Pokročilé operační systémy

Přístup k disku, OOP

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Přístup k disku

- několik různých fyzických rozhraní (PATA, SATA, PATAPI, SATAPI)
- několik různých způsobů adresace (CHS, LBA28/48)
- několik různých způsobů komunikace (PIO, DMA, AHCI)

PATA + LBA28 + PIO

- historicky patří ke starším řešením
- jednoduché na implementaci
- suveréně nejpomalejší
- dvě sběrnice (primary/secondary)
- na každé sběrnici maximálně dvě zařízení (master/slave)
- komunikace přes PMIO

Porty (1/2)

sběrnice	**io base**		**control base**	
primary	0x1f0	až 0x1f7	0x3f6	až 0x3f7
secondary	0x170	až 0x177	0x376	až 0x377

io base +

	režim	**popis**	**význam**	vel.
0	R/W	Data Register	čtení/zápis dat	16
1	R	Error Register	chybové kódy	8
1	W	Features Reg,	informace pro specifické příkazy	8
2	R/W	Sector Cnt. Reg.	počet sektorů k přenosu	8
3	R/W	LBAlo	spodních 8 bitů adresy LBA	8
4	R/W	LBAmid	prostředních 8 bitů adresy LBA	8
5	R/W	LBAhi	vyšších 8 bitů adresy LBA	8
6	R/W	Drive/Head Reg.	nejvyšší 4 bity adresy LBA, příznaky, volba zařízení	8
7	R	Status Reg.	stavový registr	8
7	W	Command Reg.	příkazový registr	8

Porty (2/2)

Stavový registr (io base + 7)

bit	id	popis
0	ERR	indikace chyby, je nutné poslat příkaz znovu nebo udělat soft. reset
1	IDX	0
2	CORR	0
3	DRQ	zařízení je připraveno odesílat nebo přijímat data
4	SRV	Overlapped Mode Service Request. ???
5	DF	chyba zařízení
6	RDY	zařízení je připraveno
7	BSY	zařízení je zaneprázdněno (zpracovává příchozí nebo odchozí data)

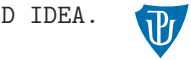
Komunikace s diskem (viz zdrojové kódy)

- soft. reset (inicializace zařízení)
- detekce zařízení
- čtení/zápis
  1. na jednotlivé porty se zapíše velikost a adresa dat (nutné rozložit) + zvolí se zařízení
  2. zašle se příkaz pro čtení/zápis
  3. data se čtou/zapisují z odpovídajícího datového portu (io base + 0)
- nutné testovat příznaky BSY, RDY, DRQ
- u jednotlivých operací je nutné počítat se zpožděním (400ns)
- doporučeno několikrát přečíst stavový registr a věnovat pozornost až poslední hodnotě

„Objektové programování" na úrovni jádra

- data + operace s nimi (struktury, funkce)
- co nabízí OOP
  1. zapouzdření
  2. polymorfismus
  3. (dědičnost)
- každá vlastnost má určitou režii
- nejsou potřeba všechny tři vlastnosti současně
- není potřeba podpora jazyka (je C++ pro jádro dobrý nápad?)

It sucks. Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.



The fact is, C++ compilers are not trustworthy. They were even worse in 1992, but some fundamental facts haven't changed:

- the whole C++ exception handling thing is fundamentally broken. It's _especially_ broken for kernels.
- any compiler or language that likes to hide things like memory allocations behind your back just isn't a good choice for a kernel.
- you can write object-oriented code (useful for filesystems etc) in C, _without_ the crap that is C++.

In general, I'd say that anybody who designs his kernel modules for C++ is either
(a) looking for problems
(b) a C++ bigot that can't see what he is writing is really just C anyway
(c) was given an assignment in CS class to do so.

Feel free to make up (d).

Linus

Modularizace a zapouzdření

- objekt reprezentován strukturou

```
struct term {
   unsigned char buf[]; // buffer terminalu
   unsigned int cur_x;  // pozice kurzoru
   unsigned int cur_y;
};
```

- jednotlivé „metody" přebírají odkaz na tuto strukturu

```
void term_init(struct term *term);
void term_putc(struct term *term, char c);
```

- samostatná metoda init (bez alokace)
- minimální režie, znalost volaných funkcí v době překladu (early-binding)

Polymorfismus (chování)

- jednotlivé metody přesunuty do dané struktury (ukazatele na funkce)
- objekt předáván jako první argument

```
struct term {
   unsigned char buf[]; // buffer terminalu
   unsigned int cur_x;  // pozice kurzoru
   unsigned int cur_y;
   void (*putc)(struct term *term, char c);
};
struct term *term;
term->putc(term, 'A');
```

- překladač neví, která funkce bude volána (late-binding)
- má urč režii, je nutné inicilizovat ukazatele na funkce (místo v paměti, čas CPU)
- sloučení souvisejících metod do jedné struktury/hodnoty, sdílení mezi jednotlivými instancemi (režie další dereference)

Polymorfismus a data

- `union` má smysl použít, pokud je tříd malé, konečné a dopředu známé množství a jednotlivé třídy mají přibližně stejnou velikost
- vnoření jednotlivých struktur

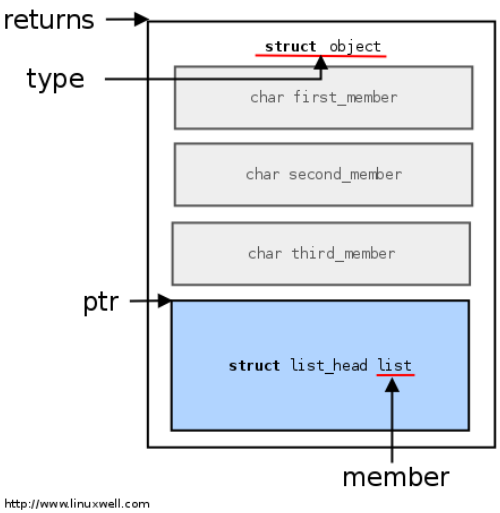
```
struct net_term {
   struct term term;
   uint32_t ip;
   uint16_t port;
};
```

- lze využít rozložení v paměti a přetypování.
- Linux makro `container_of`:

```
#define container_of(ptr, type, member) ({                      \
        const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
        (type *)( (char *)__mptr - offsetof(type,member) );})
```

container_of(ptr, type, member)

illustrated explanation



https://radek.io/2012/11/10/magical-container_of-macro/

Úkoly

Refactoring

- ověřte, že současný kód funguje správně
- upravte provizorní kód v souborech `ata.c` a `ata.h` tak, aby dokázal procovat s oběma sběrnicemi, na kterých jsou připojeny oba disky
- podrobnější vysvětlení hodnot v kódu najdete na `https://wiki.osdev.org/ATA_PIO_Mode`

Blokové zařízení (bonusový úkol)

- zkuste **v klasickém C v uživatelském prostoru** navrhnout třídu/strukturu pro obecné blokové zařízení schopné číst a adresovat jednotlivé bloky dat
- vytvořte potomka této třídy, který bude bloky dat číst a zapisovat z/do souboru