Metody připojování periferií

BI-MPP

Přednáška 8

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Jádro operačního systému Linux
- Moduly jádra
- Ovladač znakového zařízení

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- Corbet, J., Rubini A., Kroah-Hartman, G.: Linux Device Drivers. Third Edition. O'Reilly Media 2005. ISBN: 0-596-00590-3.

Jádro operačního systému Linux

- Napsané v jazyce C, pouze několik málo specifických částí je napsáno v assembleru
- Jádro je volně ke stažení ve zdrojové formě a je šířeno pod GNU licencí
- Podporuje velké množství jak procesorů (x86, PPC, ARM, …), tak platforem (PC, PDA, …)
- Obsahuje ovladače pro velké množství zařízení
- Překládá se překladačem GCC a umožňuje křížovou kompilaci (např. na PC můžeme přeložit jádro pro procesor ARM)

Konfigurace jádra

- Jádro Linuxu vyžaduje před překladem konfiguraci, která vybírá
  - Procesor
  - Platformu
  - Ovladače
- U ovladačů je možno v konfiguraci zvolit
  - nezařadit (nekompilovat)
  - zakompilovat do jádra (ovladače nezbytné pro start OS, např. ovladače disků, podpora file systémů)
  - Přeložit jako moduly jádra (možnost dynamického zavádění)

Moduly jádra Linuxu

- Ovladače jsou moduly do jádra operačního systému Linux
- Moduly jsou psané v jazyce C (kritické části mohou být i v assembleru, ale je to spíše výjimečné)
- Modul je překládán až do úrovně object souboru (*.o) a proto berou moduly jádra příponu *.ko.
- Modul ve formě object souboru je přilinkován za běhu do jádra (pozor ! *.ko není totéž jako *.so, protože *.so je produktem linkeru (ld))

Příklad modulu

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE(“Dual BSD/GPL“)

static int mo_init(void) {
        printk(KERN_ALERT “inicializace\n”);
}

static int mo_exit(void) {
        printk(KERN_ALERT “ukonceni\n”);
}

module_init(mo_init)
module_exit(mo_exit)
```

Komentář k příkladu modulu

- Makro module_init registruje funkci mo_init jako inicializační funkci. Tato funkce je volána jen jednou, když je zaveden modul do jádra příklazem insmod nebo modprobe.
- Makro module_exit registruje funkci mo_exit jako deinicializační funkci. Tato funkce je volána jen jednou, když je modul odstraněn z jádra příklazem rmmod.
- Funkce printk vytiskne zprávu, kterou je možno zobrazit pomocí příkazu dmesg. Velmi užitečné pro ladění driverů.
- Makro MODULE_LICENCE registruje název použité licence. Text se stává součástí kódu modulu.

Překlad modulu mimo jádro a zavedení modulu do jádra

Makefile

Jméno(a) modulu(ů) s příponou *.o, musí se shodovat se jménem zdrojového souboru (mpp_module.c)

```
obj-m := mpp_module.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

Překlad modulu

```
make
```

Zavedení modulu

```
insmod mpp_module.ko
```

Registrace znakového zařízení

```
struct file_operations
mydev_fops = {
.owner = THIS_MODULE,
.llseek = mydev_llseek,
.read = mydev_read,
.write = mydev_write,
.ioctl = mydev_ioctl,
.open = mydev_open,
.release = mydev_release,
};
```

Struktura file_operations obsahuje ukazatele na callback funkce, které jsou implementacemi souborových operací na straně driveru

Registrace znakového zařízení (v probe)

```
int register_chrdev(unsigned int major, const char *name,
struct file_operations *fops);
```

Funkce souborových operací I

Otevření souboru (open)
[vrací 0 po uspěšně provedené operaci, jinak chybový kód]

```
static int mpp_open(struct inode *inode, struct file *filp)
{ ... return 0; }
```

Uzavření souboru (close)
[vrací 0 po uspěšně provedené operaci, jinak chybový kód]

```
static int mpp_release(struct inode *inode, struct file *f)
{ ... return 0; }
```

Funkce souborových operací II

Čtení souboru (read)
[vrací počet přečtených bytů, tj. bytů zapsaných do bufferu. V ideálním případě by funkce měla vrátit size, tj. počet bytů požadovaných aplikací. Není to ale podmínkou.]

```
static int mpp_read(struct file *f, char *buf,
   size_t size, loff_t *offset)
{ ... return size; }
```

Zápis do souboru (write)
[vrací počet zapsaných bytů, tj. bytů přečtených z bufferu. V ideálním případě by funkce měla vrátit size, tj. počet bytů požadovaných aplikací k zapsání. Není to ale podmínkou.]

```
static int mpp_write(struct file *f, const char *buf,
       size_t size, loff_t *offset)
{ ... return size; }
```

User Space a Kernel Space

- Aplikace běží v uživatelském prostoru (UserSpace), který má řadu omezení (ochrany paměti např.)
- Jádro a tedy i ovladače běží v prostoru jádra, který má všechna privilegia
- V prostoru jádra běží program supervizorském režimu a aplikace běží v uživatelském režimu
- Při volání jádra (system call, např. kvůli souborové operaci) dochází k přepnutí z User Space na Kernel Space a při návratu zase opačně
- Uvědomte si, že User Space a Kernel Space reprezentují odlišné paměťové prostory, tudíž ukazatele (pointry) v jednom prostoru neplatí v druhém a naopak.

Kopírování dat
kernel space ↔ userspace

Kernel space → user space

```
unsigned long copy_to_user(void __user *to,
const void *from,
unsigned long count);
```

User space → kernel space

```
unsigned long copy_from_user(void *to,
const void __user *from,
unsigned long count);
```

Nejčastěji se používají v callback funkcích pro souborové operace read a write

Alokace dynamické paměti v jádře

Alokace paměťového bloku

```
char* kbuf = kmalloc(size, GFP_KERNEL);
```

Typ bloku paměti

Velikost

Uvolnění paměťového bloku

```
kfree(kbuf);
```

Komunikace s ovladačem ze strany aplikace (bash)

Zápis

```
echo -n "ahoj" >/dev/mpp
```

Čtení

```
cat /dev/mpp
```

Komunikace s ovladačem ze strany aplikace (programově - C)

Prototypy funkcí

```
#include <unistd.h>

int open(const char *pathname, int flags);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int close(int fd);
```

Příklad

```
#include <unistd.h>

int main() {
  char buf[100];
  int fd = open("/dev/mpp", O_RDWR);
  read(fd, buf, 10);
  write(fd, buf, 10);
  close(fd);
}
```