

# PB173 – Ovladače jádra – Linux

## X. DMA

Jiri Slaby

Fakulta informatiky  
Masarykova univerzita

3. 12. 2015

## LDD3 kap. 15 (zastaralá)

### Minule

- Mapování paměti jádra (mmap)

- 1 Přímý přístup do paměti (DMA)

# Sekce 1

## Přímý přístup do paměti (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 odesílání příjmu

- 1 Alokace (speciální) paměti
- 2 Vyplnění daty (paket, zvuk, data na disk, ...)
- 3 Předání ukazatele do zařízení
- 4 Odstartování přenosu v zařízení
- 5 Přerušování či jiná signalizace konce přenosu od zařízení
- 6 Práce s příchozími daty (paket, data z disku, ...)
- 7 Uvolnění (speciální) paměti

### 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` – DMA 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 dma;
    char *virt;

    virt = dma_alloc_coherent(dev, 100, &dma, GFP_KERNEL);
    if (!virt)
        return -ENOMEM;
    memset(virt, 0, 100);

    my_HW_set_addr(dev, dma);
    my_HW_start_transfer(dev);

    while (my_HW_working(dev)) /* polling */
        msleep(100);

    pr_info("%s\n", virt);
    dma_free_coherent(dev, 100, virt, dma);
    return 0;
}
```

## Alokace DMA paměti

- 1 Otevřete si ovladač EDU z předminula
- 2 V `probe` alokujte 1 stránku DMA paměti
- 3 Použijte PCI zařízení jako první parametr alokace
- 4 Vypište si virtuální, fyzickou a DMA adresu
- 5 V `remove` ji uvolněte



## Navíc

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

## Příklad z probe

```
... /* pci_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);
```

- Zvládá 28bitové adresy
- DMA řadič napojený na PCI sběrnici a lokální paměť v EDU
  - 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 nastavení masek na správnou hodnotu a zapnutí spravování sběrnice

## Specifikace baru 0 (pokračování z předminula)

Offset	Len	R/W	Contents	Meaning
0x0080	4B	R/W	src	Source address
0x0088	4B	R/W	dst	Destination address
0x0090	4B	R/W	count	Transfer count
0x0098	4B	R/W	cmd	Command register

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

## Práce s DMA na EDU

- 1 10 B DMA paměti inicializujte textovým řetězcem
- 2 Přeneste 10 B do lokální paměti EDU na adresu 0x40000
  - Nastavte všechny 4 registry
  - Bez přerušení (bez RQINT)
  - Počkejte na nulovou hodnotu bitu RUN (na dokončení přenosu)
- 3 Přeneste 10 B z EDU na DMA stránku + 10
  - Vznikne za sebou 2× stejný řetězec
- 4 Vypište DMA stránku + 10

## DMA s přerušením (část domácího)

- 1 Rozšiřte předchozí o přerušení
  - Přidejte ještě jeden přenos z EDU na DMA + 20
  - Přenos bude s RQINT
- 2 Obslužte přerušení
  - ACK přerušení
  - Iniciujte tasklet
- 3 V taskletu vypište obsah DMA paměti + 20
- 4 DMA stránku vystavte přes `mmap`

## POZOR

- Na prezenci a správnou posloupnost zabíjení taskletu, zákazu přerušení, dealokace atd.
- Na použití správné funkce pro převod na `struct page/pfn` pro `mmap`