ОСуРВ лабораторијска вежба 6

Улази-излаз у руковаоцу

верзија 1.1

Одсек РТ-РК

17. децембар 2025.

1 Увод

У овој вежби биће приказано како се на Линукс оперативном систему реализује руковалац (енгл. Driver), нашем случају за GPIO. У претходној вежби смо користили постојећи драјвер преко `/dev/gpio_stream` датотеке са проточним (енгл. Stream) командама. У проточним командама уписујемо операцију, пин и вредност да би руковали пиновима. Притом се не води рачуна о позицији у датотеци, већ се он посматра као рецимо конзола или датотека UART драјвера. У овој вежби ћемо преправити апликацију и драјвер да користе меморијски мапиране команде (енгл. Memory-mapped).

2 Радни процес

На почетку је погодно само да се уради загрејавање за драјвер на setup RPi-у:

```
cd ~/Public/OSuRV_Labs/L6_Kernel_Space/SW/Driver/gpio_ctrl/
git pull
make
make start
dmesg
```

Слично као и у претходној вежби, лабораторијска вежба се може клонирати са

```
git clone https://github.com/MilosSubotic/OSuRV_Labs
```

и позиционирамо се у њу:

```
cd OSuRV_Labs/L6_Kernel_Space/
```

Пошто ће сваки студент компајлирати драјвер, онда је процес за драјвер на локалном RPi:

```
cd SW/Driver/gpio_ctrl/
make
scp gpio_ctrl.ko pi@rpi-setup-1.local:/home/pi/Public/
```

На setup RPi преко ssh драјвер се покреће са:

```
cd /home/pi/Public
# Remove previous driver from kernel.
sudo rmmod gpio_ctrl
```

```
# Insert new driver to kernel.
sudo insmod gpio_ctrl.ko
```

На локалном RPi процес за апликацију је:

```
cd SW/Test/test_gpio/
./waf configure
./waf build
scp ./build/test_gpio pi@rpi-setup-1.local:/home/pi/Public
```

На setup RPi, преко ssh везе извршити:

```
cd /home/pi/Public
./test_gpio w 17 1
./test_gpio w 17 0
```

Ова команда ће да пали и гаси LED на драјверу корачног мотора.

3 Структура драјвера

Са асистентима треба да прођете кроз код драјвера у `SW/Driver/gpio_ctrl/main.c`. Треба кренути са дна датотеке.

```
module_init(gpio_ctrl_init);
module_exit(gpio_ctrl_exit);
```

су улазне тачке (енгл. Entry points) за овај модул драјвера, конструктор и деструктор.

Операције на `/dev/gpio_stream` датотеком драјвера се региструје путем:

```
r = register_chrdev(DEV_MAJOR, DEV_STREAM_NAME, &gpio_stream_fops);
```

где се `DEV_MAJOR` и `DEV_STREAM_NAME` (који има вредност `gpio_stream`) се повезује `/dev/gpio_stream` у љусци (енгл. Shell) и `gpio_stream_fops`.

`gpio_stream_fops` је структура са показивачима на функције. Ако би `struct file_operations` био интерфејс драјвера, `gpio_stream_fops` би била класа са имплементираним виртуелним методама. Први параметара свих ових метода је `struct file* filp`, инстанца класе датотеке драјвера. Позиви у корисничком простору функција за руковање датотекама се пресликавају на позиве у простору кернела на ове методе. Тако позив `write()` функције ће завршити на методи `gpio_stream_write()`.

Нама је овде најзначајнији упис, чија метода има потпис:

```
static ssize_t gpio_stream_write(
    struct file* filp,
    const char *buf,
    size_t len,
    loff_t *f_pos
)
```

`buf` је показивач на податке који програм у корисничком простору жели да упише. `len` је количина података која се уписује. `f_pos` је одстојање од почетка датотеке, које се додуше у проточном режиму не користи.

