PB173 – Binární programování Linux

I. GIT a binární programování

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

20. 9. 2016

Obsah cvičení

Sekce 1

Úvodní informace

Úvodní informace

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

Bodování úkolů

10 bodů za každý příklad

- -3 body za každý týden prodlení (termín je vždy do dalšího cvičení)
- -3 body za bezpečnostní díru (ve slidech značené *POZOR*)
- -2 body za nesplnění jedné části zadání
- -1 bod za kód neodpovídající stylu
- -1 bod za ostatní drobnosti

S čím budeme pracovat?

Hardware

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

S čím budeme pracovat?

Software

- GIT
  - Úvod do GITu dnes
  - Podrobněji: `http://book.git-scm.com/`
- Zdroje jádra
  - GIT: `http://git.kernel.org`
  - LXR: `http://lxr.free-electrons.com/ident`
- Zdroje glibc
  - GIT: `http://repo.or.cz/w/glibc.git`
- ANTLR, BINUTILS, COREUTILS, ELFUTILS, GDB, IPC, LIBDWARF, LIBPCIACCESS, RPC, ...

Sekce 2

GIT

Úkol

GIT a repozitář PB173

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

Úkol

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`

GIT a commity

- Každý commit je pojmenován SHA hashem
  - Např. `d3323c1503d54d83b0eae6c7927dede2d2973059`, lze i zkráceně `d3323c1503`
- `HEAD` (`@`) je alias pro horní/poslední commit
- Lze odkazovat předchůdce pomocí ~
  - Např. `HEAD~~~`, `HEAD~1`, `d3323c1503~5`
- Lze vytvořit jmenný alias, tzv. tag
- Více: `man gitrevisions`

Úkol

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 ze 2 posledních commitů (`git format-patch -2`)
4. Proveďte push (`git push`, jen pokud máte klon na aise)

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

Sekce 3

Binární programování

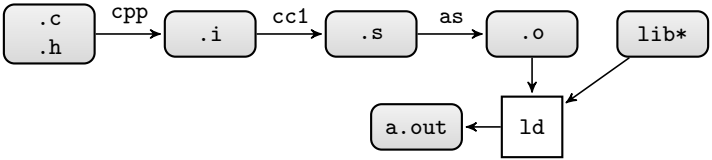
Binární programování

- Popíšeme si fáze překladače
- Nastíníme tvorbu překladače
- Uvedeme si souborové formáty a budeme s nimi pracovat
- C bez *libc* (`printf`, `strlen`, `malloc`, ...) a ostatních
- Probereme si malé základy assembleru
- Ladění z jiného úhlu pohledu
- Komunikace s HW a mezi procesy

C budeme psát ve stylu jádra (*CodingStyle*)

- Kontrola: v jádře `scripts/checkpatch.pl` (není 100%)

Překladač



GCC

- GNU C
- Pro překlad C, assembleru a hlavičkových souborů
- Pro linkování
- Důležité volby
  - `-E` – spusť jen preprocesor a skonči
  - `-S` – generuj assembler
  - `-c` – generuj objektový soubor
  - `-x` – definujeme jazyk souboru, když nelze určit dle přípony (`-x c soubor_v_C -x assembler soubor_v_asm`)
  - `-O` – optimalizace (`-O0`, `-O2`, ...)
  - `-W` – varování (`-Wall`, ...)
  - `-g` – ladicí informace (`-g0`, `-ggdb`, ...)
- Dokumentace
  - Offline: INFO/PINFO
  - Na webu: *Using the GNU Compiler Collection*

Úkol

Volání GCC

1. Pomocí roury přeložte `main`, který něco vypíše, do objektu
   - `echo -e '#include <stdio.h>\n int main() { puts("Hello"); return 0; }' | ...`
2. Spusťte všechny tři fáze (`-E`, `-S` a `-c`) postupně jako samostatné příkazy propojené rourami
   - `echo ... | gcc -E ...`
   - `echo ... | gcc -E ... | gcc -S ...`
   - `echo ... | gcc -E ... | gcc -S ... | gcc -c ...`
3. Pozorujte `-E` a `-S` výstupy
4. Zapněte optimalizace `-O2` a pozorujte oba výstupy znovu
   - Mění optimalizace oba výstupy?

OBJDUMP/OBJCOPY

- Slouží k práci s objekty
- OBJDUMP vypisuje informace
  - `-D` disassembler vstupu
  - `-h` výpis sekcí vstupu
  - `-b` formát vstupu
  - `-m` stroj vstupu
- OBJCOPY transformuje
  - `-I` formát vstupu
  - `-O` formát výstupu
- Společný důležitý argument
  - `-j` specifikace sekce

Úkol

Práce s objekty

1. Vypište si informace o svém objektu z předchozího příkladu pomocí `objdump`
   - Porovnejte výstupy `objdump -D` a `gcc -S -fverbose-asm`
   - Zavolejte `objdump -h`
2. Extrahujte `.text` sekci pomocí `objcopy`
   - Vložte ji do nového souboru v binární (surové) podobě
   - Tj. použijete `-j .sekce` a `-O binary`
3. Disassemblujte tento binární výstup pomocí `objdump`
   - Musíte specifikovat formát (`-b binary`) a stroj (`-m i386:x86-64`)

Úkol

Práce s objekty II. (domácí)

1. Prozkoumejte adresář `01` z pb173 git repozitáře
2. Je tam soubor `x.bin` podobně vytvořený jako váš
   - Formát `binary`, pro `i386:x86-64`
3. Je v něm funkce, nápověda:
   - Akceptuje dva `int` parametry a vrací `int`
   - Je „const” – tj. kromě parametrů nebere v úvahu nic
4. Zavolejte ji
5. Zjistěte, co dělá
   - Vyzkoušejte několik kombinací parametrů