PB173 – Binární programování Linux

VIII. Ladění

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

11. 11. 2014

Techniky ladění

Z pohledu vykonávání

1. Staticky
   - Spouštím analyzátor a ten hledá nesrovnalosti
   - Analýza ukazatelů, hodnot, velikosti zásobníku, ...
   - Není cílem cvičení, ale *formela*
2. Dynamicky
   - Spouštím na CPU a očekávám výsledek
   - Testování
   - Vyzkoušíme si dnes a příště

Techniky ladění

Z pohledu cíle

1. Ladění funkčnosti (dnes)
   - `gdb`
   - Obrazy paměti
   - Tracery
   - `valgrind`

Příště

- Pokračování s `gdb`
- Ladění výkonnosti/velikosti
  - Optimalizace kódu
  - Odstranění nepoužitého kódu

Sekce 1

Ladění funkčnosti (dnes)

Ladění funkčnosti

- Výpisy
  - `printf` a příbuzné
- Monitor chování
  - `gdb`, `ltrace`, `strace`, `valgrind`, ...
- Post-mortem
  - `core` soubor (obraz paměti)

Debugger `gdb`

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

Příkazy gdb

Operace s během

- Spustit: `run`
- Spustit po `main`: `start`
- Přerušit: signálem (např. Ctrl-c)
- Pokračovat po návrat z funkce: `finish`
- Pokračovat: `continue`
- Krokovat s vnořením: `step` (po řádcích), `stepi` (po instrukcích)
- Krokovat bez vnoření: `next` (po řádcích), `nexti` (po instrukcích)
- Konec: `quit`
- Opakování předchozího příkazu: ENTER

Úkol: spusťte si `yes` v `gdb` a přerušte. Vyzkoušejte si `finish` a `continue`. Zkuste krokovat po instrukcích hlavní cyklus `yes`. Poté `quit`.

Příkazy gdb

Výpisy

- Zásobník volání: `where`
  - Pohyb po rámcích zásobníku: `up`, `down`
- Programu: `list` (jen s ladicími informacemi)
- Assembleru: `disassemble`
- Paměti: `print`
- HexDump: `x`
- Ostatní: `info registers`, `info break`, `...`

Většina příkazů akceptuje parametry – výpis lokací, proměnných, registrů atd.

Úkol

Výpisy v `gdb`

1. Přeložte si `pb173-bin/08/`
2. Spusťte `debug` (mělo by dojít k pádu)
3. Otevřete `debug` v `gdb`
4. Spusťte (`run`)
5. Prohlédněte si, kde došlo k pádu (vypište):
   - Zásobník volání (`where`)
   - Místo pádu (`list` s parametrem `soubor:radek`)
   - Disassembly (`disassemble` s parametrem `adresa`)
   - Proměnnou `b` (`print b`)
   - Proměnnou `argc` v `main` (`up` a `print argc`)

Příkazy gdb

Breakpointy

- Příkaz: `break`
- Parametr
  - Adresa
  - Funkce
  - soubor:radek
  - ...

Úkol

1. Přidejte breakpoint na začátek `main` a spusťte (`run`)
2. Krokujte až k pádu (`next`)
3. Zopakujte pro `step` a také `stepi`

Obrazy paměti a `gdb`

- Jádro při pádu zapíše obraz paměti („coredump”)
  - Nutno zvýšit limit velikosti (bývá 0): `ulimit -c`
  - Cíl, kam se ukládají: `sysctl kernel.core_pattern`
- Potom se dá analyzovat v `gdb`
  - `gdb --core=core_soubor --args binarka --volby_binarky`

Úkol

Práce s obrazem paměti

1. Nastavte limit na obrazy na `unlimited` (`ulimit -c`)
2. Podívejte se, kam se obraz uloží (`sysctl kernel.core_pattern`)
3. Spusťte program z předchozího příkladu
4. Otevřete v `gdb` s obrazem paměti (`gdb --core=<core> ./debug`)
5. Vypište si podobné informace jako předtím
   - `where`
   - `print b`
   - A podobně

Tracery

- Sledují události a vypisují je
- `ltrace`: sleduje knihovní volání
- `strace`: sleduje systémová volání
- Používají systémové volání `ptrace`
  - `ltrace` si nastaví breakpointy na všechny externí symboly z ELFu
  - `strace` jednoduše použije `PTRACE_SYSCALL`
- Spuštění: `{s,l}trace binarka --volby`
- Další parametry
  - `-e`: filtr (`-e write`)
  - `-f`: sleduje i potomky
  - `-o`: výstup do souboru

Úkol

Práce s tracery

1. Projděte si `pb173-bin/08/tracer.c`
2. Spusťte nejdříve s `ltrace`
   - Kolik je volání `fwrite` do `libc`?
3. Spusťte s `strace`
   - Kolik je volání `write` do jádra?
   - $\Rightarrow$ `libc` bufferuje
4. Pomocí `ltrace` zjistěte, jak velký buffer `libc` používá
   - V cyklu vypisujte znak pomocí `fwrite`
   - Použijte volbu `-S`
5. Pomocí `strace` ověřte totéž
   - Pozorujte třetí parametr `write`

valgrind

- Sleduje (zejména) operace s pamětí (*Memcheck*)
  - Úniky paměti
  - Neinicializované a nezarovnané přístupy
  - Použití paměti po `free`
  - ...
- Několik dalších modulů
  - Callgrind: profiluje cache CPU
  - Massif: profiluje použití haldy
  - Další pro problémy se zámky atd.
- Spuštění:

  ```
  valgrind --volby_valgrindu binarka --volby_binarky
  ```
- Další parametry
  - `--tool`: výběr nástroje shora
  - `--leak-check`: sledovat úniky paměti

Úkol

Práce s nástrojem `valgrind`

1. Projděte si `pb173-bin/08/memcheck.c`
2. Spusťte v nástroji `valgrind`
3. Řiďte se pokyny na konci výpisu
   - Vypište si všechny podrobnosti
   - Přidáváním navržených parametrů („Rerun with”)