

# PB173 – Ovladače jádra – Linux

## II. Komunikace

Jiri Slaby

Fakulta informatiky  
Masarykova univerzita

1. 10. 2015

- Kolokvium za DÚ
  - DÚ do příštího cvičení
- Login/heslo
  - vyvoj/vyvoj
- GIT: <http://github.com/jirislaby/pb173>
  - `git pull --rebase`
- Studijní materiály v ISu

# Sekce 1

Komunikace jádro ↔ uživatelský prostor

## 1 Voláním funkce: system call (syscall)

- Skok do jádra speciální instrukcí (x86\_64: `syscall`)
  - O skok se stará libc (`fwrite`→`write`→`syscall`→instrukce)
  - Drahá operace (přepnutí kontextu)
- Do registru se uloží číslo operace
  - V libc: převodní „tabulka“ funkce→číslo
  - V jádře: převodní tabulka číslo→funkce
- `syscall(__NR_fork)`
- Demo: `__NR_fork` v `unistd.h`; příklady z `pb173/02/syscalls/`

## 2 Speciální syscall

- Knihovna přilinkovaná jádrem
- `vdso.so`

**Každá nová funkcionalita = nový syscall**

## V userspace: pomocí *funkce* `syscall` vypsat nějakou informaci

- 1 Inspirujte se v `pb173/02/syscalls/c.c`
- 2 Použijte `syscall` a `__NR_write`

Je třeba znát prototypy funkcí.

- Většinou jsou dokumentované: `man write`
- Jinak použít `lxr` a najít implementaci v jádře

- ③ Speciální soubory v /dev
  - Komunikace přes soubor
    - Standardní operace – open, read, ...
    - Tj. není nutný nový syscall
  - Identifikované trojicí blokové/znakové, major a minor číslo
    - Většinou major=ovladač, minor=zařízení (tty: c, maj 4, min 0–63)
  - Blokové (disky apod.)
    - Komunikace po blocích
    - Nebudeme se jimi zabývat (popsány v LDD)
  - Znakové (ostatní)
    - Komunikace po znacích (bajtech)
    - *Zbytek tohoto cvičení*
  - Seznam rezervovaných v `Documentation/devices.txt` a `/proc/devices`
- ④ Sockety, roury, ...

## Sekce 2

# Znaková zařízení

`struct file_operations` (`linux/fs.h`, LDD3 3. kapitola)

- Funkce jako:

```
ssize_t (*write)(struct file *file, const char __user *buf, size_t count, loff_t *offp)
```

- `file->private_data` slouží programátorovi (libovolně)
- `offp` slouží programátorovi (k poznamenání průběhu)
- `__user` značí ukazatel od uživatele (*POZOR* tomu nevěříme)
- Návratové hodnoty (podle návratového typu funkce)
  - `int` – záporné = -Echyba, jinak 0
  - `ssize_t` – záporné = -Echyba, jinak počet zpracovaných znaků
  - Vždy podle standardu (např. POSIX)



- 1 Najděte všechny možné funkce, které lze obsluhovat
  - Najděte `struct file_operations` v `lxr`
  - Projděte strukturu
- 2 Najděte možné chybové návratové hodnoty
  - Najděte `EPERM` v `lxr`
  - Projděte ostatní
  - Podívejte se do `man write` na možné návratové hodnoty

# Příklad operací

```
struct file_operations my_fops = {  
    .owner = THIS_MODULE,  
    .open = my_open,  
    .write = my_write,  
};
```

```
int my_open(struct inode *inode, struct file *filp) {  
    filp->private_data = 3;  
    return 0;  
}
```

```
ssize_t my_write(struct file *filp, const char __user *buf, size_t count, loff_t *offp) {  
    /* filp->private_data is 3 here */  
    if (count == 0)  
        return -EINVAL;  
    *offp += count; /* remember position for the next time */  
    return count; /* "written" all */  
}
```

## Obsluha `open`, `read`, `write`, `release` (tj. `close`)

- 1 Definice `struct file_operations`
- 2 Vytvoření funkcí dle prototypu z `file_operations` (lxr)
  - `Open` a `release` s nějakým `printk` a `return`
  - `Read` a `write` prozatím jen `return`
  - Návrátové hodnoty
    - `Open` a `release` vracejí 0 (žádná chyba)
    - `Read` též (0 znamená EOF)
    - `Write` vrací `count` (zapsáno vše)
- 3 Jen zkuste přeložit

