

# Ladění chyb

Tématicky zaměřený vývoj aplikací v jazyce C  
skupina Systémové programování – Linux

Jiří Novosad

Fakulta informatiky  
Masarykova univerzita  
novosad@fi.muni.cz

28. září 2014

## C Preprocesor

Inspirováno přednáškou Rudy Čejky, FIT VUT

# Co umí preprocesor

- Vkládání zdrojových souborů
  - Měly by mít syntax podobnou jazyku C

# Co umí preprocesor

- Vkládání zdrojových souborů
  - Měly by mít syntax podobnou jazyku C
- Předdefinovaná makra
  - `__LINE__`, `__DATE__`, `__TIME__`, `__func__`, ...

# Co umí preprocesor

- Vkládání zdrojových souborů
  - Měly by mít syntax podobnou jazyku C
- Předdefinovaná makra
  - `__LINE__`, `__DATE__`, `__TIME__`, `__func__`, ...
- Definice maker
  - `#define MA printf("Dneska je úžasný den!\n")`  
`#define MB(parametr) parametr`  
`#define MC(...) printf(__VA_ARGS__)`

# Co umí preprocesor

- Vkládání zdrojových souborů
  - Měly by mít syntax podobnou jazyku C
- Předdefinovaná makra
  - `__LINE__`, `__DATE__`, `__TIME__`, `__func__`, ...
- Definice maker
  - `#define MA printf("Dneska je úžasný den!\n")`  
`#define MB(parametr) parametr`  
`#define MC(...) printf(__VA_ARGS__)`
- Podmíněný překlad
  - `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`

# Co umí preprocesor

- Vkládání zdrojových souborů
  - Měly by mít syntax podobnou jazyku C
- Předdefinovaná makra
  - `__LINE__`, `__DATE__`, `__TIME__`, `__func__`, ...
- Definice maker
  - `#define MA printf("Dneska je úžasný den!\n")`  
`#define MB(parametr) parametr`  
`#define MC(...) printf(__VA_ARGS__)`
- Podmíněný překlad
  - `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`
- Ostatní
  - `#`, `##`, `#error`, `#pragma`, ...
- [gcc.gnu.org/onlinedocs/cpp/](http://gcc.gnu.org/onlinedocs/cpp/)

# Chybné použití preprocesoru I

Obecně jen to, o čem víte jak to skutečně funguje a jste si vědomi důsledků.

závorky, závorky, závorky

- Priorita oprátořů

```
#define MULT(x,y) x*y
```

```
int z = MULT(3 + 2, 4 + 3);
```



# Chybné použití preprocesoru I

Obecně jen to, o čem víte jak to skutečně funguje a jste si vědomi důsledků.

závorky, závorky, závorky

- Priorita oprátořů

```
#define MULT(x,y) x*y
```

```
int z = MULT(3 + 2, 4 + 3);
```

```
int z = 3 + 2*4 + 3;
```

# Chybné použití preprocesoru I

Obecně jen to, o čem víte jak to skutečně funguje a jste si vědomi důsledků.

závorky, závorky, závorky

- Priorita oprátořů

```
#define MULT(x,y) x*y  
  
int z = MULT(3 + 2, 4 + 3);  
  
int z = 3 + 2*4 + 3;  
  
#define ADD_TWO(x) x+2  
  
int y = ADD_TWO(3) * 5;
```

# Chybné použití preprocesoru I

Obecně jen to, o čem víte jak to skutečně funguje a jste si vědomi důsledků.

závorky, závorky, závorky

- Priorita oprátorů

```
#define MULT(x,y) x*y  
  
int z = MULT(3 + 2, 4 + 3);  
  
int z = 3 + 2*4 + 3;  
  
#define ADD_TWO(x) x+2  
  
int y = ADD_TWO(3) * 5;  
  
int y = 3+2 * 5;
```

# Chybné použití preprocesoru I

Obecně jen to, o čem víte jak to skutečně funguje a jste si vědomi důsledků.

závorky, závorky, závorky

- Priorita oprátořů

```
#define MULT(x,y) x*y  
  
int z = MULT(3 + 2, 4 + 3);  
  
int z = 3 + 2*4 + 3;  
  
#define ADD_TWO(x) x+2  
  
int y = ADD_TWO(3) * 5;  
  
int y = 3+2 * 5;  
  
  
#define MULT(x,y) ((x) * (y))  
  
#define ADD_TWO(x) ((x)+2)
```

# Chybné použití preprocesoru II

závorky, závorky, závorky

- Závorky kolem bloku kódu

```
#define SWAP(x,y) a ^= b; b ^= a; a ^= b;
```

```
if (x < 0)  
    SWAP(x, y);
```

