

PB173 – Ovladače jádra – Linux V.

Jiří Slabý

ITI, Fakulta Informatiky

26. 10. 2010

LDD3 kap. 8 (zastaralá část o bootmem)

Understanding the Linux Virtual Memory Manager (zastaralá)

- Organizace paměti
- Alokace paměti

2 oddělené světy (tentokrát z pohledu HW)

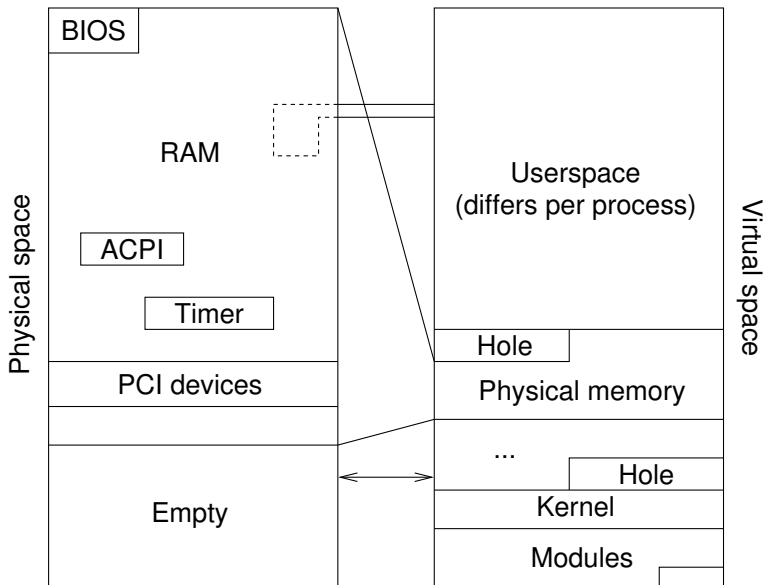
Fyzický prostor

- Rozložení určuje BIOS (popř. ekvivalenty – (U)EFI, Qemu)
- x86: `dmesg | grep BIOS-e820`

Virtuální prostor (paměť)

- Rozložení si určuje OS
 - Různé na každé architektuře
- Dané mapování mezi VP a FP
- Nutná podpora HW (MMU), jinak mapování 1:1

Organizace paměti – graficky



- To, co vidí MMU
- RAM, časovače, řadiče přerušení, ACPI tabulky, zařízení
- Co kde je \Rightarrow otázka na BIOS
- Jak který prostor vypadá \Rightarrow specifikace bridge, ACPI, PCI, ...
- I hierarchický prostor
 - Např. rozsah 100-1ff \rightarrow ChipXY
 - ChipXY směřuje dále:
 - Adresa 100-18f \rightarrow řadič klávesnice
 - Adresa 190-1ff \rightarrow časovač

RAM

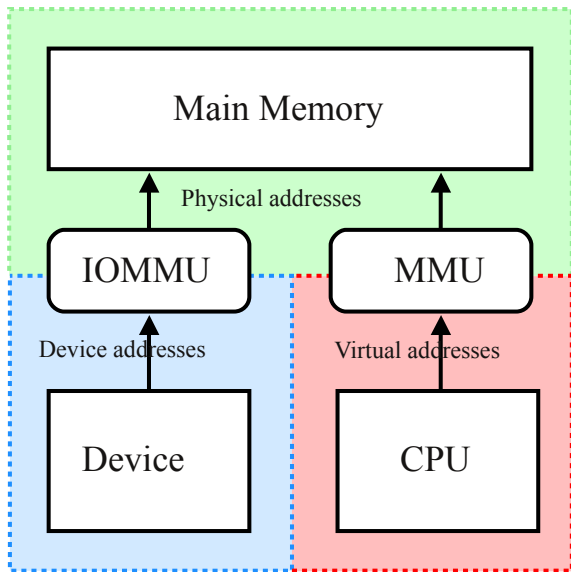
- Reprezentována rozsahy lineárních (fyzických) adres
 - Rozsahy kvůli vloženým prostorům zařízení (e820)
- Cache (některá zařízení také)

Práce s fyzickou adresou – přečtení signatury a délky nějaké ACPI tabulky z fyzické adresy

- 1 Najít fyzickou adresu nějaké ACPI tabulky v `dmesg`
 - Řádky formátu ACPI: `RSDP 000000000000f8550 00014`
 - Jeden si zvolit a zapamatovat adresu (`0x000000000000f8550`)
- 2 Získat virtuální adresu (`u32 *virt = ioremap(phys, 8)`)
- 3 Vypsát signaturu (`text, 4B - virt[0]`)
- 4 Vypsát délku tabulky (`4B - virt[1]`)
- 5 Odmapovat (`iounmap(virt)`)
- 6 Porovnejte délku s výpisem (`0x00014`)
- 7 Můžete si přemapovat a vypsát celou tabulku a porovnat položky se specifikací ACPI