Након неких провера, први корак је копирање података из бафера у корисничком меморијском простуру у бафер у меморијском простору кернела:

```
if(copy_from_user(pkg, buf, len) != 0){
    return -EFAULT;
}
```

Након интерпретације пакета завршавамо са следећим кодом који позива функције библиотеке за GPIO у простору кернела:

```
gpio__steer_pinmux(gpio_no, GPIO__OUT);

if(wr_val){
    gpio__set(gpio_no);
}else{
    gpio__clear(gpio_no);
}
```

4 Задатак: апликација

Треба променити апликацију `SW/Test/test_gpio/test_gpio.c`. Апликација треба да користи `/dev/gpio_mmap` датотеку драјвера. Апликација и драјвер треба да подржавају меморијски мапиране команде. У овом режиму се у датотеку драјвера треба уписати вредност коју желимо на GPIO пин (0 или 1) на са одговарајућим одстојањем од почетка датотеке (енгл. Offset). Ширина команде је 1 бајт. Примера ради, ако упишемо 1 на 3 бајт онда ће на GPIO3 пину бити 1, а ако упишемо 0 на 4 бајт онда ће на GPIO4 пину бити 0.

Први корак је заменити `/dev/gpio_stream` са `/dev/gpio_mmap`.

Следећи је корак измена операције уписа. Тренутни пакет за проточне команде је:

```
if(op == 'w'){
    uint8_t pkg[3];
    pkg[0] = 'w';
    pkg[1] = gpio_no;
    pkg[2] = wr_val;
```

Јасно је да пакет сада треба да буде 1 бајт у који уписујемо `wr_val`.

Крајњи корак је да пре позива `write()` функције треба померити се на одговарајуће одстојање у датотеци са `lseek()` функцијом:

```
    lseek(fd, gpio_no, SEEK_SET);
```

Додатни задатак ако остане времена: проверити повратну вредност `lseek()`.

Додатни задатак ако остане времена: Подржати оба командна приступа са селекцијом при компајлирању коришћењем C макроа:

- `#define MMAP_API 1`
- `#if MMAP_API`
- `#else`
- `#endif`

5 Задатак: драјвер

Треба променити у драјверу `SW/Driver/gpio_ctrl/` датотеке `main.c` и `Makefile`. Први корак је рефакторисање кода да се промени `gpio_stream` у `gpio_mmap`, што је релативно лако са Find and Replace функционалношћу у графичким едиторима.

Из корисничког простора смо користили `lseek()` и `write()`. У драјверу `gpio_mmap_llseek()` је већ реализован и главна ствар је да имплементира

```
    case SEEK_SET:
        filp->f_pos = offset;
        break;
```

Можете додати неки испис чисто да се испрати рад позиционирања.

```
    printk(KERN_INFO DRV_NAME": offset = %d\n", offset);
    printk(KERN_INFO DRV_NAME": filp->f_pos = %d\n", filp->f_pos);
    break;
```

Следећи корак је модификација `gpio_mmap_write()`. Команда је овог пута 1 бајт, али се мора узети у обзир одстојање `f_pos`. Оно нам говори у који пин треба да упишемо `gpio_no`. Водити рачуна да је то показивач, јер га `gpio_mmap_write()` увећава за број уписаних бајтова `len`. Користи се исти код GPIO библиотеке из Листингу 3. Наравно треба прилагодити провере а обрисати остале операције.

Додатни задатак ако остане времена: у апликацији за мотор брисача уписати са једним `lseek()` и једним `write()` GPIO пинове 2, 3, 4.

Додатни задатак ако остане времена: читање с пина. Ултимативни тест је `SW/App/3_Wiper_Limit_SW`. Треба га покренути са `/dev/gpio_stream` пре него што се пресели на `/dev/gpio_mmap`.

Додатни задатак ако остане времена: Подржати оба командна приступа са селекцијом при компајлирању коришћењем C макроа.