Drajveri za sekvencijalne uređaje u Linux OS-u

dr Predrag Teodorovic

Fakultet Tehničkih Nauka, Novi Sad

November 17, 2016

Uvod

- Cilj ovog poglavlja je da se opiše drajver za sekvencijalne uređaje (char device driver)
- Detaljno ćemo dokumentovati razvoj sekvencijalnog drajvera, jer je ova klasa drajvera pogodna za većinu jednostavnih hardverskih uređaja, a pored toga su jednostavniji od drajvera za blok uređaje i drajvera za mrežni interfejs

Upravljački brojevi uređaja

- Sekvencijalnim uređajima se pristupa preko imena u fajl sistemu
- Ta imena su nazvana specijalne datoteke ili datoteke uređaja ili jednostavno čvorovi stabla filesystem-a
- Oni su konvencionalno smešteni u /dev direktorijumu
- Posebne datoteke za sekvencijalne drajvere se prepoznaju po slovu "c" u prvoj koloni na izlazu koji se dobije komandom ls -l
- Ako unesemo ls -l komandu, videćemo dva broja (razdvojena zarezom)
- Ovi brojevi su glavni (*major*) i sporedni (*minor*) broj uređaja za određeni uređaj i veoma su značajni jer se preko njih vrši „povezivanje“ uređaja sa odgovarajućim drajverom
- Nazivamo ih ***upravljački brojevi uređaja***

Upravljački brojevi uređaja

```
crw-rw-rw-    1 root root 1, 3     Apr 11 2002   null
crw-------    1 root root 10, 1    Apr 11 2002   psaux
crw-------    1 root root 4, 1     Oct 28 03:04  tty1
crw-rw-rw-    1 root tty 4, 64     Apr 11 2002   ttys0
crw-rw----    1 root uucp 4, 65    Apr 11 2002   ttyS1
crw--w----    1 vcsa tty 7, 1      Apr 11 2002   vcs1
crw--w----    1 vcsa tty 7, 129    Apr 11 2002   vcsa1
crw-rw-rw-    1 root root 1, 5     Apr 11 2002   zero
```

- Glavni upravljački brojevi u primeru iznad su 1, 4, 7 i 10, dok su sporedni 1, 3, 5, 64, 65 i 129

Šta predstavljaju upravljački brojevi uređaja?

- Standardno, glavni broj identifikuje povezanost drivera sa uređajem
- Na primer /dev/null i /dev/zero su upravljani drajver-om sa glavnim brojem 1, dok virtuelnim konzolama i serijskim terminalima upravlja driver 4
- Linux kernel omogućava drajverima da dele glavne upravljačke brojeve, ali većina uređaja je dalje organizovana po principu gde jedna kombinacija glavni- sporedni broj odgovara jednom uređaju
- Sporedni broj koristi kernel da utvrdi tačno kojim uređajem se upravlja: zavisno od toga kako je napisan driver, možete dobiti direktno pokazivač na uređaj od kernela, ili možete koristiti sporedni broj kao indeks u nizu lokalnih uređaja. U svakom slučaju, sam kernel ne zna skoro ništa o sporednim brojevima uprkos činjenici da se oni odnose na uređaje implementirane vašim drajverom

Interna reprezentacija upravljačkih brojeva

- Unutar kernela, *dev_t* tip (definisan u <linux/types.h>) se koristi za predstavljanje brojeva uređaja, kako glavnih, tako i sporednih
- Tip *dev_t* je 32- bitni: 12 bita se izdvaja za glavne brojeve i 20 za sporedne
- Kod drajvera, ipak, ne treba praviti pretpostavke o načinu zapisivanja glavnih I sporednih brojeva u tipu *dev_t*, jer je činjenica da se sama veličina tipa *dev_t* kao i broj bita rezervisan za glavne i sporedne brojeve menjala od kernela do kernela

Kako onda pristupamo upravljačkim brojevima?

