# X36PJC
# Proseminář #2

# Ladění a profilování aplikace

# Osnova cvičení

- Vysvětlení pojmu debugger
- Seznámení s debuggerem dbg
- Správné návyky při psaní kódu
- Ukázka grafického prostředí debuggeru
- Profilování

# Debugger

- Nástroj pro ladění aplikace
- Umožňuje podmíněné zastavení chodu aplikace (breakpoint)
- Umožňuje krokovat běh aplikace
- Umožňuje zjistit aktuální hodnoty proměnných za chodu aplikace
- Grafický nebo textový
- Lze debugovat predevším speciálně přeložené aplikace (debug X release)
- Proč jej používat

# Co je třeba pro debugging

- Správně přeložit aplikaci

  *#>gcc -o app -g main.*
- Spustit aplikaci v debuggeru

  *#>gdb app*
- Orientovat se v prostředí debuggeru

  *(gdb)...*

# GDB

- Standardní C/C++ debugger pro unixové systémy
- Koncipován pro textové rozhraní
- Svou koncepcí umožňuje snadno přidat grafickou nadstavbu (Eclipse, Insight, ...)
- Základní textové rozhraní je řešeno jako terminál, kterému lze zadavát různé příkazy
- Primárně neslouží k ladění problémů s pamětí

# Práce s GDB I.

- Spuštění aplikace

  *(gdb) run/r*
- Spuštění aplikace s parametry

  *(gdb) run/r argument1 argument2 …*
- Zastavení chodu aplikace – breakpoint

  *(gdb) break/b cislo_radku/nazev_funkce*
- Podmíněný breakpoint

  *(gdb) break cislo_radku if podminka*
- Dodatečné přidání podmínky k breakpointu

  *(gdb) condition cislo_breakpointu podminka*

# Práce s GDB II.

- Zobrazení přehledu breakpointů

  *(gdb) info/i breakpoints*
- Smazání break pointu

  *(gdb) delete/d cislo_breakpointu*
- Pokračování v běhu aplikace

  *(gdb) continue/c*
- Pokračování chodu aplikace o jeden příkaz

  *(gdb) step/s [pocet_kroku]*
- Pokračování chodu aplikace o jeden řádek

  *(gdb) next/n [pocet_radku]*

# Práce s GDB III.

- Zjištění hierarchie volaných funkcí

  *(gdb) backtrace/bt*
- Zjištění hodnoty proměnné

  *(gdb) print/p promenna*
- Zjištění hodnoty dynamicky alokované proměnné

  *(gdb) print/p *prommena@pocet_prvku_k_zobrazeni*
- Zobrazení nápovědy

  *(gdb) help/h*
- Ukončení práce s GDB

  *(gdb) quit/q*

# Jak si usnadnit ladění

- V každém řádku kódu jen jeden příkaz
- Nepoužívat volání funkce či metody jako argument
- Nepoužívat kombinace podmíněných výrazů v podmínce
- Složitější výpočty rozdělit na dílčí části

# Profilování

- Jedná se o formu optimalizace
- Opírá se o statistickou analýzu běhu aplikace
- Hledájí se
  - Místa kódu, kde aplikace stráví nejvíce času
  - Místa kódu, které se volají nejčastěji
- Skládá se ze dvou kroků
  - Sběr dat
  - Analýza dat

# Jak získat profilovací data

- Explicitně
  - Do kódu aplikace se začlení speciální funkce, které pomohou sbírat potřebna data
  - Vše závisí na programátorovi samotném
  - Velmi náchylné na lidskou chybu
- Implicitně
  - Využije se implicitní mechanismus překladače
  - Neklade na programátora žádné nároky
  - Analýza je pak velice přesná

# Gcc a gprof

- Překladač gcc nabízí sběr profilovacích dat
- Pro jejich získání je třeba přeložit aplikaci s přepínačem „-pg“
- Po spuštění aplikace se generuje „call graph“ uložený v souboru *gmon.out*
- Výsledná analytická data zobrazíme pomocí programu *gprof* jehož argumentem je příkaz pro spuštění profilované aplikace

# Použité zdroje a užitečné odkazy

- http://sourceware.org/gdb/current/onlinedocs/gdb/
  - Podrobný manual ke gdb
- http://www.cs.cmu.edu/~gilpin/tutorial/
  - Tutorial gdb
- http://www.unknownroad.com/rtfm/gdbtut/
  - Tutorial gdb
- #> *man gdb*
  - Manuálové stránky gdb