Úvod, základní nástroje pro vývoj v prostředí GNU/Linux

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

Jiří Novosad

Fakulta informatiky
Masarykova univerzita
novosad@fi.muni.cz

Brno, září 2015

Představení

Mgr. Jiří Novosad

- CVT FI (unix@fi, IS)

kontakt

email novosad@fi.muni.cz
kancelář místnost B205

slidy

Úvodní informace o kurzu

náplň kurzu a podmínky úspěšného absolvování

Náplň výuky

- obecně vývoj v prostředí GNU/Linux a nástroje, které se vám budou hodit,
- procesy a vlákna,
- komunikace mezi procesy,
- synchronizace a vzájemné vyloučení,
- pokročilé operace se soubory (select, poll, fcntl, ioctl, . . .)
- ladění a debugování aplikací
- . . . (*„témata na přání“*)

Cílem není naučit se konstrukce jazyka, které neznáte, ale **vyzkoušet** si vývoj aplikací s využitím toho, co nabízí GNU/Linux.

Administrativní informace

Požadavky na ukončení

- účast je povinná (max. 2 neomluvené absence)
- domácí úkoly a úlohy z hodiny

Odevzdávání úloh – SVN

- fakultní SVN `https://fadmin.fi.muni.cz/auth/sys/svn_ucty.mpl`
- vytvořte repozitář `pb173` kam budete commitovat svou práci
- pro tento repozitář přidejte právo číst uživateli `xnovosa1`
- úkoly ze cvičení ukládejte do adresářů ve formátu `exNN`, domací úlohy `hwNN` (`ex01`, `hw01`, ...)
- slidy, různé zdrojáky a další materiály najdete ve studijních materiálech v ISu (Skupina 02)

Obecné požadavky na úkoly

- Odevzdávají se zdrojové soubory, ne binárky, ne generované soubory.
- Spolu se zdrojovými soubory se odevzdává i **Makefile** – ke kompilaci by mělo stačit spustit `make` (případně `README`)
- Dokumentujte! Je ve vašem zájmu, aby bylo použití programu pro opravujícího co nejjednodušší.
- Dodržujte coding-style, neopisujte!
- Neopisujte!

Bodování

- Odevzdávají se všechny úlohy započaté na cvičení – za ty je možno získat až 5 bodů
- a domácí úkoly – až 10 bodů.
- Obojí odevzdat do 10 dnů od cvičení (tedy do pátku následující týden, do 23:59).
- V případě omluvené neúčasti individuální posunutí termínu (napište e-mail).
- Selže kompilace / nejde spustit / v triviálních případech končí se Segmentation fault / vůbec nedělá, co má $\Rightarrow$ 0 bodů
- Hodnotím i jak kód vypadá a jestli jsou ošetřeny chybové stavy.
- Testujte na Nymfách s Ubuntu (nymfe23 – nymfe105).

E-maily

O odevzdání úlohy mě informujte e-mailem, do předmětu uveďte PB173. Příklad:

```
Subject: PB173 - ukol 01 - odevzdani

Dobry den,

odevzdal jsem ukol cislo 01.

Jan Uzivatel
xuzivat1
321567
```

Jakmile úlohu opravím, tak na tento e-mail odpovím. Upozorním vás na chyby a sdělím vám počet získaných bodů.

Druhý pokus

Pokud odevzdáte úlohu v termínu, máte právo na druhý, opravný, pokus. V tom případě musíte odevzdat úlohu do 7 dnů od opravy. Upozorněte mě odpovědí na můj opravný e-mail. Příklad:

```
Subject: PB173 - ukol 01 - oprava

Dobry den,

odevzdal jsem opravu ukolu 01.

Jan Uzivatel
xuzivat1
321567
```

Kultura kódu

coding-style, dokumentace, commit policy

Coding-style I

Myslete na to, že kód po vás bude číst někdo další.

Coding style obvykle zahrnuje pravidla pro:

- odsazování
- závorky a mezery
- pojmenování proměnných a funkcí
- komentáře
- používání struktur jazyka – typedef, makra, enum, ...
- návratové hodnoty funkcí
- ...

Coding-style II

Existuje mnoho různých coding-stylů a většinou si dost protiřečí – není důležitý konkrétní formát, ale to, že **je jednotný pro celý kód a že ho dodržují všichni** přispěvatelé projektu.

