

PB173 – Ovladače jádra – Linux

I. GIT a jádro

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

16. 9. 2014

- 1 Úvodní informace
- 2 GIT
- 3 Jádro
 - Paměť

Sekce 1

Úvodní informace

- Semestr = 13 týdnů (14. 10. konference)
- Cvičící
 - Vývoj jádra od r. 2005 (NetBSD, Linux)
 - Absolvent FI
- Cíle cvičení
 - Nastítnit trochu jiný model programování
 - Prohloubit znalosti vnitřností OS a HW
- Ukončení: k
 - Splnění *všech* domácích úkolů
 - 10 bodů na úkol, alespoň $\frac{3}{5}$ z celkového počtu
- Vše potřebné ve studijních materiálech v ISu

Na začátku: 10 bodů za každý příklad

- -3 body za každý týden prodlení (termín je vždy do dalšího cvičení)
- -2 body za každé vrácení v případě nějaké nefunkčnosti
- -2 body za bezpečnostní díru (ve slidech značené POZOR)
- -1 bod za kód neodpovídající stylu
- -1 bod za ostatní drobnosti

S čím budeme pracovat?

HW

- Stroje satyr01–10
 - CentOS 6.x
 - Login/heslo: vyvoj/vyvoj
 - Nemají viditelnou IP

S čím budeme pracovat?

SW

- GIT
 - Úvod do GITu dnes
 - Podrobněji: <http://book.git-scm.com/>
- Zdroje jádra
 - Použijeme předinstalované z RPM (`/usr/src/kernels/`)
 - <http://git.kernel.org/> (od 2.6.12)
 - full-history-linux GIT (od 0.01 do 2.6.26)
- QEMU
 - HW k práci s PCI, I/O, na přerušení, DMA...

Sekce 2

GIT

GIT a repozitář PB173

- 1 Stáhněte si repozitář na aisu (doporučeno, můžete vynechat)
 - `git clone git://github.com/jirislaby/pb173`
- 2 Vytvořte si lokální klon
 - `git clone aisa:pb173`
- 3 Prozkoumejte strukturu
 - Příklady ze cvičení
 - Adresář pro domácí úkoly

GIT a úpravy souborů

- 1 Změňte cokoliv v souboru `sandbox/hello`
- 2 Zkontrolujte změny (`git diff --color`)
- 3 Uložte do lokálního repozitáře (`git commit -a`)
 - Git může chtít nastavit jméno a e-mail (instrukce jsou na stdout)
 - Formát logu (*vzor*: `git.kernel.org`)
 - Shrnutí na řádek
 - <Volný řádek>
 - Odůvodnění (delší popis)
 - <Volný řádek>
 - Podpisy a CC
- 4 Smažte `sandbox/hello` (`git rm`)
- 5 `git commit -a`

- Každý commit je pojmenován SHA hashem
 - Např. d3323c1503d54d83b0eae6c7927dede2d2973059, lze i zkráceně d3323c1503
- HEAD je alias pro poslední
- Lze odkazovat předchůdce pomocí ~
 - Např. HEAD~1, d3323c1503~5
- Lze vytvořit jmenný alias, tzv. tag

GIT a commity

- 1 Zkontrolujte log, zda obsahuje 2 změny (`git log --color`)
- 2 Podívejte se na poslední 2 změny (`git show --color HEAD`, resp. `HEAD~1`)
- 3 Vygenerujte záplaty 2 posledních commitů (`git format-patch -2`)
- 4 Proveďte push (`git push`, pokud máte klon na aise)

**Odevzdávání domácích úkolů odesláním záplaty na e-mail.
Posílejte jen plain-text přílohy!**

Záplaty v linuxovém jádře

- 1 Poslat odpovídajícímu správci (`scripts/get_maintainer.pl`)
- 2 Poslat na ML „pull request” a vystavit celý GIT strom
 - Popis v `Documentation/SubmittingPatches`

Sekce 3

Jádro

Hlavní rozdíly

- Žádné *libc* (`printf`, `strlen`, `malloc`, ...), ani ostatní (`pthread`)
- Ne/oddělený *paměťový prostor*
 - Chybový kód může přepisovat cokoliv (i FW)
- *Počáteční funkce* (`main`)
 - `module_init` (=main), `module_exit` (=on_exit)
- Pád jádra \Rightarrow pád všeho

- Výhradně *GNU C* (x86 už jen GCC \geq 4.x)
- Pevně daný *CodingStyle*
 - Kontrola: `scripts/checkpatch.pl` (není 100%)
- Vše slinkováno do jednoho
 - `vmlinux` \rightarrow (b)zImage
 - Ale moduly (standardní ELF: *.ko)

Průzkum modulu (.ko objektu)

- 1 Zvolte si jeden modul v systému (`lsmod`)
- 2 Zavolejte na něm `modinfo`
- 3 Proveďte `objdump -d` sekce `.modinfo` sekce
- 4 Porovnejte oba výstupy

- Linux Device Drivers, 3. edice
- `Documentation/*`
- Generovaná tamtéž (podobná DocBook)
- Kód (<http://lxr.free-electrons.com/ident>)

Demo: pomocí lxr najděte v jádře obecnou (v `lib/`) a optimalizovanou (v `arch/`) implementaci `strlen`

Hello World

- 1 Spusťte virtuální stroj
- 2 Prozkoumejte adresář 01 z pb173 git repozitáře
 - Makefile, pb173.c
- 3 Do `init` funkce doplňte výpis „Hello World”
 - `printk(KERN_INFO "... \n");` (*ne, není tam čárka*)
 - Nebo zkráceně: `pr_info(... \n);`
 - Objeví se v `/proc/kmsg` a na konzoli
- 4 Přeložte a vložte do systému
 - `make, insmod <modul.ko>`
- 5 Zkontrolujte výstup `dmesg`
- 6 Můžete udělat `commit` a `push` do svého repozitáře

- Omezený zásobník (4–8 K)
 - Žádná nebo malá rekurze
 - Jen pro malá data (`int`, malé `struct`, krátká pole, ...)
- Pracuje se se stránkami (na x86 4 K–1 G)
 - Omezená velikost alokace (fragmentace)
 - Podrobnosti v dalších cvičeních

Malé alokace (sta bajtů, maximálně až cca. 4 M)

- Horní mez závislá na architektuře
- `kmalloc`, `kfree` (`linux/slab.h`)

Alokace mají (většinou) GFP parametr. Ten určuje, co si alokátor může dovolit (spát, swapovat, použít HIGHMEM, ...). Prozatím nám stačí `GFP_KERNEL`.

```
void *mem = kmalloc(100, GFP_KERNEL);
if (mem) {
    ...
}
kfree(mem);
```

- 1 V `exit` funkci naalokujte 1000 bytů (`kmalloc`)
- 2 Udělejte do nich strcpy „Bye”
- 3 Vypište paměť jako řetězec (`printf` a `%s`)
- 4 Uvolněte paměť
- 5 Vložte a odeberte modul ze systému (`insmod` a `rmmmod`)
- 6 `dmesg`
- 7 Můžete provést `commit` a `push`