

# Základy vývoje 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

28. září 2015

## Zvyklosti vývoje v prostředí GNU/Linux

zvyklosti v chování aplikací, interakce aplikací s okolím

# Adresářová struktura GNU/Linux

Filesystem Hierarchy Standard [www.pathname.com/fhs/pub/fhs-2.3.html](http://www.pathname.com/fhs/pub/fhs-2.3.html)

- /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 filesystem 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>`

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

- `$HOME`
- `$USER`
- `$PATH`
- `$LD_LIBRARY_PATH`

# Proměnné prostředí – bezpečnost

Nespoléhejte se na proměnné prostředí (PATH, IFS, HOME, ...)  
– považujte je za nedůvěryhodný vstup.

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

## *Dezinfekce sady proměnných prostředí*

- převzetí **vybraných** původních hodnot
- smazání prostředí (`clearenv(3)`/`unsetenv(3)`)
- nastavení proměnných prostředí na bezpečné hodnoty

## úkol

- Napište si vlastní verzi aplikace `printenv(1)`



### úkol

- Napište si vlastní verzi aplikace `printenv(1)`
- Včetně všech parametrů (případně dokončit doma)

# 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 (zdrojové kódy)

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=help date`; `LD_DEBUG=libs date`

# Dynamické knihovny v Linuxu II

- soname: 'lib' + jméno + '.so.' + verze (libpam.so.0)
- /lib/ld-linux.so.X – run-time linker (loader)
- ldd(1) – zjistí závislosti dynamických objektů
- objdump(1), nm(1), readelf(1) – vypíše informace o objektových souborech (readelf -l /bin/ls)

## 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 (`libupper.so.1`, `liblower.so.1`), které různě implementují podobnou funkcionalitu: filtr řetězců.
- Vytvořte program, který volá funkce podle přání uživatele – po startu programu podle parametru příkazové řádky (`--filter=<lower|upper>`) a proměnné prostředí (`FILTER=<lower|upper>`).
- Program opakovaně načítá od uživatele vstup (text) a vypisuje výstup po aplikaci právě aktivního filtru.
- Program musí uživateli poskytnout možnost opakovaně měnit aktivní filtr/plugin: příkaz pro změnu bude ve formátu `>lower` resp. `>upper`.
- Nezapomeňte na srozumitelnou nápovědu a ošetření chyb – pozor na `dlsym()`!



# Zdroje

## **implicitně – manuálové stránky**

### **parametry příkazové řádky**

[pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap12.html](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap12.html)

[www.gnu.org/prep/standards/standards.html#Command\\_002dLine-Interfaces](http://www.gnu.org/prep/standards/standards.html#Command_002dLine-Interfaces)

### **proměnné prostředí**

[etutorials.org/Programming/secure+programming/Chapter+1.+Safe+Initialization/1.1+Sanitizing+the+Environment/](http://etutorials.org/Programming/secure+programming/Chapter+1.+Safe+Initialization/1.1+Sanitizing+the+Environment/)

[www.securecoding.cert.org/confluence/display/c/ENV03-C.+Sanitize+the+environment+when+invoking+external+programs](http://www.securecoding.cert.org/confluence/display/c/ENV03-C.+Sanitize+the+environment+when+invoking+external+programs)

### **dynamicky linkované objekty**

[tldp.org/HOWTO/Program-Library-HOWTO/index.html](http://tldp.org/HOWTO/Program-Library-HOWTO/index.html)

### **ostatní**

[www.di.uniovi.es/~cernuda/noprogramming.html](http://www.di.uniovi.es/~cernuda/noprogramming.html)

[wezfulong.org/blog/2006/dec/coding-for-coders-api-and-abi-considerations-in-an-evolving-code-base](http://wezfulong.org/blog/2006/dec/coding-for-coders-api-and-abi-considerations-in-an-evolving-code-base)