Skutečné ukázky

- K&R coding style
- Kernel Normal Form (BSD coding style)
  `http://en.wikipedia.org/wiki/Kernel_Normal_Form`
- Linux Kernel Coding Style
  `http://www.kernel.org/doc/Documentation/CodingStyle`
- GNU coding standards
  `http://www.gnu.org/prep/standards/`

Coding-style - ukázka Linux Kernel Coding Style

```
int function(int x)
{
        int a;

        if (x == 0) {
                a = 0;
        }

        switch (x) {
        case '0':
                a = 1;
                break;
        case '1':
                a = 0;
                /* fall through */
        default:
                break;
        }

        return a;
}
```

domácí úkol (do příští hodiny)

- Projděte si různé coding styly a pro jeden se rozhodněte (případně si ho upravte podle svých potřeb).
- Do své SVNky uložte popis svého coding stylu (`CodingStyle`) a dodržujte ho v dalších úlohách.
- Tip: existují programy na kontrolu dodržování stylu (např. Linux)

Dokumentace

Často se to zdá zbytečné, ale **pište komentáře a dokumentaci !**

Co je dokumentace:

Dokumentace

Často se to zdá zbytečné, ale **pište komentáře a dokumentaci !**

Co je dokumentace:

- komentáře v kódu (doxygen)
- krátká README
- man stránky
- handbooky, tutoriály, howtos – podle zaměření (uživatel/vývojář)

Doxygen

Nástroj pro automatické generování dokumentace.

`www.doxygen.org/`

příklad:

`http://dbus.freedesktop.org/doc/api/html/index.html`

Commit policy

V zásadě jde o pravidla práce s verzovacím systémem (CVS, SVN, GIT, . . .)

- přístupová práva
- pravidla pro adresářovou strukturu
- **pravidla popisu provedených změn**
- `http://techbase.kde.org/Policies/SVN_Commit_Policy`

Přenositelnost

aneb kde všude funguje váš kód?

Nikdo nechce, aby váš program fungoval všude, ale musíte vědět **PROČ** nefunguje.

- rozšíření kompilátoru
- závislosti na knihovnách/prostředí/architektuře/...

Při řešení problémů s přenositelností dobře slouží Autotools.

Přenositelnost – datové typy

Problémy

- 32b vs. 64b architektura

  ```
  #include <limits.h>
  #include <float.h>
  #include <stdint.h>
  ```

  Normy (C99) udávají pouze minimální velikost jednotlivých datových typů.

  en.wikipedia.org/wiki/C_data_types

- little vs. big endian (network byte order)

  `#include <endian.h>` (`man 3 endian`)

  případně `<byteswap.h>`

  a dále funkce jako `ntohs`, `ntohl`, `htons`, `htonl`

Shrnutí a další rady

Modularita kódu

- udržujte pohromadě jen to, co spolu skutečně souvisí
- rozdělujte kód podle funkcionality
- každá funkce by měla dělat jednu konkrétní věc
- neduplikujte kód

Dokumentace

- pište komentáře a dokumentaci!
- myslete na toho, kdo bude dokumentaci číst
- jednou to pravděpodobně budete vy

