Cíl cvičení

Cílem tohoto cvičení je prakticky si procvičit informace probírané na 2. přednášce a motivovat vás ke 4. přednášce "*Linuxové jádro – vznik, vývoj, skladba a ovladače; GNU libc a uživatelský prostor*". Pokud vám nejsou jasné některé souvislosti, doporučujeme zkusit si informace vygooglovat a připravit si otázky na 4. přednášku.

Open source software (OSS) projekty se často nepoužívají osamoceně, ale v kombinaci s jinými OSS projekty, čímž vznikají tak tzv. OSS stacky. Asi nejznámějším stackem je LAMP – Linux, Apache, MySQL, PHP. Na dnešním cvičení se seznámíme s dalším, velmi často používaným, stackem Linux + BusyBox (+Dropbear SSH server).

BusyBox je sada UNIXových uživatelských nástrojů (shell, editor, utility jako ls, mkdir, …) zkompilovaná do jedné binárky. V kombinaci s jádrem Linuxu tak dostáváme kompletní operační systém s poměrně malými nároky na paměť. Díky tomu se tato kombinace často používá ve vestavěných (embedded) aplikacích jako například WiFi routery či ADSL modemy.

V tomto cvičení si zkusíte vytvořit kompletní open source operační Systém z Linuxového jádra a uživatelského prostředí tvořeného právě BusyBoxem. Dále si vyzkoušíte naprogramovat jednoduchý modul do jádra.

Postup

1. Stáhněte si zdrojové kódy z git repozitáře projektu BusyBox (adresu repozitáře si najděte sami).

   ```
   git clone ...
   cd busybox
   ```

   Protože se jedná o aktuální vývojový snapshot, je možné, že se při překladu, instalaci nebo používání vyskytne chyba. Naší výhodou však je, že máme k dispozici kompletní historii projektu

   ```
   gitk
   ```

   a můžeme si vybrat verzi, kde se chyba nevyskytuje. Například

   ```
   git checkout -f 1_22_0
   ```

2. Zkonfigurujeme jak chceme BusyBox přeložit.

   ```
   make menuconfig
   ```

Vystačíme s výchozí konfigurací, takže zvolte `Exit` a odpovězte `Yes`, že chcete konfiguraci uložit.

3. Kompilaci provedeme tradičně příkazem

```
make
```

4. Příkazem

```
make install
```

nainstalujeme busybox do adresáře `./_install`. Všimněte si (`ls -lR _install`), že se tam nachází pouze jedna binárka `bin/busybox` a všechno ostatní jsou pouze symbolické odkazy na tuto binárku.

Protože neprovádíme tzv. *křížový překlad* (cross compilation), který je běžný v případě vestavěných zařízení, můžeme výsledek hned otestovat například spuštěním shellu: `./_install/bin/sh` (ukončíme ho např. příkazem `exit`).

V případě skutečného vestavěného systému bychom museli pokračovat dál a busybox otestovat až po nabootování na cílovém hardwaru.

5. Pokud máte na svém počítači práva uživatele `root`, můžete otestovat BusyBox v chroot prostředí, t.j. se stejným jádrem jako právě běží na vašem počítači, ale se souborovým systémem tvořeným pouze BusyBoxem:

```
# chroot _install /bin/sh
```

Fungovat to ale nebude, protože ke spuštění BusyBoxu jsou potřeba knihovny, které v nejsou v adresáři `_install` dostupné.

6. Knihovny, které `busybox` potřebuje ke svému běhu, zjistíte příkazem

```
ldd _install/bin/busybox
```

Na 32-bitovém systému to může vypadat například takto:

```
linux-gate.so.1 =>   (0xffffe000)
libm.so.6 => /lib/i686/cmov/libm.so.6 (0xf777c000)
libc.so.6 => /lib/i686/cmov/libc.so.6 (0xf7635000)
/lib/ld-linux.so.2 (0xf77aa000)
```

`linux-gate.so.1` (nebo `linux-vdso.so.1`) je virtuální knihovna, kterou poskytuje jádro Linuxu, takže o tu se starat nemusíte. Ostatní knihovny je potřeba nakopírovat do cílového systému (adresáře `_install`) například následujícím způsobem:

```
mkdir _install/lib
cp /lib/i686/cmov/libm.so.6 /lib/i686/cmov/libc.so.6 /lib/ld-linux.so.2 _install
```

Nezapomeňte, že v cílovém filesystému musí být i tzv. interpretr (`ld-linux`), který je na **64-bitovém systému v adresáři `/lib64`**.

Nyní už můžete spustit BusyBox v chroot prostředí.

7. Nejjednodušší možnost jak nabootovat do právě vytvořeného uživatelského prostředí je uložit ho ve formátu pro Linuxový startovací RAM-disk a nabootovat Linuxové jádro s tímto RAM-diskem.