# Vytvoření znakového zařízení

- LDD3 3. a 6. kapitola
- 2–3 kroky
  - 1 Registrace rozsahu major+minor (v `module_init`)
    - `alloc_chrdev_region`, `register_chrdev_region`, `unregister_chrdev_region` (`linux/fs.h`)
    - Přidání záznamu do `/proc/devices`
  - 2 Registrace jednotlivých minorů (při objevení zařízení)
    - `cdev_init`, `cdev_add`, `cdev_del` (`linux/cdev.h`)
    - Parametr pro `cdev_init` je `struct file_operations`
    - Po odpovídajícím `mknod` lze zařízení používat
  - 3 Podat zprávu `udev` (vytvoření `/dev/*`) – nepovinné
    - `device_create`, `device_destroy` (`linux/device.h`)
    - Předem je potřeba vytvořit `class` (`module_init`)

## Vytvoření znakového zařízení

- 1 Definujte si globální `struct cdev`
- 2 V `module_init`
  - Zavolejte `alloc_chrdev_region(&dev, 0, 10, "jmeno")`
    - `dev` je návratová hodnota typu `dev_t`
    - Vypište si `MAJOR(dev)` a `MINOR(dev)`
  - Zavolejte `cdev_init` na svou globální `struct cdev`
  - Zavolejte `cdev_add` na svou globální `struct cdev` a `dev`
- 3 V `module_exit`
  - `cdev_del`
  - `unregister_chrdev_region`
- 4 `make a insmod`
- 5 Ověřte vytvoření v `/proc/devices`
- 6 Podle výpisu z 2. bodu: `mknod soubor c MAJ MIN`
- 7 Vyzkoušejte `cat` soubor

- Něco, čemu nelze věřit (NULL, ukazatel do tabulek oprávnění, ...)
- Nutnost kontroly *POZOR*
- `copy_from_user`, `copy_to_user`
  - „`memcpy`” s kontrolou
  - Vracejí počet *NEzkopírovaných* znaků (0 znamená OK)
- `get_user`, `put_user`
  - „`var = *buf`” a „`*buf = var`” s kontrolou
  - Tzn. jen pro primitiva (`char`, `short`, `int`, `long`)
  - Vracejí 0 nebo chybu (záporná hodnota)
- Definované v `linux/uaccess.h`

Demo: pb173/02

## Dopsat těla funkcí `read` a `write` tak, aby zpracovávala data

- 1 `write` vypíše uživatelský buffer pomocí `printf`
  - `copy_from_user` (při chybě `return -EFAULT`)
  - Nezapomeňte ukončit zkopírovaný řetězec použitím `\0`
- 2 `read` bude vracet řetězec „Ahoj“
  - `copy_to_user`

## Misc vrstva

- **Stačí-li 1 minor** (málokdy)
- Dělá všechnu práci včetně volání `udev`
- Potřebujeme
  - Definici misc zařízení (`struct miscdevice`)
  - Opět `struct file_operations`
  - `struct cdev` není třeba
- `misc_register, misc_deregister`
- Objeví se v `/proc/misc` a `/dev`
- `linux/miscdevice.h`



## Vytvořit misc zařízení

- 1 Definice `struct miscdevice`
  - `minor` = `MISC_DYNAMIC_MINOR` (automaticky přidělit)
  - `name` – jméno vytvářeného zařízení
  - `fops` – vaše `struct file_operations`
- 2 `misc_register/deregister` do `module_init/exit`
- 3 `make a insmod`
- 4 Ověřte vytvoření v `/proc/misc`
- 5 Vyzkoušejte `cat /dev/<name>` (jméno shora)

- Jádro a proces může běžet s různými bitovými šířkami (32, 64-bit)
- Problém s ukazateli a long proměnnými
  - Jiná délka dat
  - Jiné zarovnání struktur

Uživatel (32-bit)		Jádro (64-bit)
data ... 4 bajty na offsetu 4	<pre>struct my {     char x;     void *data; };</pre>	data ... 8 bajtů na offsetu 8

# Pevné typy v jádře

z hlediska počtu bitů

- `linux/types.h`
- `__u8`, `__u16`, `__u32`, `__u64`
- `__s8`, `__s16`, `__s32`, `__s64`
- Ukazatele musí být v `union` s `__u64`

```
struct bad {  
    unsigned long flags;  
    short index;  
    void *data;  
} my;
```

⇒

```
struct good {  
    __u64 flags;  
    __s16 index;  
    union {  
        void *data;  
        __u64 filler ;  
    };  
} my;
```

```
read(fd, &my, sizeof(my));  
ioctl(fd, DO_SOMETHING, &my);
```