

PB173 – Ovladače jádra – Linux

VII. Komunikace s HW

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

4. 11. 2014

LDD3 kap. 9 a 12

- 1 I/O
 - I/O – porty
 - I/O – MMIO
- 2 PCI zařízení
- 3 Zobecnění na jiné sběrnice
- 4 EDU karta

Sekce 1

I/O

Na x86: 2 přístupy

- Porty
- Memory Mapped I/O

Porty

- Speciální adresový prostor
 - Závislé na architektuře
 - Speciální instrukce (`in`, `out` na x86)
- Samostatná (malá) sběrnice
 - Na x86: řadič klávesnice, PC spkr, staré časovače a ovladače přerušení, ladicí port (`0x80` \Rightarrow segmentový displej na desce), ...

API

- `linux/io.h`, `linux/ioport.h`
- Vytvoření: `request_region`, `release_region`
- R/W: `inX(port)`, `outX(co, port)`, kde $X \in \{b, w, l\}$
- Vícenásobné R/W: `insX`, `outsX`

Demo: pb173/07

Úkol: Přečíst port `0x80`, zapsat do něj 1B a znovu přečíst

Memory Mapped I/O

- Součástí fyzického adresového prostoru
- Přístup standardním čtením/zápisem
 - Nutnost přemapovat na virtuální adresy

API

- `linux/io.h`, `linux/ioport.h`
- Vytvoření: `request_mem_region`, `release_mem_region`
 - `/proc/iomem`
- Mapování: `virt=ioremap(phys)`, `iounmap(virt)`
 - `/proc/vmallocinfo` (novější jádra)
- R/W: `readX(odkud)`, `writeX(co, kam)`, kde $X \in \{b, w, l, q\}$
- Vícenásobné R/W: `memcpy_fromio`, `memcpy_toio`

Sekce 2

PCI zařízení

- PCI, PCI-X, PCIe
- Hierarchická sběrnice
 - Identifikace doména:bus:slot:funkce
 - Bridge (=routery)
- Konfigurační prostor
 - Automatická konfigurace
 - ID zařízení (vendor, device), I/O prostory, IRQ
 - *Obsah* I/O – specifikace zařízení (výrobce)
 - lspci
- Podrobnosti v PCI specifikaci

- `linux/pci.h`, `Documentation/pci/*`
- `struct pci_dev`, `struct pci_bus`
- `pci_{set,get}_drvdata`
 - Uloží/načte programátorova data do/z `struct pci_dev`
- Referenční počet
 - Zvýšit: `pci_dev_get`
 - Snížit: `pci_dev_put`
 - PCI zařízení může být (fyzicky) odebráno ze systému
 - Ale `struct pci_dev` zůstává, pokud reference > 0
- Identifikace (ID) zařízení
 - `pci_dev->vendor`, `pci_dev->device`, ...
 - `PCI_ANY_ID` značí jakékoliv ID (při hledání)

Hledání zařízení

Iterátory (starší)

- Dovoluje přístup k zařízení více ovladači
- Nepodporuje hotplug
- `pci_get_device (vendor, device)`

Iterace přes zařízení

```
struct pci_dev *pdev = NULL;

while ((pdev = pci_get_device(VENDOR, DEVICE, pdev))) {
    pr_info ("%2.x:%.2x:%.2x\n",
            pdev->bus->number,
            PCI_SLOT(pdev->devfn),
            PCI_FUNC(pdev->devfn));
}
```

Úkol: vypsát všechna zařízení v systému (jejich *vendor+device* ID)

Hledání zařízení

Událostmi

- Registrace háčků a seznamu chtěných zařízení
 - Seznam: `struct pci_device_id` (vendor, device, atd.)
 - Háčky: `struct pci_driver` (probe, remove, suspend, atd.)
- `pci_register_driver`, `pci_unregister_driver`

Události a PCI zařízení

```
static struct pci_device_id my_table[] = {
    { PCI_DEVICE(VENDOR1, DEVICE1) }, { PCI_DEVICE(VENDOR1, DEVICE2) },
    { PCI_DEVICE(0x8086, PCI_ANY_ID), .driver_data = 1 },
    { 0, }
};
MODULE_DEVICE_TABLE(pci, my_table);
static struct pci_driver my_pci_driver = {
    .name = "my_driver", .id_table = my_table, .probe = my_probe,
};
int my_probe(struct pci_dev *pdev, const struct pci_device_id *id) {
    pr_info ("%2.x %lu\n", pdev->bus->number, id->driver_data);
    return 0;
}
```

Navázání PCI zařízení

- 1 `linux/pci.h`
- 2 Definice `struct pci_device_id`
 - `lspci -mn` a najít EDU a jeho vendor+device ID
- 3 Definice `struct pci_driver`
 - `probe`, `remove`, `name`, `id_table`
- 4 Definice háčeků (viz definici `struct pci_driver`)
 - `int (*probe)(struct pci_dev *, const struct pci_device_id *)`
 - `void (*remove)(struct pci_dev *)`
- 5 V `probe` a `remove` vypsát
 - `pdev->bus->number`
 - `PCI_SLOT(pdev->devfn)`
 - `PCI_FUNC(pdev->devfn)`
- 6 Zavolat `pci_register_driver` a `pci_unregister_driver`
- 7 Vložit a odebrat modul (a zkontrolovat `dmesg`)

Probe

- Inicializace PCI zařízení
 - `pci_enable_device`
 - Do té doby nelze některé vlastnosti pdev používat (`irq`)
- Rezervace a mapování I/O (první část cvičení)
 - Vytvoření: `pci_request_region`, `pci_request_regions`
 - Mapování: `pci_ioremap_bar` – alias pro `ioremap(pci_resource_start(), pci_resource_len())`
- Nastavení/detekce zařízení

Remove

- Opak Probe
- `iounmap`, `pci_release_region(s)`, `pci_disable_device`

Sekce 3

Zobecnění na jiné sběrnice

Většina ostatních sběrnic funguje stejně

- USB, I²C, HID, IEEE1394, ACPI, INPUT, EISA, ...
- Nějaké probe/remove
- Seznam ID
- Registrace „ovladače“

Sekce 4

EDU karta

Specifikace baru 0 karty EDU

Specifikace baru 0 karty EDU

Offset	Len	R/W	Meaning
0x0000	4B	R	ID & Revision
0x0004	4B	R/W	Inverted value
0x0008	4B	R/W	Factorial
0x0020	4B	R	Status

- ID & Revision: 0xRRrrr00edu, RR – major, rr – minor
- Inverted value: zapsané číslo se obrátí (operátor ~)
- Faktoriál: vypočte se faktoriál (je nutné počkat na status bit)
- Status: 0. bit – probíhá výpočet faktoriálu

Vyčtení identifikace karty EDU a práce s ní

- 1 Povolení zařízení (`pci_enable_device`)
- 2 Rezervace baru 0 (`pci_request_region`)
- 3 Vypsát fyzickou adresu baru 0 a porovnat s `lspci`
- 4 Přemapovat bar 0 (`ioremap`)
- 5 přečíst a rozkódovat identifikaci a revizi (`readX`)
- 6 Ověřit živost karty pomocí invertovacího registru
- 7 Uklidit v remove (`unmap`, `release`, `disable`)