   Aby vše fungovalo jak má, kromě souborů v adresáři `_install` musí RAM-disk obsahovat i několik položek v adresáři `/dev` pro přístup k virtuálním terminálům. V závislosti na vašem oprávnění můžete RAM-disk vytvořit jedním z následujících způsobů:

   a. Pokud máte *rootovská* práva, použijte ke tvorbě RAM-disku následující příkazy:

   ```
   mkdir _install/{dev,etc,proc,sys}
   sudo cp -a /dev/tty? _install/dev
   ln -s bin/busybox _install/init
   (cd _install; find . | cpio -o -H newc | gzip) > ramdisk
   ```

   b. Bez rootovských práv můžete RAM-disk vytvořit pomocí nástroje gen_init_cpio. Pokud program nemáte na svém počítači, zkompilujte si ho ze zdrojových kódů.

   ```
   (
   cat <<EOF
   dir /dev 755 0 0
   nod /dev/tty0 644 0 0 c 4 0
   nod /dev/tty1 644 0 0 c 4 1
   nod /dev/tty2 644 0 0 c 4 2
   nod /dev/tty3 644 0 0 c 4 3
   nod /dev/tty4 644 0 0 c 4 4
   slink /init bin/busybox 700 0 0
   dir /proc 755 0 0
   dir /sys 755 0 0
   EOF

   find _install -mindepth 1 -type d -printf "dir /%P %m 0 0\n"
   find _install -type f -printf "file /%P %p %m 0 0\n"
   find _install -type l -printf "slink /%P %l %m 0 0\n"
   ) > filelist

   gen_init_cpio filelist | gzip > ramdisk
   ```

   Výše uvedené příkazy fungují následovně: Nejprve vytvoříme seznam souborů (`filelist`), které má ramdisk obsahovat. Nástroj `gen_init_cpio` pak podle toho seznamu vytvoří obraz ramdisku, který “zazipujeme” příkazem `gzip` a uložíme do souboru `ramdisk`.

8. **Jádro Linuxu**. Příprava jádra je téměř stejná jako u BusyBoxu: stáhneme zdrojový kód, nakonfigurujeme a přeložíme. Vzhledem k rozsáhlosti jádra (zkompilované zabere na disku cca 1 GB a překlad trvá cca 20 minut) tyto kroky přeskočíme a použijeme již připravené jádro z distribuce.

   Pokud byste si chtěli přeložit vlastní jádro, v následujícím příkazu byste za parametrem *-kernel* uvedli cestu k vámi zkompilovanému jádru.

9. Bootování jádra s naším filesystémem (v softwarovém emulátoru systému):

   Na 64-bitovém systému spustíme emulátor následovně:

   ```
   qemu-system-x86_64 -kernel /boot/vmlinuz-3.2.0-4-amd64 -initrd ramdisk
   ```

Na 32-bitovém systému použijte:

```
qemu-system-i386 -kernel /boot/vmlinuz-2.6.26-2-686 -initrd ramdisk
```

Pokud systém nabízí hardwarovou podporu virtualizace, je výhodné použít virtualizační nástroj KVM. Výsledek pak běží rychleji. KVM vychází z qemu a má tudíž téměř stejné parametry příkazové řádky. Například:

```
kvm -kernel /boot/vmlinuz-3.2.0-4-amd64 -initrd ramdisk
```

10. Pokud vše proběhlo správně, zobrazila se hláška

```
Please press Enter to activate this console.
```

Po stisku Enteru se spustí shell a můžete začít pracovat ve vašem právě vytvořeném systému.

Možná vylepšení

Dále můžete provést drobná (či větší) vylepšení svého nového systému, která vám mohou zjednodušit další práci.

1. Můžete připojit souborový systém `/proc`, aby fungovaly příkazy jako např. `ps` (výpis běžících procesů). Příkaz spusťte v emulátoru, ne na vaší pracovní stanici.

```
mount -t proc none /proc
```

2. V RAM-disku můžete vytvořit soubor `/etc/init.d/rcS`, který bude obsahovat příkazy, které budou spuštěny při bootu systému.

```
mkdir -p _install/etc/init.d
cat <<EOF > _install/etc/init.d/rcS
#!/bin/sh
mount -t proc none /proc
echo Nazdar!!!!
EOF
chmod +x _install/etc/init.d/rcS    # nastavení spustitelnosti
```

Nyní musíte znovu vytvořit RAM-disk a nabootovat.

3. Zachytávání zpráv jádra spuštěného v emulátoru QEMU do souboru. Zprávy jádra je možné přesměrovat na virtuální sériový port a odtamtud pak například na standardní výstup:

```
qemu -serial stdio ...
```

a jádru předáme parametr `console=ttyS0`

```
qemu -serial stdio -append console=ttyS0 ...
```

4. Pokud chcete z vašeho systému komunikovat po síti, připojte ho na vnější síť pomocí NAT na uživatelské úrovni:

```
qemu -net nic,vlan=0,model=ne2k_pci -net user,vlan=0 ...
```

5. Pokud QEMU nebo KVM podporuje vytvoření virtio sítě Plan9 a virtuálního souborového systému, tak je možné propagovat do vnitřního systému obsah adresáře hostitelského systému:

   ```
   qemu -virtfs local,path=shared_dir_name,security_model=none,mount_tag=shared_tag
   ```

   Adresářovou strukturu lze z vnitřního systému připojit následujícími příkazy

   ```
   modprobe virtio
   modprobe virtio_ring
   modprobe virtio_pci
   modprobe 9pnet
   modprobe 9pnet_virtio
   modprobe 9p
   mkdir -p /mnt/shareddir
   mount -t 9p -o trans=virtio shared_tag /mnt/shareddir
   ```

Další tipy a triky v oblasti virtualizace používané a odzkoušené zprávci sítě na naší katedře nalezente na Wiki Technické Podpory (support).

