

ОСуРВ лабораторијска вежба 6

Улази-излаз у руковаоцу

верзија 1.0

Одсек РТ-РК

15. децембар 2025.

1 Увод

У овој вежби биће приказано како се на Линукс оперативном систему из реализује руковалац (енгл. Driver), нашем случају за GPIO. У претходној вежби смо користили постојећи драјвер преко `/dev/gpio_stream` фајла са проточним (енгл. Stream) командама. У проточним командама уписујемо операцију, пин, вредност да би руковали пиновима. Притом се не води рачуна о позицији у фајлу, већ се он посматра као рецимо конзола или фајл 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 загревавање за драјвер на setup RPi-у:

```
1 cd ~/Public/OSuRV_Labs/L6_Kernel_Space/SW/Driver/gpio_ctrl/  
2 git pull  
3 make  
4 make start  
5 dmesg
```

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

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

и позиционирамо се у њу:

```
1 cd OSuRV_Labs/L6_Kernel_Space/
```

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

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

На setup RPi преко ssh драјвер се покреће са:

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

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

На локалном RPi процес за апликацију је:

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

На setup RPi преко ssh везе извршити:

```
1 cd /home/pi/Public
2 ./test_gpio w 17 1
3 ./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 фајла.

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

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

Операције на /dev/gpio_stream фајлом драјвера се региструје путем:

```
1 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 параметар свих ових метода је struct file* filp, инстанца класе фајла драјвера. Позиви у корисничком простору функција за руковање фајловима се пресликавају на позиве у кернелском простору на ове методе. Тако позив write() функције ће завршити на методи gpio_stream_write().

Нама је овде најзначајнији упис, чија метода има потпис:

```
1 static ssize_t gpio_stream_write(
2     struct file* filp,
3     const char *buf,
4     size_t len,
5     loff_t *f_pos
6 )
```

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

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

```
1 if(copy_from_user(pkg, buf, len) != 0){
2     return -EFAULT;
3 }
```

Након интерпретације пакета завршавамо са следећим кодом који позива функције библиотеке за GPIO у кернелском протору:

```
1 \label{lst:drv_write}
2 gpio__steer_pinmux(gpio_no, GPIO__OUT);
3
4 if(wr_val){
5     gpio__set(gpio_no);
6 }else{
7     gpio__clear(gpio_no);
8 }
```

4 Задатак: апликација

Треба променити апликацију SW/Test/test_gpio/test_gpio.c. Апликација треба да користи /dev/gpio_mmap фајл драјвера. Апликација и драјвер треба да подржавају меморијски мапирани команди. У овом режиму се у фајл драјвер треба уписати вредност коју желимо на GPIO пин (0 или 1) на са одговарајућим одстојањем од почетка фајла (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:

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

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

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

```
1 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 драјвер у SW/Driver/gpio_ctrl/main.c. Први корак је копирање gpio_stream и рефакторисање кода gpio_mmap. Значи копирати методе, структуру и регистрацију и променити stream у mmap. На овај начин ће коегзистирати ова радна режима, што је сигурније за тестирање.

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

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

Можете додати неки испис чисто да се испрати рад позиционирања.

```
1  printk(KERN_INFO DRV_NAME": offset = %d\n", offset);
2  printk(KERN_INFO DRV_NAME": filp->f_pos = %d\n", filp->f_pos);
3  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 ???. 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 пине 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`.