

Internacionalizace aplikací

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

Radek Krejčí

Fakulta informatiky
Masarykova univerzita
radek.krejci@mail.muni.cz

Brno, 15. prosince 2010

Internacionalizace aplikací

Postupy pro lokalizaci aplikací

Národní prostředí – z pohledu uživatele I

Prizpůsobení aplikací národnímu prostředí uživatele bez nutnosti rekompilace.

```
man 7 locale
```

Různé kategorie

`LC_COLLATE` – třídění řetězců

`LC_CTYPE` – typy znaků

`LC_MESSAGES` – jazyk zpráv

`LC_MONETARY` – formát měnových řetězců

`LC_NUMERIC` – formát čísel

`LC_TIME` – formát času, názvů měsíců, dnů v týdnu

...

Národní prostředí – z pohledu uživatele II

Proměnné prostředí

`LANG` – implicitní hodnota pro kategorie

`LC_*` – nastavení jednotlivých kategorií

`LC_ALL` – přebíjí vše

Formát locales

`Jazyk` – podle ISO 639 (cs, en, ...)

`Země` – podle ISO 3316 (CZ, US, GB, ...)

`Znaková sada` – ISO8859-2, UTF-8, ...

Příklady: `cs_CZ.ISO8859-2`, `en_US.UTF-8`

Změna národního prostředí v aplikaci

```
#include <locale.h>
char *setlocale(int category, const char *locale);
```

Aplikace by při startu měla zavolat:

```
setlocale(LC_ALL, "");
```

- Výchozí hodnota po spuštění aplikace je "C" nebo "POSIX"

Internacionalizace aplikací

Internacionalizace – program komunikuje v jazyce, který si uživatel zvolí

Lokalizace – program se chová (požaduje vstup, produkuje výstup, ...) podle lokálních zvyklostí

Internacionalizace aplikací

Internacionalizace – program komunikuje v jazyce, který si uživatel zvolí

Lokalizace – program se chová (požaduje vstup, produkuje výstup, ...) podle lokálních zvyklostí

PO (Portable Object) – soubor identifikátorů řetězců spolu s jejich překladem do konkrétního jazyka

MO (Machine Object) – binární obdoba PO souborů určená pro použití aplikacemi

PO soubor – ukázka

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2010-12-12 18:08+0100\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: hello.c:14
#, c-format
msgid "Hello world\n"
msgstr ""
```

GNU gettext()

```
#include <libintl.h>
char * gettext (const char * msgid);
char * dgettext (const char * domainname, const char * msgid);
char * dcgettext (const char * domainname, const char * msgid,
                 int category);
char * bindtextdomain (const char * domainname, const char * dirname);
char * textdomain (const char * domainname);
```

- S minimálními zásahy do kódu umožňuje internacionalizaci aplikací.
- Umožňuje rozdělit práci mezi programátora a překladače.

Internacionalizace – použití I

- 1 Napsat aplikaci s využitím `gettext()`
`#define _(STRING) gettext(STRING)`
- 2 Správně nastavit `locale` a doménu
`setlocale(LC_ALL, "");`
`bindtextdomain("domain", "/usr/share/locale");`
`textdomain("domain");`
- 3 Vytvoření generického PO souboru
`xgettext -d prg --keyword=_ prg.c`
- 4 Vyrobení PO souboru s překladem pro konkrétní jazyk
`msginit -l cs_CZ.utf8 -i prg.po`
- 5 Vygenerování MO souboru
`msgfmt -c -o prg.mo cs.po`
- 6 Umístění souboru na správné místo (instalace)

úkol

- Napište Hello, world! program.
- Výchozí jazyk aplikace je angličtina.
- Připravte lokalizaci do češtiny.

C Preprocesor

aneb Co všechno je ještě Céčko?
(Inspirováno přednáškou Rudy Čejky, FIT VUT)

Co umí preprocesor

- Vkládání zdrojových souborů
 - Zcela libovolné soubory s libovolným obsahem!

Co umí preprocesor

- Vkládání zdrojových souborů
 - Zcela libovolné soubory s libovolným obsahem!
- Předdefinovaná makra
 - `__LINE__`, `__DATE__`, `__TIME__`, ...

Co umí preprocesor

- Vkládání zdrojových souborů
 - Zcela libovolné soubory s libovolným obsahem!
- Předdefinovaná makra
 - `__LINE__`, `__DATE__`, `__TIME__`, ...
- Definice maker
 - `#define MA printf("Dneska je úžasný den!\n")`
`#define MB(parametr) parametr`
`#define MC(fmt, ...) vprintf(fmt, __VA_ARGS__)`

Co umí preprocesor

- Vkládání zdrojových souborů
 - Zcela libovolné soubory s libovolným obsahem!
- Předdefinovaná makra
 - `__LINE__`, `__DATE__`, `__TIME__`, ...
- Definice maker
 - `#define MA printf("Dneska je úžasný den!\n")`
`#define MB(parametr) parametr`
`#define MC(fmt, ...) vprintf(fmt, __VA_ARGS__)`
- Podmíněný překlad
 - `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`

Co umí preprocesor

- Vkládání zdrojových souborů
 - Zcela libovolné soubory s libovolným obsahem!
- Předdefinovaná makra
 - `__LINE__`, `__DATE__`, `__TIME__`, ...
- Definice maker
 - `#define MA printf("Dneska je úžasný den!\n")`
`#define MB(parametr) parametr`
`#define MC(fmt, ...) vprintf(fmt, __VA_ARGS__)`
- Podmíněný překlad
 - `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`
- Ostatní
 - `#error`, `#pragma`, `##`, ...

Co umí preprocesor

- Vkládání zdrojových souborů
 - Zcela libovolné soubory s libovolným obsahem!
- Předdefinovaná makra
 - `__LINE__`, `__DATE__`, `__TIME__`, ...
- Definice maker
 - `#define MA printf("Dneska je úžasný den!\n")`
`#define MB(parametr) parametr`
`#define MC(fmt, ...) vprintf(fmt, __VA_ARGS__)`
- Podmíněný překlad
 - `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`
- Ostatní
 - `#error`, `#pragma`, `##`, ...
- gcc.gnu.org/onlinedocs/cpp/

Co si tedy můžeme dovolit? I

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

Co skutečně nikdy nedělat

- Makra ovlivňující tok programu

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

Co si tedy můžeme dovolit? I

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

Co skutečně nikdy nedělat

- Makra ovlivňující tok programu

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

- Makra závisující na lokálních proměnných

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

Co si tedy můžeme dovolit? I

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

Co skutečně nikdy nedělat

- Makra ovlivňující tok programu

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

- Makra závisující na lokálních proměnných

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

- Makra jako l-hodnota

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

Co si tedy (ne)můžeme dovolit? II

- Pozor na prioritu operátorů

