PB173 – Ovladače jádra – Linux

VI. Datové typy

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

25. 10. 2016

Obsah cvičení

Datové typy

LDD3 kap. 11

- Základní typy
- Endianita (pořadí bytů)
- Datové struktury (seznam, FIFO)

Sekce 1

Základní typy

Základní typy

- `char`, `short`, `int`, `long`, `long long` (+ `unsigned`)
- `u8`, `u16`, `u32`, `u64`, `__u8`, `__u16`, `__u32`, `__u64`
- `s8`, `s16`, `s32`, `s64`, `__s8`, `__s16`, `__s32`, `__s64`

`sizeof` základních typů pro různé architektury

Architektura	`char`	`short`	`int`	`long`	`void *`	`long long`
alpha	1	2	4	8	8	8
arm	1	2	4	4	4	8
ia64	1	2	4	8	8	8
m68k	1	2	4	4	4	8
mips	1	2	4	4	4	8
ppc	1	2	4	4	4	8
sparc	1	2	4	4	4	8
sparc64	1	2	4	8	8	8
x86_32	1	2	4	4	4	8
x86_64	1	2	4	8	8	8

Základní typy

`sizeof` dalších typů (ukazatelů)

Architektura	`long`	`void *`	phys
x86_32	4	4	4
x86_32+PAE	4	4	4.5+
x86_64	8	8	8

Pro práci s fyzickou adresou `long` nebo `void *` nestačí

- Ale `u64` nebo `long long` může být zbytečné (čisté 32-bity)
- Proto speciální typy (*POZOR*)
  - `phys_addr_t` (RAM, za MMU)
  - Anebo `long`, ale pak v něm uchovávat jen `pfn` (`phys/PAGE_SIZE`)
  - `dma_addr_t` (RAM, před IOMMU)
  - `resource_size_t` (PCI prostor apod.)

Úkol: opravte typy v pb173/06 (jen *první* TODO)

Pořadí bytů

Číslo `0x12345678` reprezentované v paměti

Pořadí	**Příklad architektury**	**Offset v paměti** 0	1	2	3
Big-endian	`arm`, `s390`	0x12	0x34	0x56	0x78
Little-endian	`arm`, `x86`	0x78	0x56	0x34	0x12
Mixed-endian	`PDP-11`	0x34	0x12	0x78	0x56

Pro typy o velikosti větší než 1 je nutné znát pořadí

- Interpretace dat z/do zařízení
- Specifikace
  - CPU (příklady nahoře)
  - PCI: LE
  - Síťový provoz: BE
  - ...

Funkce pro pořadí bytů

- `asm/byteorder.h`
- `__leXX`, `__beXX`
- `XX` $\in \{16, 32, 64\}$
- `cpu_to_leXX`, `cpu_to_beXX`
- `leXX_to_cpu`, `beXX_to_cpu`
- `ntohs`, `ntohl`, `htons`, `htonl`

Úkol: doplňte tělo `decode_and_print` v pb173/06 (druhé TODO)

Chyby jako ukazatele

- `void *funkce()`
- Jak vrátit chytřejší chybu než NULL?
  - V uživatelském prostoru máme `errno` a ekvivalenty
- Díra na konci adresového prostoru
  - Tj. nevalidní ukazatel (dereference by způsobila pád)
  - Např. na `x86_64`: `0xfffffffffff00000-0xffffffffffffffff`
  - Lze použít k zakódování chyby
- `void *ptr = ERR_PTR(-Eerror)`
- `IS_ERR(ptr) == true` ⇒ `int err = PTR_ERR(ptr)`
- `IS_ERR(NULL)` je `false`!

Úkol: přepište `error_handling` v pb173/06 (třetí TODO)

Sekce 2

Datové struktury

Seznam I.

- `linux/list.h`
- `struct list_head` (obsahuje `prev`, `next`)
  - *Jak samotný seznam (počátek), tak jeho prvky*
- `LIST_HEAD(my_list)`$_S$, `INIT_LIST_HEAD(&my_list)`$_D$
- `list_add(co, kam)`, `list_add_tail(co, kam)`, `list_del(co)`
- `list_move(co, kam)`, `list_move_tail(co, kam)`

Příklad

```
struct my_struct {
  int a;
  struct list_head list ;
};
LIST_HEAD(my_list);
```

```
struct my_struct *s;
s = kmalloc(sizeof(*s), GFP_KERNEL);
if (!s) { ... }
s->a = 10;
list_add(&s->list, &my_list); /* or list_add_tail (&s->list, &my_list); */
```

Úkol: naalokujte 20 stránek a v seznamu si je pamatujte.

Seznam II.

- `linux/list.h`
- `list_empty(list)`
- Jedna položka: `list_entry`, `list_first_entry`
- Iterace: `list_for_each_entry(...) { ... }` (jako `for`), `list_for_each_entry_reverse(...) { ... }`
- `*_safe` varianty – pokud měním aktuální člen seznamu

Příklad

```
struct my_struct *s, *s1;
s = list_first_entry (&my_list, struct my_struct, list );
s = list_entry (s->list.next, struct my_struct, list );

list_for_each_entry(s, &my_list, list ) { s->a; }
list_for_each_entry_safe(s, s1, &my_list, list ) { list_del (&s->list); }
```

Úkol: předchozích 20 stránek uvolněte.

FIFO

- `linux/kfifo.h`, `samples/kfifo/*`
- `struct kfifo`
  - Statické: `DECLARE_KFIFO` + `INIT_KFIFO`
  - Dynamické: `kfifo_alloc(&fifo, size, GFP_*)`, `kfifo_free(&fifo)`
- Zápis: `kfifo_in(&fifo, buf, count)`
- Čtení: `kfifo_out(&fifo, buf, count)`
- Obsazeno: `kfifo_len(&fifo)`
- Volno: `kfifo_avail(&fifo)`

Úkol

Práce s FIFO (`linux/kfifo.h`)

1. Globální FIFO 8 intů (`DECLARE_KFIFO`)
2. `INIT_KFIFO`
3. Zapisovat čísla (`kfifo_in`) dokud je místo
4. Vypsat všechna čísla (`kfifo_out` + `kfifo_len`)

Pozn.: ve starších jádrech ($<$ 2.6.33) je jiné rozhraní:

- jen dynamicky: `my_fifo = kfifo_alloc`
- `__kfifo_put` je `kfifo_in`
- `__kfifo_get` je `kfifo_out`
- `__kfifo_len` je `kfifo_len`
- `avail` není