# Chybné použití preprocesoru II

závorky, závorky, závorky

- Závorky kolem bloku kódu

```
#define SWAP(x,y) a ^= b; b ^= a; a ^= b;
```

```
if (x < 0)
    SWAP(x, y);

else
    SWAP(y, x);
```

# Chybné použití preprocesoru II

závorky, závorky, závorky

- Závorky kolem bloku kódu

```
#define SWAP(x,y) a ^= b; b ^= a; a ^= b;
```

```
if (x < 0)
    SWAP(x, y);
```

```
else
    SWAP(y, x);
```

```
#define SWAP(x,y) do { \
    a ^= b;           \
    b ^= a;           \
    a ^= b;           \
} while (0)
```

# Chybné použití preprocesoru III

- Změna toku programu

```
#define FOO(x)          \  
do {                  \  
    if (blah(x) < 0)  \  
        return -EBUGGERED; \  
} while(0)
```



# Chybné použití preprocesoru III

- Změna toku programu

```
#define FOO(x)          \  
do {                  \  
    if (blah(x) < 0)  \  
        return -EBUGGERED; \  
} while(0)
```

- Přístup k lokálním proměnným

```
#define FOO(val) bar(index, val)
```

# Chybné použití preprocesoru III

- Změna toku programu

```
#define F00(x)          \  
do {                  \  
    if (blah(x) < 0)  \  
        return -EBUGGERED; \  
} while(0)
```

- Přístup k lokálním proměnným

```
#define F00(val) bar(index, val)
```

- Makro jako l-hodnota

```
F00(x) = y;
```

# Chybné použití preprocesoru III

- Změna toku programu

```
#define FOO(x)          \  
do {                  \  
    if (blah(x) < 0)  \  
        return -EBUGGERED; \  
} while(0)
```

- Přístup k lokálním proměnným

```
#define FOO(val) bar(index, val)
```

- Makro jako l-hodnota

```
FOO(x) = y;
```

- Argumenty s vedlejšími efekty

```
#define MIN(a,b) ((a)>(b)?(b):(a))  
  
int z = MIN(x++, y--);
```

# Chybné použití preprocesoru III

- Změna toku programu

```
#define FOO(x)          \  
do {                  \  
    if (blah(x) < 0)  \  
        return -EBUGGERED; \  
} while(0)
```

- Přístup k lokálním proměnným

```
#define FOO(val) bar(index, val)
```

- Makro jako l-hodnota

```
FOO(x) = y;
```

- Argumenty s vedlejšími efekty

```
#define MIN(a,b) ((a)>(b)?(b):(a))
```

```
int z = MIN(x++, y--); // x++ a y-- jsou vyhodnoceny 2 krát!
```

# Chybné použití preprocesoru IV

- Středníky

```
#define PRETTY_PRINT(msg) printf(msg);  
if (n < 10)  
    PRETTY_PRINT("n is less than 10");  
else  
    PRETTY_PRINT("n is at least 10");
```

# Chybné použití preprocesoru IV

- Středníky

```
#define PRETTY_PRINT(msg) printf(msg);  
if (n < 10)  
    PRETTY_PRINT("n is less than 10");  
else  
    PRETTY_PRINT("n is at least 10");
```

- Pokud můžete téhož dosáhnout funkcí, nepoužívejte makra

## Logování událostí

chybové a debugovací výpisy, syslogd

# Formát zpráv

Každá zpráva **MUSÍ** být **SROZUMITELNÁ!**

Zvažte

- kdo bude zprávy číst – jinak bude vypadat debugovací hláška a jinak chybová hláška, kterou uvidí uživatel
- kolik zpráv je potřeba (podrobnost výpisu) – je dobrý nápad mít několik úrovní vypisovaných hlášek
- jak/čím zprávu zobrazit – stderr, logovací soubory, email, ...



# Zprávy pro debugovací účely

Zprávy jsou pro vás – vypisujte informace, které mají smysl a hlavně identifikují místo/zdroj problému

Využívejte makra preprocesoru

- `__FILE__`, `__LINE__`, (`__func__`)
- `__DATE__`, `__TIME__`

# Zprávy pro debugovací účely

Zprávy jsou pro vás – vypisujte informace, které mají smysl a hlavně identifikují místo/zdroj problému

Využívejte makra preprocesoru

- `__FILE__`, `__LINE__`, `(__func__)`
- `__DATE__`, `__TIME__`

## DEBUG\_MSG

```
extern int debug_level;
#ifdef DEBUG
# define DEBUG_MSG(level, ...) do {           \
    if (debug_level >= level) {             \
        fprintf(stderr, __VA_ARGS__);      \
    }                                       \
} while (0)
#else
# define DEBUG_MSG(level, ...) ((void) 0)
#endif
```

