PB173 – Ovladače jádra – Linux

X. DMA

Jiri Slaby

ITI, Fakulta informatiky

26. 11. 2013

Paměť jinak II.

LDD3 kap. 15 (zastaralá)

- I. (minule): Mapování paměti jádra (mmap)
- II. (dnes): Přímý přístup do paměti (DMA)

DMA

Přímý přístup do paměti

- Prozatím jsme ze/do zařízení četli/zapisovali přes CPU
  - Tj. standardními operacemi s (přemapovanou) pamětí
- Velké přenosy = velká zátěž CPU
  - Cykly, čekání na pomalou sběrnici atd.
- Místo toho naprogramujeme HW, aby přenášel data sám
  - Nutná podpora HW (sběrnice – arbitrace, zařízení – přenosy)

Princip DMA

Princip

1. Alokace (speciální) paměti
2. *Pro TX*: vyplnění daty (paket, zvuk, data na disk, . . .)
3. Předání ukazatele do zařízení
4. Odstartování přenosu v zařízení
5. Přerušení či jiná signalizace konce přenosu od zařízení
6. *Pro RX*: práce s příchozími daty (paket, data z disku, . . .)

DMA v Linuxu

API alokace

- `linux/dma-mapping.h`, `Documentation/DMA-*`
- `dma_alloc_coherent`, `dma_free_coherent`

```
void *dma_alloc_coherent(struct device *dev, size_t size, dma_addr_t
*dma_handle, gfp_t gfp)
```

- `dev` – zařízení, které bude k paměti přistupovat (`NULL` pokud neznáme)
- `size` – velikost, jakou požadujeme
- `dma_handle` – fyzická adresa (návratová hodnota) – pro zařízení
- návratová hodnota – virtuální adresa – pro nás

DMA v Linuxu – příklad

```
int my_do_DMA(struct device *dev)
{
  dma_addr_t phys;
  char *virt;

   virt = dma_alloc_coherent(dev, 100, &phys, GFP_KERNEL);
   if (! virt )
     return -ENOMEM;
   memset(virt, 0, 100);
   my_HW_set_addr(dev, phys);
   my_HW_start_transfer(dev);
   while (my_HW_working(dev)) /* polling */
     msleep(100);
   printk(KERN_INFO "%s\n", virt);
   return 0;
}
```

Úkol: alokujte a uvolněte pomocí `dma_*` 1 stránku paměti (v ovladači EDU z předminula)

DMA pro PCI

Navíc

- Nastavení masky adres
  - Ne všechna zařízení zvládnou adresovat celý fyzický prostor
  - `pci_set_dma_mask(pdev, DMA_BIT_MASK(27))`
- Zapnutí „spravování sběrnice”
  - Tj. zařízení umí samo iniciovat přenosy apod.
  - `pci_set_master(pdev)`

```
int my_probe(struct pci_dev *pdev, ...)
{
  dma_addr_t phys;
  void * virt ;
   ... /* enable etc. here */
  ret = pci_set_dma_mask(pdev, DMA_BIT_MASK(27));

  pci_set_master(pdev);

   virt = dma_alloc_coherent(&pdev->dev, 100, &phys, GFP_KERNEL);
   ...
}
```

EDU a DMA

- Zvládá 32bitové adresy
- DMA řadič napojený na PCI a lokální paměť
  - Na adrese 0x40000
  - Velikost 4096 B
- Generuje přerušení 8 (`0x100`) na kartě
  - Po dokončení přenosu
  - (Jen je-li vyžádáno před přenosem)

Úkol: doplňte předešlou alokaci (o `set_dma_mask` a `set_master`)

EDU DMA registry

Offset	Len	R/W	Contents	Meaning
0x0080	4B	R/W	src	Source address
0x0084	4B	R/W	dst	Destination address
0x0088	4B	R/W	count	Transfer count
0x008c	4B	R/W	cmd	Command register

Tabulka : Specifikace baru 0 (pokračování z minula)

- `cmd` registr
  - Zapisuje se po nastavení ostatních registrů
  - Zapisuje se jednou s kombinací bitů
  - Bit 0: `RUN` (transfer now)
  - Bit 1: `DIRECTION` (0: from RAM, 1: to RAM)
  - Bit 2: `RQINT` (generate an interrupt after the transaction)

Úkol

Práce s DMA na EDU

1. Naalokovat stránku DMA prostoru
2. 10 B inicializovat textovým řetězcem
3. Přenést 10 B do EDU na adresu 0x40000
   - Nastavit všechny 4 registry
   - Bez přerušení (bez `RQINT`)
   - Počkat na nulovou hodnotu bitu `RUN` (na dokončení přenosu)
4. Přenést 10 B z EDU na DMA stránku + 10
   - Vznikne za sebou 2$\times$ stejný řetězec

Úkol

DMA s přerušením (domácí)

1. Rozšířit předchozí o přerušení
   - Přidat ještě jeden přenos z EDU na DMA + 20
   - Přenos bude s `RQINT`
2. Obsloužit přerušení
   - ACK přerušení
   - Iniciovat tasklet
3. V taskletu vypsat obsah DMA paměti + 20
4. DMA stránku vystavit přes `mmap`

POZOR na prezenci a správnou posloupnost zabíjení taskletu, zákazu přerušení, dealokace atd.