

PB173 – Ovladače jádra – Linux

VI.

Jiri Slaby

ITI, Fakulta Informatiky

25. 10. 2012

LDD3 kap. 11

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

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

arch	char	short	int	long	ptr	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

Tabulka: sizeof základních typů pro různé architektury

arch	long	virt	phys
x86_32	4	4	4
x86_32+PAE	4	4	4.5+
x86_64	8	8	8

Tabulka: `sizeof` dalších typů (ukazatelů)

Pro práci s fyzickou adresou `long`/ukazatel nestačí

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

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

Tabulka: Číslo 0x12345678 v paměti

Adresa:	0	1	2	3
Big-endian	0x12	0x34	0x56	0x78
Little-endian	0x78	0x56	0x34	0x12
Mixed-endian	0x34	0x12	0x78	0x56

Pro typy o velikosti větší než 1 je nutné vědět pořadí

- Specifikace
 - x86: LE
 - PowerPC: LE nebo BE (dle nastavení CPU)
 - PCI: LE
 - Síťový provoz: BE

Funkce pro pořadí bytů

- **asm/byteorder.h**
- `__leXX, __beXX`
- `cpu_to_leXX, cpu_to_beXX`
- `leXX_to_cpu, beXX_to_cpu`
- $XX \in \{16, 32, 64\}$
- `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`?
- Díra na konci adresového prostoru
 - Tj. nevalidní ukazatel (dereference by způsobila pád)
 - Např. `0xffffffffffffffff00000-0xffffffffffffffffffff` (x86_64)
 - Lze použít k zakódování chyby
- `void *ptr=ERR_PTR(-Error)`
- `IS_ERR(ptr) ⇒ int err=PTR_ERR(ptr)`
- `IS_ERR(NULL)=false`

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

Seznam I.

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

```
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) { ... }  
list_add(&s->list, &my_list);  
/* list_add_tail(&s->list, &my_list); */
```

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

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

```
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);  
/* or */  
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.

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

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 Vypsát všechna čísla (`kfifo_out + kfifo_len`)

Pozn.: ve starších jádrech je jiné rozhraní:

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