge, scp og indsætte device driveren, samt oprette node. Skriv til noden for at tænde og slukke led'en.

```
echo 1 > /dev/led3
```