- Da bi se dobili glavni i sporedni broj iz *dev_t*, koriste se makroi definisani u linux/kdev_t.h

```
MAJOR(dev_t dev);
MINOR(dev_t dev);
```

- Ako, umesto toga, imate na raspolaganju glavni i sporedni broj nekog uređaja koje treba pretvoriti u *dev_t*, koristi se:

```
MKDEV(int major, int minor);
```

Dodeljivanje upravljačkih brojeva od strane OS-a

- Jedna od prvih stvari koju će driver morati da uradi prilikom postavljanja *char* uređaja jeste da dobije jedan ili više upravljačkih brojeva za rad
- Neophodna funkcija za ovaj zadatak je ***register_chrdev_region***, koja je definisana u <linux/fs.h>:

```
int register_chrdev_region(dev_t first,
    unsigned int count, char *name);
```

- Prvi argument definiše, indirektno, glavni i sporedni broj uređaja za koji se razvija drajver
- Argument *count* predstavlja ukupan broj uređaja koji će biti pokriveni drajverom i na ovaj način možemo automatski da dobijemo na korišćenje upravljačke brojeve sa glavnim brojem definisanim u argumentu *first* i sporednim brojevima koji se nalaze u opsegu $n, n+1, \ldots, n+count-1$, gde je n sporedni broj definisan u argumentu *first*

Dodeljivanje upravljačkih brojeva od strane OS-a

- Poslednji argument zapravo je ime uređaja koji bi trebalo da bude povezan sa opsegom zatraženih brojeva, i to ime će se pojaviti u /dev i SYSFS
- Kao i kod većine kernelovih funkcija, vrednost koju vraća funkcija *register_chrdev_region* će biti 0 ako je uspešno izvršena alokacija
- U slučaju greške, negativni kod greške će biti vraćen, i nećete imati na raspolaganju zahtevani opseg upravljačkih brojeva

Problem sa ovakvim pristupom?!?

- Funkcija *register_chrdev_region* radi dobro ako unapred znate tačno broj uređaja koji želite
- Često, međutim, nećete znati koji od glavnih brojeva su već zauzeti od strane operativnog sistema
- Kernel može, u skladu sa brojevima koji mu se nalaze na raspolaganju, izdvojiti glavni broj koji će vam dodeliti na korišćenje, ali morate ovaj zahtev uputiti korišćenjem sledeće funkcije

```
int alloc_chrdev_region (dev_t *dev,
    unsigned int firstminor,
    unsigned int count,
    char *name);
```

Šta ovo radi?

- U ovoj funkciji, *dev* je izlazni parametar koji će, po uspešnom završetku, sadržati prvi upravljački broj u određenom opsegu i zbog toga je prosleđen funkciji preko svog pokazivača
- Argument *firstminor* treba da zahteva prvi sporedni broj za korišćenje i on je obično 0
- Ostala dva argumenta funkcije su isti kao i kod već opisane funkcije *request_chrdev_region*

Kako se oslobađaju upravljački brojevi?

- Bez obzira kako su upravljački brojevi dodeljeni uređaju, trebalo bi da ih oslobodimo kada oni više nisu u upotrebi
- Upravljački brojevi uređaja se oslobađaju pomoću:

```
void unregister_chrdev_region(dev_t first,
    unsigned int count);
```

- Uobičajeno mesto za poziv *unregister_chrdev_region* je u funkciji za uklanjanje modula iz kernela

Dinamička raspodela upravljačkih brojeva

- U nekim slučajevima dinamička dodela upravljačkih brojeva je bolja opcija, u poređenju sa nasumičnim biranjem upravljačkog broja od onih koji su trenutno slobodni
- U takvim situacijama kod za drajver bi trebao da koristi funkciju *alloc_chrdev_region*, pre nego funkciju *register_chrdev_region*
- Nedostatak kod dinamičke dodele je, ipak, nemogućnost kreiranja čvorova u fajl sistemu koji će biti zaduženi za uređaj
- Da biste mogli da koristite drajver za neki uređaj, morate napraviti čvor u fajl sistemu preko koga će kernel znati koji drajver da koristi za vaš uređaj:

```
mknod /dev/ur0 c 240 0
```

