

Základy vývoje v prostředí GNU/Linux

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

Petr Velan

Fakulta informatiky
Masarykova univerzita
velan@ics.muni.cz

Brno, 24. září 2014

Zvyklosti vývoje v prostředí GNU/Linux

zvyklosti v chování aplikací, interakce aplikací s okolím

Adresářová struktura GNU/Linux

/bin/	obecné systémové binárky pro všechny uživatele
/boot/	soubory potřebné pro zavaděč, jádro
/dev/	soubory zařízení (disky, usb, porty, ...)
/etc/	konfigurační soubory
/lib/	sdílené systémové knihovny
modules/	moduly jádra
/proc/	speciální virtuální filesystém obsahující informace o jádře
/sbin/	obecné systémové binárky pro privilegované uživatele
/tmp/	dočasné soubory
/usr/	soubory potřebné pro běžnou práci
bin/	binárky programů
doc/	dokumentace aplikací
include/	hlavičkové soubory
lib/	další sdílené knihovny
local/	aplikace, knihovny, ... instalované ze zdrojových souborů
man/	manuálové stránky
sbin/	další systémové binárky
share/	konfigurační soubory a další nastavení aplikací
/var/	pracovní (měnící se) soubory pro běžící programy a služby

Parametry příkazové řádky

Základní způsob předávání vstupních informací programu.
Pravidla popsána v rámci normy POSIX → GNU Standards.
Používejte standardní názvy parametrů a standardní chování.
Obvyklá syntax přepínačů:

- -<pismo>
- -<pismo> <argument>
- --
- --<slovo>
- --<slovo> <argument>
- --<slovo>=<argument>

info "(standards)User Interfaces"

Využívejte standardní funkce:

- getopt – zpracovává krátké parametry
- getopt_long – zpracovává dlouhé parametry, není součástí POSIX (GNU rozšíření)

Proměnné prostředí

Slouží k nastavení chování aplikace.

Z příkazové řádky:

```
export MYVAR=ahoj  
echo $MYVAR
```

Z aplikace:

- `getenv()` – vrátí hodnotu proměnné
- `putenv()`, `setenv()` – nastaví hodnotu proměnné

Proměnné prostředí

Slouží k nastavení chování aplikace.

Z příkazové řádky:

```
export MYVAR=ahoj
echo $MYVAR
```

Z aplikace:

- `getenv()` – vrátí hodnotu proměnné
- `putenv()`, `setenv()` – nastaví hodnotu proměnné

Užitečné proměnné prostředí

- `$LD_LIBRARY_PATH`
- `$PATH`
- `$RANDOM`

Proměnné prostředí – bezpečnost

Nespoléhejte se na proměnné prostředí (PATH, IFS, HOME, ...).

```
extern char** environ;
```

Dezinfekce sady proměnných prostředí

- vytvoření vlastního pole `environ`
- nastavení proměnných prostředí na bezpečné hodnoty
- převzetí **vybraných** původních hodnot

úkol

- Napište si vlastní verzi aplikace `printenv(1)`
- Za domácí úkol včetně všech parametrů

Konfigurační soubory

Obvyklé umístění v `/etc/`, ale také `/usr/share/`.

Neexistuje jednotný formát konfiguračních souborů.

- proměnná *hodnota* (sshd)
- XML (D-Bus)
- vlastní syntax (Apache)

Konfigurační soubory

Obvyklé umístění v `/etc/`, ale také `/usr/share/`.

Neexistuje jednotný formát konfiguračních souborů.

- proměnná *hodnota* (sshd)
- XML (D-Bus)
- vlastní syntax (Apache)

→ Snažte se dodržovat obvyklé konvence pro typ aplikace, kterou vytváříte.

Opravy kódu

Standardní utility `diff` a `patch`

Opravy kódu

Standardní utility `diff` a `patch`

Postup vytvoření a aplikace patche

- vytvořte kopii originálu (celou adresářovou strukturu, konkrétní soubor)
- upravte kód
- `diff -ur file.orig file > file.patch`
- `patch -p0 -i file.patch`

úkol

- Ve studijních materiálech je soubor `param.c`
- Najděte v kódu chybu a připravte patch.

Dynamicky linkované objekty

dynamické knihovny, kód přidávaný za běhu (pluginy)

API vs. ABI (I)

API – Application Programming Interface

(vysokoúrovňové) rozhraní mezi zdrojovým kódem a knihovnami – zdrojový kód lze zkompileovat

ABI – Application Binary Interface

(nízkoúrovňový) popis toho, jak jsou data uložena v paměti, tento popis používá compiler např. při referenci proměnných, umožňuje již zkompilevanému kódu běžet v prostředí s kompatibilním ABI

API i ABI měňte co nejméně často (raději funkce přidávejte).

Rozhodně rozumně verzujte při změnách!

Změny v ABI jsou nebezpečnější – nemusí být hned vidět a přijdete na ně až když něco nefunguje.

