

# PB173 – Binární programování Linux

## IX. Gdb a ladění výkonnosti

Jiri Slaby

Fakulta informatiky  
Masarykova univerzita

18. 11. 2014

## Minule

- Ladění funkčnosti

1 Ještě více gdb

- 2 Ladění výkonnosti
- Profilování
  - Pokrytí kódu

# Sekce 1

Ještě více 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 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
- S *optimalizacemi* se ladí se obtížněji
  - Proměnné zmizely, konstanty se propagovaly, ...
  - Ale bez optimalizací je pomalý kód
    - Někdy i nepřeložitelný
  - V gcc 4.8 se objevilo -Og
    - Zapíná jen optimalizace, které neztěžují ladění
    - Nezapíná samotnou generaci ladicích informací!
    - Používá se v kombinaci s -g...

**Demo:** makra, gdb a pb173-bin/09/mac.c

- 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

## 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

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

**Úkol:** vypište si aktuální instrukci po každém příkazu



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

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

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

## Demo

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

## Sekce 2

# Ladění výkonnosti

## 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>

## 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`
  - `perf record -g prof`
- 5 Zobrazte si výstup
  - `perf report -g`
- 6 Zopakujte `perf` pro `prof_pg`

## 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`

## Pokrytí kódu

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