mknod

```
mknod /dev/ur0 c 240 0
```

- Ovo zapravo pravi u fajl sistemu „fajl“ preko koga će se upravljati uređajem ur0
- Ono što je dalje značajno kada govorimo o fajlu ur0 to je da je on definisan kao čvor koji će upravljati char uređajem (zbog slova c koje se nalazi na mestu drugog argumenta komande mknod) i to char uređajem koji ima glavni upravljački broj 240 i sporedni 0
- Videćemo kasnije da na ovaj način Linux dozvoljava komunikaciju sa uređajem preko “upisa” u fajl i “čitanja” iz fajla, kao da upisujemo ili čitamo podatke iz nekog tekstualnog fajla
- Drajver registrovan sa glavnim upravljačkim brojem 240 i sporednim brojem 0 će biti zadužen da podatke koji se “upisuju” u fajl prosledi uređaju ur0, a da od uređaja ur0 traži podatke kada se vrši “čitanje”

Kako znati upravljačke brojeve kod dinamičke dodele?

- Očigledno je da je problem sa dinamičkom dodelom upravljačkih brojeva upravo u tome što ne znamo koji je glavni a koji sporedni broj dodeljen našem drajveru, pa samim tim ne možemo da kreiramo čvor u fajl sistemu preko koga ćemo upravljati našim uređajem
- Ipak, postoji način da se ovaj problem prevaziđe
- Kada je broj dodeljen drajveru, uvek ga možete pročitati u okviru fajl sistema na putanji /proc/devices
- Kada želimo da registrujemo driver koristeći dinamičku dodelu upravljačkog broja, pozivanje *insmod* može biti zamenjeno pozivanjem shell skripte koja nakon pozivanja *insmod*, čita /proc/devices, i nakon toga kreira odgovarajuće čvorove u /dev direktorijumu preko kojih će se pristupati uređaju

/proc/devices

- Tipičan /proc/devices fajl izgleda ovako:

```
Character devices:
1 mem
2 pty
3 ttyp
4 ttyS
10 misc
Block devices:
2 fd
8 sd
```

Shell skripta koja pomaže

```
#!/bin/sh
module="ur"
device="ur"
# invoke insmod with all arguments we got
# and use a pathname, as newer modutils
# don't look in . by default
/sbin/insmod ./$module.ko $* || exit 1
# remove stale nodes
rm -f /dev/${device}[0-3]
# find first argument in the line where
# second one is ur, this is major number
major=$(awk "\\$2= =\"$module\" {print \\$1}" \
    /proc/devices)
mknod /dev/${device}0 c $major 0
mknod /dev/${device}1 c $major 1
mknod /dev/${device}2 c $major 2
mknod /dev/${device}3 c $major 3
```

Bitne strukture

- Četiri strukture su značajne prilikom implementacije i korišćenja drajvera za sekvencijalne uređaje:

1. file_operations
2. file
3. inode
4. cdev

file_operations

- Ova struktura je zadužena za sve operacije koje želimo da vršimo na uređaju za koji pišemo drajver
- Polja ove strukture su pointeri na funkcije i funkcije "postavljene" na ta mesta će biti pozivane od strane kernel-a kada za to dođe vreme
- Ovo nam omogućava da na jednostavan način uključimo drajver u već postojeći kompleksan i obiman kernel kod
- Na narednim slajdovima su navedena polja strukture
- O njima ćemo mnogo detaljnije pričati u nastavku, za sada ćemo samo navesti neka od njih

file_operations, polja

```
struct module *owner
loff_t (*llseek)(struct file *, loff_t, int);
ssize_t (*read)(struct file *,char __user *,
    size_t, loff_t *);
ssize_t (*write)(struct file *,const char __user *,
    size_t, loff_t);
int (*ioctl)(struct inode *, struct file *,
    unsigned int, unsigned long);
int (*mmap)(struct file *, struct vm_area_struct *);
```

file_operations, polja

```
int (*open)(struct inode *, struct file *);
int (*release)(struct inode *, struct file *);
int (*fasync)(int, struct file *, int);
```