API vs. ABI (II)

Příklady

- reimplementace funkce –

API vs. ABI (II)

Příklady

- reimplementace funkce – API, ABI kompatibilní
- změna hodnoty položky enumu –

API vs. ABI (II)

Příklady

- reimplementace funkce – API, ABI kompatibilní
- změna hodnoty položky enumu – API kompatibilní, ABI nekompatibilní
- změna pořadí položek struktury –

API vs. ABI (II)

Příklady

- reimplementace funkce – API, ABI kompatibilní
- změna hodnoty položky enumu – API kompatibilní, ABI nekompatibilní
- změna pořadí položek struktury – API kompatibilní, ABI nekompatibilní
- odstranění funkce –

API vs. ABI (II)

Příklady

- reimplementace funkce – API, ABI kompatibilní
- změna hodnoty položky enumu – API kompatibilní, ABI nekompatibilní
- změna pořadí položek struktury – API kompatibilní, ABI nekompatibilní
- odstranění funkce – API nekompatibilní
- přidání nové funkce –

API vs. ABI (II)

Příklady

- reimplementace funkce – API, ABI kompatibilní
- změna hodnoty položky enumu – API kompatibilní, ABI nekompatibilní
- změna pořadí položek struktury – API kompatibilní, ABI nekompatibilní
- odstranění funkce – API nekompatibilní
- přidání nové funkce – API **zpětně** kompatibilní

API vs. ABI (II)

Příklady

- reimplementace funkce – API, ABI kompatibilní
- změna hodnoty položky enumu – API kompatibilní, ABI nekompatibilní
- změna pořadí položek struktury – API kompatibilní, ABI nekompatibilní
- odstranění funkce – API nekompatibilní
- přidání nové funkce – API **zpětně** kompatibilní

nekompatibilní API – aplikaci je třeba upravit

nekompatibilní ABI – aplikaci je třeba pouze překompilovat

Dynamické (sdílené) knihovny

- kolekce kódů dynamicky připojitelných k aplikaci za běhu
- připojitelných i k několika aplikacím najednou
- způsob jak předcházet duplikaci kódu
- šetří diskové místo :)
- snadná aktualizace a oprava chyb
- program je pak ale závislý na požadovaných knihovnách

Dynamické knihovny v Linuxu I

ldconfig(8)

- vytvoří symbolické odkazy na instalované knihovny – linker většinou použije právě tyto symlinky
- vytvoří cache instalovaných knihoven pro dynamický linker
- potřebné soubory včetně informace o umístění knihoven jsou v `/etc/ld.so.*`
- obvyklé umístění knihoven je
`/lib/,/lib64/,/usr/lib/,/usr/lib64/, ...`

proměnné prostředí

`LD_LIBRARY_PATH` – umístění knihoven (před standardními cestami)

`LD_PRELOAD` – knihovna, která má být přilinkována jako první

`LD_DEBUG` – `LD_DEBUG=libs date`

Dynamické knihovny v Linuxu II

- soname: 'lib' + jméno + '.so.' + verze
- /lib/ld-linux.so.X – run-time linker (loader)
- ldd(1) – zjistí závislosti dynamických objektů
- objdump(1) – vypíše informace o objektových souborech

Vytváření dynamické knihovny

- vytvořte objekty s kódem nezávislým na pozici (PIC)

```
gcc -fPIC -c file.c
```

- z objektů vytvořte dynamickou knihovnu

```
gcc -shared -Wl,-soname,libmyname.so.1 \  
-o libmyname.so.1.0.0 file.o
```

úkol

- napište vlastní verzi funkce `localtime()` - je na vás, jaký čas bude vracet
- vyzkoušejte si `LD_PRELOAD` s vaší verzí `localtime()` na programu `date`

Pluginy (dynamicky linkovaný kód)

Otevření dynamického objektu za běhu aplikace.

- `dlopen()`
- `dlsym()`
- `dlerror()`
- `dlclose()`

Při překladu (`gcc`) je nutné použít přepínač `-ldl`.

Ukazatel na funkci:

```
návratový_typ (*identifikátor)(seznam_typů_parametrů);
```

Pluginy (dynamicky linkovaný kód)

Otevření dynamického objektu za běhu aplikace.

- `dlopen()`
- `dlsym()`
- `dlerror()`
- `dlclose()`

Při překladu (`gcc`) je nutné použít přepínač `-ldl`.

Ukazatel na funkci:

```
návratový_typ (*identifikátor)(seznam_typů_parametrů);
```

```
void* (*funkce)(void* ukazatel, size_t velikost);
```

Pluginy (dynamicky linkovaný kód)

Otevření dynamického objektu za běhu aplikace.

- `dlopen()`
- `dlsym()`
- `dlerror()`
- `dlclose()`

Při překladu (`gcc`) je nutné použít přepínač `-ldl`.

Ukazatel na funkci:

```
návratový_typ (*identifikátor)(seznam_typů_parametrů);
```

```
void* (*funkce)(void* ukazatel, size_t velikost);
```

<http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html>

Závěr

domácí úkoly a zdroje

Domácí úkol (rozšíření úkolu z hodiny)

- Využijte princip pluginu a napište 2 knihovny, které různě implementují podobnou funkcionalitu (např. tisknou různé zprávy)
- Vytvořte program, který volá jednotlivé funkce podle přání uživatele – při startu programu podle parametru příkazové řádky nebo proměnné prostředí (priorita je na vás).
- Program musí uživateli poskytnout možnost vypsat zprávu (použít funkce) opakovaně (včetně změny použité funkce) – způsob interakce s uživatelem je na vás.
- Nezapomeňte na srozumitelnou nápovědu a ošetření chyb!
- Vše uložte do SVN.

Zdroje

parametry příkazové řádky

www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap12.html

www.gnu.org/prep/standards/standards.html#Command_002dLine-Interfaces

www.gnu.org/prep/standards/standards.html#Option-Table

dynamicky linkované objekty

tldp.org/HOWTO/Program-Library-HOWTO/index.html

www.gnu.org/prep/standards/standards.html#Command_002dLine-Interfaces

www.gnu.org/prep/standards/standards.html#Option-Table

ostatní

www.di.uniovi.es/~cernuda/noprogramming_ENG.html

wezfulong.org/blog/2006/dec/coding-for-coders-api-and-abi-considerations-in-an-evolving-code-base