Pozn.: na starších jádrech se výpis v `dmesg` liší

- To, co vidí CPU (a překládá MMU)
 - 1:1 – start systému, noMMU, ...
- OS „rozdělí“ FP na stránky
 - I různě velké
 - Pak mají jiné virtuální adresy
 - Např. přístupy programů po 4K, přístupy jádra po 1G (méně tabulek)
 - Index stránky je tzv. page frame number (pfn)
- OS namapuje stránky do VP
 - A může přidat „virtuální“ stránky např. na disku (swap)
 - Programy vidí více „paměti“
 - Toto mapování čte MMU (ví odkud)
- Každá stránka má svoji `struct page`
 - Informace o stránce (nikdy nevyswapovat, špinavá, počet uživatelů)



Zdroj: wikipedia

- **linux/mm.h, linux/io.h**
- Např. `void *virt=__get_free_page`
- `phys_addr_t phys=virt_to_phys(virt)`
- `void *virt = phys_to_virt(phys)`
- `struct page *page=virt_to_page(virt)`
- `void *virt=page_to_virt(page)`
- `unsigned long pfn=virt_to_pfn(virt)`
- `void *virt=pfn_to_virt(pfn)`
- `unsigned long pfn=phys >> PAGE_SHIFT`
- `phys_addr_t phys=pfn << PAGE_SHIFT`

Práce s virtuální adresou

- 1 `linux/mm.h`
- 2 Naalokujte stránku (`virt=__get_free_page(GFP_KERNEL)`)
- 3 Nakopírujte do ní řetězec
- 4 Zjistěte fyzickou adresu (`phys=virt_to_phys(virt)`)
- 5 Zjistěte adresu struct page (`page=virt_to_page(virt)`)
- 6 Zarezervujte stránku (`SetPageReserved(page)`)
- 7 Přemapujte fyzickou stránku (`map=ioremap(phys, ...)`)
- 8 Vypište `virt`, `phys`, `page`, `map` a `page_to_pfn(page)`
- 9 Vypište obsah `(%s) virt` a `map`
- 10 Změňte obsah `map` a vypište obsah `virt`
- 11 `iounmap`, `ClearPageReserved` a `free_page`
- 12 Zkonzultujte s [Documentation/x86/x86_64/mm.txt](#)

Základní typy alokací

- Malé bloky až souvisle po stránkách (`kmalloc`)
- Souvisle po stránkách (`__get_free_pages`)
- Nesouvisle po stránkách a namapovat (`vmalloc`)

GFP_*

- Parametr pro alokátory
- `GFP_ATOMIC` – nespí (např. uvnitř spinlocků)
- `GFP_TEMPORARY` – uvolním během pár příštích instrukcí
- `GFP_KERNEL` – obyčejné alokace všude jinde
- a další: `GFP_NOFS`, `GFP_NOIO`, ...

- `linux/slab.h`
- Až cca. 8M (různí se pro architektury)
- Využívá 2. typ alokátorů, ale rozděluje stránky na menší kusy
- `kmalloc`, `kzalloc`, `kfree`
- Různé implementace: SLAB, *SLUB*, SLOB, SLQB

- **linux/mm.h**
- Stránka má velikost `PAGE_SIZE` (x86: 4K)
- Velikost alokace se udává řádem, `order` (2^{order} = počet stránek)
 - `get_order`, `MAX_ORDER` (na x86 = 11 = 8M)
- Musí najít souvislý blok volných stránek
 - Může být problém najít větší bloky (řádu ≥ 2) kvůli fragmentaci
- Používá tzv. *buddy* systém
 - Snižuje fragmentaci
 - Podrobnosti viz wiki
- Rychlejší než `kmalloc`
- `__get_free_pages`, `free_pages`
- `__get_free_page`, `free_page`

- **linux/vmalloc.h**
- Podobný `kmalloc`
- Neatomický (nemá `GFP_*`)
- Maximální velikost daná omezením architektury
 - x86: 128M (lze zvýšit parametrem, ne o moc)
 - x86_64: 30T
- Alokuje po stránkách a mapuje je souvisle
 - Tj. používá 2. alokátor
 - Fyzické stránky jsou umístěné různě po RAM
- Dražší
- `vmalloc`, `vfree`
- Odlišná práce s adresami
 - `vmalloc_to_page`, `vmalloc_to_pfn`

- Vše, co může číst uživatel se musí nejprve vymazat
 - `kzalloc`, `get_zeroed_page`, `__GFP_ZERO`, `vzalloc`
 - Nejlépe vymazat všechno, co alokuji a není kritické
- Používat správné `GFP_*`
 - `GFP_ATOMIC` funguje všude, ale ubírá vzácné prostředky

Alokace paměti

- 1 Zjistěte, kolikrát za sebou lze naalokovat paměť:
 - řádu 10 jako `GFP_ATOMIC`
 - řádu 10 jako `GFP_ATOMIC` (znovu po 5 vteřinách)
 - řádu 10 jako `GFP_KERNEL`
 - řádu 10 jako `GFP_KERNEL` po uvolnění předešlé paměti
 - Pozn.: $2^{10} \cdot 4096 = 4 \text{ M}$, jak velké pole na ukazatele?
- 2 Zkuste ve smyčce alokovat paměť s řádem \leq `PAGE_ALLOC_COSTLY_ORDER`, co se stane? Přejde OOM killer. . .

Demo: pb173/05