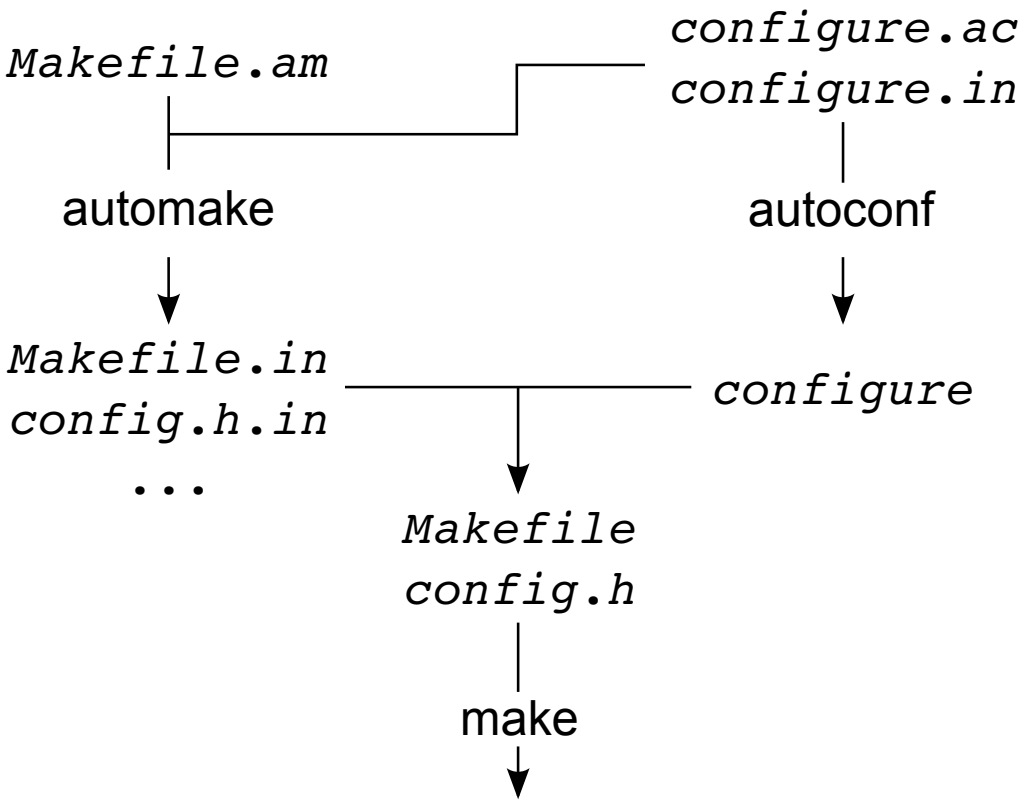
Čitelnost

- pozor na tzv. magické konstanty – použijte alespoň komentované makro
- pozor na mrtvý kód – debugovací kód, stará funkcionalita

Úvod do prostředí GNU/Linux

autotools, gcc, dynamické knihovny, zdroje informací

GNU Autotools – základní přehled



Cílem je usnadnit práci tvůrci aplikace a především uživateli – umožnit mu standardně přeložit a nainstalovat aplikaci:

```
# ./configure && make && make install
```

GNU Autoconf

- makro procesor (převlečený GNU M4)
- kromě nástroje `autoconf` obsahuje balík i další nástroje (`autoreconf`, `autoscan`, `aclocal`, ...)
- `autoreconf` zaručí spuštění všech potřebných nástrojů GNU Autoconf
- vstupem je soubor `configure.ac`
- výstupem je soubor `configure`
- www.gnu.org/software/autoconf/manual/autoconf.html

configure

- shell skript pro kontrolu konfigurace překladového prostředí (programy, knihovny, hlavičkové soubory, ...)
- vstupem je `Makefile.in` (`config.h.in`)
- výstupem `Makefile` (`config.h`)

GNU Automake

- generování Makefile.in ze šablony
- obsahuje sadu specifických proměnných pro `make`
- vstupem je `Makefile.am` a `configure.ac`
- výstupem je soubor `Makefile.in`
- poměrně často se tento krok přeskakuje a ručně se upravuje až existující `Makefile.in`

Makefile.am:

```
bin_PROGRAMS = mybinary
mybinary_SOURCES = main.c
```

configure.ac:

```
AC_INIT([hello], [0.1], [xuser@fi.muni.cz])
AM_INIT_AUTOMAKE([foreign -Wall -Werror])
AC_PROG_CC
AC_OUTPUT(Makefile)
```

GNU Make (I)

- automatický „překlad“
- nejen překlad, často generování dokumentace a další

soubor Makefile

- proměnné: `CC = gcc` a použití `$(CC)`
- cíle – co se má vytvořit
- závislosti – za jakých podmínek se to má vytvořit
- pravidla/příkazy – jak se to má vytvořit (začínají **Tab**!!!)
- `# make -f Makefile target`
- dále funkce, falešné cíle, podmínky, . . .

běžné cíle

- all
- `%.o: %.c`
- clean

GNU Make (II)

implicitní pravidla

- `$(CC) $(CPPFLAGS) $(CFLAGS) -c`
- `$(CC) $(LDFLAGS) n.o $(LOADLIBES) $(LDLIBS)`
- `...`

implicitní proměnné

- `$(CC)`
  – kompilátor jazyka C
- `$(CPP)`
  – C preprocessor
- `$(CFLAGS), $(CPPFLAGS)`
  – flagy/parametry pro kompilátor/preprocesor
- `$(LDFLAGS)`
  – flagy pro kompilátor při linkování
- `...`