# Syslog

- Komplexní systém pro logování informací pro případnou pozdější analýzu.
- Zprávy lze dělit podle zdroje i podle důležitosti.
- Zprávy lze zapisovat na konkrétní zařízení (tiskárna, konzole), do lokálních souborů (`/var/log/`) nebo na vzdálený server.
- V prostředí GNU/Linuxu postupně několik generací logovacích nástrojů (`syslogd(8)`, `syslog-ng(8)` a `rsyslogd(8)`)
- Konfigurace je např. v `/etc/rsyslog.conf/`, `/etc/rsyslog.d/*`<sup>1</sup>

## Funkce ze `syslog.h`:

- `openlog(3)`
- `syslog(3)`
- `closelog(3)`

---

<sup>1</sup>podle použitého daemona se může lišit

## úkol

- použijte připravený debug.c a rozšiřte makro `DEBUG_MSG` o možnost zapisovat do syslogu

## Testování

na co si dávat pozor a jak chyby řešit

# Motivace

## raketa Ariane 5

- nastala výjimka jejíž ošetření bylo vypnuté z výkonostních důvodů
- havárie sw → havárie rakety

## havárie Mars Polar Lander

- signál ze senzorů byl interpretován jako dosednutí na zem ačkoli zařízení bylo asi ve 40 m
- chyba byla v jediném řádku kódu

# Typologie chyb I

## ● Textové chyby

- typografie, gramatika a překlepy
- chyby v překladu, problémy s dokumentací
- obvykle snadná oprava, nízká priorita
- úloha pro spell-checker

## ● Pády aplikace

- Segmentation fault, buffer overflows, . . .
- kritický význam, často těžko reprodukovatelné
- úloha pro debugger

## ● Neočekávané chování

- program se chová jinak, než uživatel očekává nebo/a je popsáno v dokumentaci
- význam se může lišit, často přístup bug → feature

# Typologie chyb II

## ● Memory leaky

- program neuvolňuje alokovanou paměť
- význam se liší podle typu aplikace
- úloha pro valgrind a spol.

## ● Výkonostní problémy

- v určitých situacích program významně zpomalí a konzumuje nepřiměřené množství zdrojů
- význam obvykle vysoký
- úloha pro profilování

## ● Bezpečnostní problémy

- problémy s přístupovými právy, neoprávněnými zásahy do paměti, uživatelským vstupem, souběhem, . . .
- význam se liší podle účelu aplikace



# Postupy – co nefunguje

- Kompletní testování** stavový prostor je příliš velký (až nekonečný)
- Statistické testování** nefunguje u SW – výskyt chyby je ovlivnitelný  
příliš mnoha faktory (příliš mnoho náhodných veličin)

Testování a hledání chyb je otázkou ceny, jakou chceme zaplatit.

# Postupy – co funguje

**unit testy** testování malých, jasně definovaných částí.  
Následovat může testování subsystémů a celého systému.

**automatizace** zvyšuje frekvenci testů a snižuje jejich cenu

**testování vstupů** – náhodné/systematické/hraniční

Testujte co nejdříve – rychle odhalená chyba je levnější

# assert()

```
#include <assert.h>
void assert(scalar expression);
```

- testování podmínky a ukončení aplikace v případě její nesplnění
- pokud je definované makro NDEBUG, assert() nic neprovádí
- pozor na vedlejší efekty testovacích podmínek
- pozor na testování vstupů

# Unit testy – framework check

- [check.sourceforge.net](http://check.sourceforge.net)
- jeden z mnoha Unit test frameworků
- hlavičkový soubor: `#include <check.h>`
- knihovna: `-lcheck`

# Unit testy – framework check

- [check.sourceforge.net](http://check.sourceforge.net)
- jeden z mnoha Unit test frameworků
- hlavičkový soubor: `#include <check.h>`
- knihovna: `-lcheck`

## úkol

- naimplementujte funkci kontrolující zletilost uživatele
- funkce vrací false (0) v případě, že od narození uživatele (`born`) do času kontroly (`check_time`) neuběhlo více než 18 let
- jinak vrací true
- pro tuto funkci si vytvořte Unit test ve frameworku check

```
#include <time.h>
int is_mature(const struct tm *born, const struct tm *check_time);
```

## Nástroje pro debuggování a reverzní inženýrství

debuggery, valgrind, ltrace, strace, ldd, file, object, nm, strip, size, objdump, perf, ...

# Debugery

`gdb(1)`

- textový debugger
- základem pro grafické nadstavby `ddd(1)`, `kdbg`, ...

# Debugery

`gdb(1)`

- textový debugger
- základem pro grafické nadstavby `ddd(1)`, `kdbg`, ...

