



Nízkoúrovňové programování

Nástroje pro ladění

Petr Krajča



Katedra informatiky
Univerzita Palackého v Olomouci



- debugger (gdb)
- kontrola práce s pamětí (valgrind)
- profiler (gprof)



- konzolová aplikace
- „netradiční“ ovládání
- nutné program přeložit s informacemi pro debugger, parametr gcc -g

spuštění

- `gdb program //` spustí debugger pro program
- `run, r //` spustí ladění programu
- `run arg1 arg2 ... //` spustí ladění programu se zadanými argumenty

ukončení

- `quit, q //` ukončí debugger
- `kill, k //` ukončí laděný program



procházení obsahu souborů

- `list, l //` zobrazí aktuální místo v kódu
- `list file:n //` zobrazí obsah zadaného souboru od řádku `n` a nastaví jej jako aktuální

obsah proměnných

- `print, p //` zobrazí obsah zadané proměnné
- `print/x, p/x //` zobrazí obsah zadané proměnné v hexadecimální soustavě
- `info locals //` zobrazí obsah lokálních proměnných



breakpointy

- `break, b //` nastaví breakpoint na dané místo, např.
 - `b get_foo //` při vstupu do funkce `get_foo`
 - `b 42 //` na řádku 42 v aktuální souboru
 - `b bar.c:42 //` na řádku 42 v souboru `bar.c`
- `delete, d //` odstraní breakpoint s daným číslem (alternativně příkazy `disable/enable`)

krokování

- `step, s //` další krok (vstupuje do funkcí, *step into*)
- `next, n //` další krok (nevstupuje do funkcí, *step over*)
- `finish, fin //` vyskočení z funkce
- `continue, c //` pokračuje v provádění programu

- `backtrace`, `bt` // zobrazí zásobník volání (call stack)
- `backtrace full`, `bt f` // zobrazí zásobník volání včetně lokálních proměnných
- `frame`, `f` // přepne aktuální volací rámeček (dle čísla z `backtrace`)

analýza spadnuvšího programu

- „Segmentation fault (**core dumped**).“
- soubor `core` v aktuálním adresáři (nebo jinde, viz `systemd`)
- `ulimit -c unlimited`
- `gdb foo core`
- analýza zásobníku volání, paměti, ...

- kontrola práce s pamětí
- interpretuje kód a hlídá přístup do paměti
- výrazně pomalejší než kód prováděný přímo CPU
- hlídá
 - uniklou paměť (memory leak)
 - práci s nealokovanou nebo uvolněnou pamětí
 - práci s neinicializovanou pamětí

příklady užití

```
valgrind ./foo  
valgrind --leak-check=full ./foo  
valgrind --track-origins=yes ./foo
```



- umožňuje identifikovat úzká hrdla v programu (počet volání, čas strávený ve funkci)
- přepínač `gcc -pg`
- spuštěný program generuje soubor `gmon.out`
- prohlížení pomocí nástroje `gprof foo`



- vyberte si vhodnou složitější datovou strukturu (hash tabulka, AVL/23/RB-strom)
- implementujte ji tak, aby podporovala operace vložení páru klíč-hodnota, vyhledání hodnoty podle klíče, odstranění páru dle zadaného klíče
- při programování/ladění použijte nástroje GDB a valgrind
- naplňte strukturu větším množstvím dat a vyzkoušejte profiler