úkol

- Napište program helloworld
- Připravte si pro něho configure.ac a Makefile.am
- Tip: využijte `autoscan`, `autoreconf -i`

úkol

- Napište program helloworld
- Připravte si pro něho configure.ac a Makefile.am
- Tip: využijte `autoscan`, `autoreconf -i`
- přidejte do `configure.ac` parametr `--enable-debug`, který přidá do vygenerované binárky kód vypisující ID procesu (`getpid(2)`, `unistd.h`)
- v ideálním případě tak, aby fungoval (automaticky generovaný) přepínač `--disable-debug` podle očekávání
- Tip: AC_ARG_ENABLE, AC_DEFINE, AS_IF (, AM_CONDITIONAL)

GCC

- překladač nejen jazyka C
- standardní překladač na dnešních UNIX-like systémech
- dostupný pro desítky platforem

```
# gcc -o jmeno_programu <vstupni_soubory>
```

GCC - důležité přepínače

Více najdete v man stránce

- `-std=` – použití konkrétního standardu jazyka (gnu89)
- `-c` – nepouštět linker (pouze překlad)
- `-g` – přidat debugovací informace
- `-pg` – přidat profilovací instrukce
- `-O<num>` – optimalizace kódu
- `-I` – cesta kde hledat hlavičkové soubory
- `-l<lib>` – přilinkovat knihovnu
- `-L<path>` – cesta kde hledat knihovny
- `-D<makro>` – definice makra
- `-Wall` – zapne hlavní sadu varování
- `-WExtra` – zapne další důležitá varování

GCC – standardní C a viditelnost POSIX funkcí

- použitím přepínače `-std=`, např. `-std=c99`, jsou schovány všechny funkce a objekty, které nejsou součástí příslušného standardu
- programátor může rozhodnout, které objekty budou viditelné, pomocí speciálních maker, která definuje před includováním hlavičkových souborů
- nejsnáze toho dosáhne použitím přepínače `-D`
- nejčastěji chceme povolit funkce nejnovějšího standardu POSIX, definujeme makro `_POSIX_C_SOURCE` s hodnotou `200809L`
- viz také `man 7 feature_test_macros`

```
# gcc -std=c99 -D_POSIX_C_SOURCE=200807L <vstupni_soubory>
```

úkol

- použijte vytvořený helloworld a jeho configure/Makefile
- vyzkoušejte si různé parametry gcc – které parametry mají vliv (a jaký) na velikost výsledného programu?

Závěr

domácí úkoly a zdroje

Domácí úkol

- dokončete co jste nestihli na cvičeních
- projděte si odkazované zdroje pro práci s Autotools
- připravte si šablonu vlastního Makefilu pro úkoly během semestru
- `hw01/CodingStyle`

Zdroje

obecně

- **používejte man stránky!**
  - 0 – hlavičkové soubory
  - 1 – uživatelské příkazy (aplikace)
  - 2 – systémová volání
  - 3 – knihovní funkce
  - 4 – speciální soubory (`/dev`)
  - 5 – formáty souborů, popis protokolů, ...
  - 6 – hry
  - 7 – různé
  - 8 – systémové příkazy (systémoví daemoni)
- `man 1 printf` vs. `man 3 printf`

Zdroje

obecně

- **používejte man stránky!**
  - 0 – hlavičkové soubory
  - 1 – uživatelské příkazy (aplikace)
  - 2 – systémová volání
  - 3 – knihovní funkce
  - 4 – speciální soubory (`/dev`)
  - 5 – formáty souborů, popis protokolů, ...
  - 6 – hry
  - 7 – různé
  - 8 – systémové příkazy (systémoví daemoni)
- `man 1 printf` vs. `man 3 printf`
- info(1)
- zdrojáky (/usr/src/linux)
- Google

Zdroje

- www.lrde.epita.fr/~adl/autotools.html
- www.gnu.org/software/make/manual/
- www.gnu.org/software/automake/manual/
- www.freesoftwaremagazine.com/books/autotools_a_guide_to_autoconf_automake_libtool