**DEMO**



# Valgrind a spol.

Sada nástrojů pro debugování a profilování.

`--tool=`

- memcheck
- cachegrind
- callgrind
- ... (<http://valgrind.org/info/tools.html>)

# Valgrind a spol.

Sada nástrojů pro debugování a profilování.

`--tool=`

- memcheck
- cachegrind
- callgrind
- ... (<http://valgrind.org/info/tools.html>)

## DEMO

# Valgrind a spol.

Sada nástrojů pro debugování a profilování.

`--tool=`

- `memcheck`
- `cachegrind`
- `callgrind`
- ... (<http://valgrind.org/info/tools.html>)

## DEMO

### Další užitečné přepínače

- `--leak-check=yes`
- `--dump-instr=yes`
- `--cache-sim=yes`

# Profilování

Vyhledání míst vhodných k optimalizaci – **řešte to, co má smysl!**

## Postup

- 1 (překlad s nestandardními volbami překladače)
- 2 vygenerování profilovacích dat
- 3 analýza profilovacích dat

## Nástroje

- gprof (překlad programu s volbou -pg)
- callgrind, kcachegrind

# Profilování

Vyhledání míst vhodných k optimalizaci – **řešte to, co má smysl!**

## Postup

- 1 (překlad s nestandardními volbami překladače)
- 2 vygenerování profilovacích dat
- 3 analýza profilovacích dat

## Nástroje

- gprof (překlad programu s volbou -pg)
- callgrind, kcachegrind

## DEMO

# Performance Counter Subsystem (perf)

- Abstrakce hardwarových čítačů moderních CPU
- Součást přímo kernelu od verze 2.6.31
- [https://perf.wiki.kernel.org/index.php/Main\\_Page](https://perf.wiki.kernel.org/index.php/Main_Page)
- Minimální režie, možnosti měření v kernel- i user-space
- Měřit lze i již běžící aplikace

## Použití

- `perf record -- ./mujprog`
- `perf report`

# Další nástroje (reverzního inženýrství)

- file – informace o souboru
- objdump – vypíše obsah objektového souboru
- ldd – použité dynamické knihovny
- strace – sledování systémových volání a signálů
- ltrace – sledování volání knihovních funkcí
- tcpdump/wireshark – monitorování síťové komunikace

# Souborový systém procfs

V Linuxu je (téměř) všechno soubor – i procesy a parametry jádra jsou *soubory* dostupné v souborovém systému procfs (/proc).

- obsah souborů generuje jádro při každém open()
- největší část obsahuje informace o běžících procesech
- lze nejen číst, ale i zapisovat a měnit tak chování systému

[www.root.cz/serialy/co-pred-nami-taji-proc/](http://www.root.cz/serialy/co-pred-nami-taji-proc/)



# Souborový systém procfs

V Linuxu je (téměř) všechno soubor – i procesy a parametry jádra jsou *soubory* dostupné v souborovém systému procfs (/proc).

- obsah souborů generuje jádro při každém open()
- největší část obsahuje informace o běžících procesech
- lze nejen číst, ale i zapisovat a měnit tak chování systému

[www.root.cz/serialy/co-pred-nami-taji-proc/](http://www.root.cz/serialy/co-pred-nami-taji-proc/)

- sysfs
- debugfs

## Závěr

domácí úkoly a zdroje

# Domácí úkol

- Připravte si pro další úlohy vlastní knihovnu s debugovacími makry a funkcemi
  - výpis chybových hlášek
  - výpis debugovacích hlášek
  - verze s výstupem na stderr i do syslogu (případně další výstupy)
  - různé úrovně výpisů
  - ...
- Seznamte se s možnostmi různých ladících nástrojů a vyberte si vhodné nástroje pro vás.

# Zdroje

## syslog

- [www.gnu.org/s/libc/manual/html\\_node/Submitting-Syslog-Messages.html](http://www.gnu.org/s/libc/manual/html_node/Submitting-Syslog-Messages.html)

## debuging/profiling

- [www.alexonlinux.com/how-debugger-works](http://www.alexonlinux.com/how-debugger-works)
- [www.kdbg.org/](http://www.kdbg.org/)
- [perf.wiki.kernel.org/index.php/Main\\_Page](http://perf.wiki.kernel.org/index.php/Main_Page)
- [www.fit.vutbr.cz/~martinek/clang/profiling.html.cs](http://www.fit.vutbr.cz/~martinek/clang/profiling.html.cs)
- [valgrind.org/docs/manual/QuickStart.html](http://valgrind.org/docs/manual/QuickStart.html)
- [kcachegrind.sourceforge.net/html/Home.html](http://kcachegrind.sourceforge.net/html/Home.html)