PB173 – Binární programování Linux

IX. Ladění

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

26. 11. 2016

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

1. Ladění funkčnosti (dnes)
   - Tracery
   - `ptrace`
   - `valgrind`

Příště

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

Sekce 1

Ladění funkčnosti (dnes)

Ladění funkčnosti

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

Tracery

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

Úkol

Práce s tracery

1. Projděte si `pb173-bin/09/tracer.c`
2. Přeložte
3. Spusťte nejdříve s `ltrace`
   - Kolik je volání `fwrite` do `libc`?
4. Spusťte s `strace`
   - Kolik je volání `write` do jádra?
   - $\Rightarrow$ `libc` bufferuje
5. Pomocí `ltrace` zjistěte, jak velký buffer `libc` používá
   - Přidejte cyklus, který vypisujte znak pomocí `fwrite`
   - Použijte parametr `-S`
6. Pomocí `strace` ověřte totéž
   - Pozorujte třetí parametr `write`

ptrace

- `sys/ptrace.h`
- Systémové volání ke sledování procesů
  - Naváže se na proces
  - Ovládá ho
- `long ptrace(enum __ptrace_request r, pid_t pid, void *addr, void *data)`
  - `r` – co chceme
  - `p` – na kom to chceme provést
  - `addr` a `data` – význam závisí na hodnotě `request`

Dvě možnosti

1. Svého potomka
   - Potomek: `ptrace(PTRACE_TRACEME)` a `raise(SIGSTOP)`
   - Rodič: `waitpid` a `ptrace(neco)`, např. `PTRACE_CONT`
   - `waitpid` počká na `raise` potomka, který už je p-trasovatelný
2. Cizí proces
   - Trasovač: `ptrace(PTRACE_ATTACH, pid)` a `waitpid`
   - Nastaví `pid` p-trasovatelný, pošle `SIGSTOP`, `waitpid` na signál počká

Odpojení *zastaveného* trasovaného

`ptrace(PTRACE_DETACH, pid)`

Úkol

Trasování potomka

1. Proveďte `fork`
2. Potomek
   - Nastavte jej p-trasovatelného
   - Něco vypište
   - Pošlete mu signál
   - Něco vypište
3. Rodič
   - Počkejte na zastaveného potomka
   - Čekejte na standardní vstup
   - Něco vypište
   - Odpojte potomka
   - Něco vypište
4. Vyzkoušejte

- `ptrace(PTRACE_CONT, pid, NULL, NULL)` – pokračuj ve vykonávání
- Přerušení klasickým `kill(pid, SIGSTOP)`
- `ptrace(PTRACE_SYSCALL, pid, NULL, NULL)` – do příštího systémového volání
- `ptrace(PTRACE_SINGLESTEP, pid, NULL, NULL)` – jedna instrukce
- Vždy je třeba počkat na signál od potomka

Úkol

Krokování potomka

1. Připište do rodiče cyklus
2. V cyklu čekejte na standardní vstup
3. A poté vždy krokujte potomka po instrukci
   - `PTRACE_SINGLESTEP`
4. Nezapomeňte ošetřit stav, kdy potomek skončil
5. Vyzkoušejte

- `ptrace(PTRACE_PEEKTEXT)` – přečti kód
- `ptrace(PTRACE_PEEKDATA)` – přečti data
- `ptrace(PTRACE_PEEKUSER)` – přečti registr, ...
- `ptrace(PTRACE_GETREGS)` – přečti registry
  - `struct user_regs_struct`
- `ptrace(PTRACE_GETFPREGS)` – přečti plovoucí desetinné registry
  - `struct user_fpregs_struct`
- `SET*` a `POKE*` varianty

Úkol

Manipulace s procesem

1. Dopisujte navíc do cyklu v rodiči
2. Vypište adresu aktuálně zastavené instrukce
   - `PTRACE_GETREGS` a `eip`/`rip` v `struct user_regs_struct` → `data`
3. První byte instrukce
   - `PTRACE_PEEKTEXT` a `eip`/`rip` shora → `addr`
4. Vyzkoušejte
5. Můžete ověřit pomocí `objdump -d` binárky

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
  - `-v`: podrobnější informace

Úkol

Práce s nástrojem VALGRIND

1. Projděte si `pb173-bin/09/memcheck.c`
2. Spusťte `memcheck` v nástroji VALGRIND
3. Řiďte se pokyny na konci výpisu
   - Vypište si všechny podrobnosti
   - Přidáváním navržených parametrů („Rerun with”)
4. Opravte, aby si VALGRIND nestěžoval