PB173 – Binární programování Linux

IX. Gdb a ladění výkonnosti

Jiri Slaby

ITI, Fakulta informatiky

19. 11. 2013

Ladění

1. Funkčnosti (minule)
   - Dnes ještě více `gdb`
2. Výkonnosti/velikosti (dnes)
   - Optimalizace kódu
   - Odstranění nepoužitého kódu

Část I

gdb

Debugger `gdb` (opakování)

- Pracuje s ELFem a DWARFem
- Používá systémové volání `ptrace`
  - Sleduje proces
  - Upravuje vykonávání
  - Čte a mění paměť
  - ...
- Spuštění: `gdb --args binarka --volby`
- Nápověda: `help`
- Manuál: GDB User Manual

gcc a gdb

- `gcc` generuje různé úrovně ladicích informací
  - `-g1` (méně informací)
  - `-g2` alias `-g`
  - `-g3` (navíc informace o makrech)
  - `-ggdb` (i v kombinaci s `-g<cislo>`)
    - `gcc` použije i `gdb` rozšíření, pokud možno
- Ladí se obtížněji s *optimalizacemi*
  - Proměnné zmizely, konstanty se propagovaly, . . .
  - Bez optimalizací je pomalý kód
  - V `gcc` 4.8 se objevilo `-Og`
    - Nezapíná samotnou generaci ladicích informací!
    - Používá se v kombinaci s `-g...`

Demo: makra a `gdb`

Výpisy

- Dump (`x`) a print (`p`)
- Lze použít s formátováním (`x/f` a `p/f`, kde `f` je formát)
- Formát jsou tři hodnoty
  1. Počet vypisovaných hodnot
  2. Formát vypisovaných hodnot
     - `x`: hexa (podobně jako `printf`)
     - `d`: decimálně
     - `u`: decimálně bez znaménka
     - `s`: řetězec
     - `a`: adresa
     - `i`: instrukce
     - Ostatní: viz `help x`
  3. Velikost skupiny
     - `b`: 1 bajt
     - `h`: 2 bajty
     - `w`: 4 bajty
     - `g`: 8 bajtů

Úkol

Výpisy

1. Otevřete si nějaký program v `gdb`
2. Spusťte (`start`)
3. Vypište
   - Adresu aktuální instrukce (`p/a $pc`)
   - 20 instrukcí od aktuální (`x/20i $pc`)
   - 32 hexa byte hodnot na zásobníku (`$sp`)
   - 8 hexa giant hodnot na zásobníku

Automatické výpisy

- Přidání: `display`
- Informace: `info display`
- Lze nastavit několik automatických výpisů po každém příkazu
- Parametry stejné jako `p` nebo `x`

Vlákna a gdb

- Informace: `info threads`
- Přepnutí: `thread <cislo>`

Demo: `pb173-bin/09/thread.c`

Další informace

- Spousta dalších informací v `info`
- `functions`, `types`, `variables`, `locals`
- `sources`, `line`, `macros`, `scopes`
- `symbol`
- Další viz `help info`

Demo

Změna běhu

- `set`, `set variable`
- Registry procesoru: `$rax`, `$rbx`, ...
- Proměnné `gdb`: `$pc`, `$sp`, ...
- Vnitřní proměnné: `$ostatni`
  - Pro vlastní použití
- Paměť: `{typ}adresa`

Část II

Ladění výkonnosti

Profilování

Co zabírá nejvíce času při vykonávání?

1. `gprof`
   - Při překladu: `gcc -pg ...`
   - Potom se program spustí a data se uloží na disk
   - `gprof` data načte a zobrazí
   - Demo: dump
2. `perf`
   - Používá čítače v procesoru (nutná podpora CPU)
   - Zahrnuje chování jádra
   - Umí pracovat s `gcc -pg` binárkami
   - Spuštění: `perf record -g -- binarka --volby`
   - Dokumentace: `http://perf.wiki.kernel.org`

Úkol

Profilování

1. Projděte si `pb173-bin/09/prof.c`
2. Spusťte `prof_pg`
3. Prostudujte výstup `gprof -b prof_pg`

4. Nahrajte si výstup
   - `perf record -g prof`
5. Zobrazte si výstup
   - `perf report -g`
6. Zopakujte `perf` pro `prof_pg`

Pokrytí kódu

Která část kódu se vůbec nevykonává?

- `gcov`
- Při překladu: `gcc --coverage ...`
- Potom se program spustí a data se uloží na disk
- `gcov` data načte a zapíše `cov.c.gcov`

Úkol

Pokrytí kódu

1. Projděte si `pb173-bin/09/cov.c`
2. Spusťte `cov`
3. Prostudujte výstup `gcov cov`
4. Prostudujte `cov.c.gcov`