```
#define RADTODEG(x) (x * 57.29578)
```

Co si tedy (ne)můžeme dovolit? II

- Pozor na prioritu operátorů

```
#define RADTODEG(x) (x * 57.29578)
```

```
RADTODEG(a + b) → (a + b * 57.29578)
```

Co si tedy (ne)můžeme dovolit? II

- Pozor na prioritu operátorů

```
#define RADTODEG(x) (x * 57.29578)
```

```
RADTODEG(a + b) → (a + b * 57.29578)
```

```
#define RADTODEG(x) ((x) * 57.29578)
```

Co si tedy (ne)můžeme dovolit? II

- Pozor na prioritu operátorů

```
#define RADTODEG(x) (x * 57.29578)
```

```
RADTODEG(a + b) → (a + b * 57.29578)
```

```
#define RADTODEG(x) ((x) * 57.29578)
```

- Vícenásobné vyhodnocování výrazů

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

Co si tedy (ne)můžeme dovolit? II

- Pozor na prioritu operátorů

```
#define RADTODEG(x) (x * 57.29578)
```

```
RADTODEG(a + b) → (a + b * 57.29578)
```

```
#define RADTODEG(x) ((x) * 57.29578)
```

- Vícenásobné vyhodnocování výrazů

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

```
MIN(a++, b--), a i b jsou vyhodnocovány 2x!
```

Co si tedy (ne)můžeme dovolit? II

- Pozor na prioritu operátorů

```
#define RADTODEG(x) (x * 57.29578)
RADTODEG(a + b) → (a + b * 57.29578)
#define RADTODEG(x) ((x) * 57.29578)
```

- Vícenásobné vyhodnocování výrazů

```
#define MIN(a,b) ((a)>(b)?(b):(a))
MIN(a++, b--), a i b jsou vyhodnocovány 2x!
```

- Středníky v makrech – nepoužívat

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

Co si tedy (ne)můžeme dovolit? II

- Pozor na prioritu operátorů

```
#define RADTODEG(x) (x * 57.29578)
RADTODEG(a + b) → (a + b * 57.29578)
#define RADTODEG(x) ((x) * 57.29578)
```

- Vícenásobné vyhodnocování výrazů

```
#define MIN(a,b) ((a)>(b)?(b):(a))
MIN(a++, b--), a i b jsou vyhodnocovány 2x!
```

- Středníky v makrech – nepoužívat

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

- A spousta dalších problémů – en.wikipedia.org/wiki/C_preprocessor

Závěr

shrnutí, domácí úkoly a zdroje

Domácí úkol

Dokončete úlohy ze všech cvičení, uložte do SVN a oznamte to emailem cvičícímu

Zdroje

lokalizace a internacionalizace

- www.gnu.org/software/gettext/manual/gettext.html
- www.tuxamito.com/joomla/index.php/en/component/content/article/60-gettext-tutorial

C preprocesor

- gcc.gnu.org/onlinedocs/cpp/
- en.wikipedia.org/wiki/C_preprocessor