Primer korišćenja file_operations strukture

```
struct file_operations device_fops =
{
    .owner = THIS_MODULE,
    .llseek = device_llseek,
    .read = device_read,
    .write = device_write,
    .ioctl = device_ioctl,
    .open = device_open,
    .release = device_release,
};
```

file_operations

- Ako se ovako definiše struktura *device_fops* tipa *file_operations*, svako od polja (sa izuzetkom naravno *owner* polja) mora biti posebno definisano i deklarisano kao funkcija
- Nakon što se to uradi svaki sistemski poziv na fajlu koji predstavlja naš uređaj će biti preusmeren na "naše" funkcije
- Na primer, otvaranje fajla odgovarajućeg uređaja, automatski poziva funkciju *device_open*, dok čitanje uređaja poziva *device_write* funkciju

file struktura

- Nema nikakve veze sa FILE strukturom definisanom u C biblioteci i ona se nikada ne pojavljuje u kernelu
- Sa druge strane struktura *file* se nikada ne pojavljuje u korisničkom (aplikacionom) delu
- Ima nekoliko polja od kojih je jedno polje f_op:

```
struct file_operations * f_op;
```

- na ovo polje će kernel postaviti *file_operations* strukturu koju smo prethodno popunili i povezali sa *cdev* strukturom (videćemo za par slajdova na koji način)
- Prilikom svakog konkurentnog otvaranja fajla koji predstavlja uređaj, kernel će kreirati novi fajl deskriptor na osnovu kojeg će znati koje operacije treba da poziva prilikom sistemskih poziva na tom fajlu (čitanje, upis, otvaranje, zatvaranje,...)

inode struktura

- Struktura *inode* se koristi interno od strane kernela da predstavi fajl
- Samim tim je ona drugačija od *file* strukture koje predstavlja deskriptor otvorenog fajla
- Razlika između *inode* i *file* struktura ogleda se u tome da mogu postojati nekoliko *file* struktura koje predstavljaju višestruke deskriptore istog otvorenog fajla, ali sve one zapravo referenciraju na jednu istu inode strukturu

inode struktura

- Polja inode strukture koja su nama sada od značaja su:

```
dev_t i_rdev;
struct cdev * i_cdev;
```

- Registrovanjem *cdev* strukture koja predstavlja naš drajver u kernelu (videćemo uskoro kako) kernel će povezati *cdev* strukturu sa onim fajlom koji ima iste upravljačke brojeve kao i *cdev* struktura
- Na ovaj način se zaokružuje povezanost fajla u fajl sistemu, sa *cdev* strukturom (drajverom) i operacijama koje se na njemu mogu izvršavati kroz sistemske pozive na asociranom fajlu

Struct cdev

- Kernel koristi strukture tipa *struct cdev* da predstavlja char uređaje interno
- Pre nego što kernel poziva operacije, morate dodeliti i registrovati jednu ili više ovih struktura
- Da biste to uradili, kod treba da uključi <linux/cdev.h>, gde je definisana struktura
- Korisnik kreira *cdev* strukturu tokom izvršavanja (u runtime-u), pomoću sledećeg koda:

```
struct cdev *my_cdev = cdev_alloc();
my_cdev->ops = &my_fops;
```

- Alternativno, isti posao može da se uradi i na sledeći način:

```
cdev_init(my_cdev, &my_fops);
```

Struct cdev

- U prethodnom primeru, najpre je alociran prostor za strukturu tipa *cdev* koja se zove *my_cdev*, a zatim je za tu strukturu vezana instanca strukture *file_operations* koja se zove *my_fops* (ovde je, na primer, mogla da se iskoristi u prethodnom primeru kreirana struktura *device_ops*).
- Osim postavljanja polja ops strukture tipa *cdev*, drugo polje koje mora da se postavi jeste, kao i kod strukture *file_operations*, *owner* polje (koje takođe u većini slučajeva treba da bude THIS_MODULE)
- Kada su polja cdev strukture postavljena, poslednji korak je da kernel obavi registrovanje upravo kreirane strukture:

```
int cdev_add(struct cdev *dev, dev_t num,
    unsigned int count);
```

- Uklanjanje cdev strukture iz kernela vrši se na sledeći način:

```
void cdev_del(struct cdev *dev);
```