Jaderné moduly

Jaderné moduly jsou přeložené kusy kódu, které lze za běhu nahrávat do Linuxového jádra. Pokud bychom chtěli nalézt analogickou věc v uživatelském prostředí, pak by to byly *sdílené knihovny*. Jaderný modul může obsahovat kód ovladače zařízení, podporu určitého souborového systému, může přidávat do jádra nové funkce (např. firewall) či sloužit jako knihovna pomocných funkcí pro jiné moduly (např. libata).

Zdrojový kód jednoduchého jaderného modulu vypadá následovně:

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}

static void hello_exit(void)
{
    printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Příklad převzat z LDD3.

Překlad modulu provedeme pomocí jednoduchého souboru Makefile, který bude obsahovat jedinou řádku (zde předpokládáme, že výše uvedený soubor se jmenuje **khello.c**):

```
obj-m = khello.o
```

Nyní stačí zavolat `make` se správnými parametry:

```
make -C /lib/modules/$(uname -r)/build M=$(pwd) modules
```

Tímto říkáme, že příkaz `make` načte `Makefile` z adresáře se zdrojovými kódy aktuálně běžícího jádra (o který adresář se jedná můžete zjisti pomocí `readlink -f /lib/modules/$(uname -r)/build`), pomocí proměnné `M` řeknete, že se váš modul nachází v aktuálním adresáři a slovo `modules` na konci znamená, že chcete, aby se zkompilovaly pouze moduly.

Pokud vše dopadlo dobře, objevil se vám soubor `khello.ko`, což je modul, který můžete zavést do jádra příkazem

```
insmod khello.ko
```

Zadání

1. Stáhněte si tento program (32-bitová verze) a zprovozněte ho ve vámi vytvořeném systému. Zprovoznění znamená, že po spuštění program nevypíše žádnou chybu. Nezapomeňte nastavit práva pro spouštění příkazem `chmod +x`.

   Ke zjištění případným problémů by se vám mohly hodit příkazy `strace` a `ltrace`. Ten prvý vypisuje všechna systémová volání vyvolaná daným programem a druhý vypisuje jaké funkce ze sdílených knihoven program volá. Zkuste si například spustit následující příkazy:

   ```
   echo Ahoj
   strace echo Ahoj
   ltrace echo Ahoj
   ```

2. Vytvořte jednoduchý jaderný modul, který po zavedení do jádra vypíše vaše jméno (objeví se ve výstupu příkazu `dmesg`). Jinak nemusí dělat nic. Předveďte činnost vašeho modulu ve vámi vytvořeném systému běžícím v emulátoru.

   Kdo se bude nudit, může zkusit rozšířit modul tak, aby se jeho jméno objevilo v souboru `/proc/myname` nebo vytvořit jednoduchý ovladač, který bude vracet vaše jméno při čtení z `/dev/myname`. Návod najdete v tomto článku (strany 2 a 3).

Tipy a triky

- Pro vyvolání určitého příkazu z historie můžete použít klávesu `Ctrl-R` následovanou textem hledaného příkazu. Např:

  ```
  <Ctrl-R>cpio<Enter>
  ```

- Pro rychlé kopírování textu mezi programy (např. příkazy z této stránky do shellu), můžete použít prostřední tlačítko myši. Funguje to tak, že text označíte myší (nemačkáte při tom Ctrl-C) a stiskem prostředního tlačítka myši v terminálovém okně ho vložíte na příkazovou řádku shellu. Tím, že při tom nemusíte šahat na klávesnici vám to půjde rychleji.

- Pokud je Qemu spouštěný přes vzdálené připojení (např. server postel), je potřeba pro zobrazení emulované obrazovky spouštěného stroje buď provést protunelování X protokolu (`ssh -X`) nebo používat Qemu s emulací obrazovky v textovém režimu `qemu -curses`. Další možnost je emulovat HW bez grafické karty `qemu -nographic` a nastavit

testovaný systém tak, aby systémová konzole směřovala na sériový port (`-append console=ttyS0`).

Reference

- ramfs, rootfs and initramfs
- Early userspace support
- Inside the Linux boot process
- The Linux Bootdisk HOWTO
- Linux Loadable Kernel Module HOWTO
- Linux Device Drivers, Third Edition
- Kbuild & modules
- /dev/hello_world: A Simple Introduction to Device Drivers under Linux
- Rostislav Lisový: Diplomová práce - Rozšíření QEMU o podporu PCI I/O karty a tvorba ovladačů
- Filip Navara: Diplomová práce - Rozšíření QEMU o podporu mikrokontroléru AT91SAM7X256
- Aleš Kapica a další: Stránky o virtualizaci z oddělení IT K13135

Static binary

Trable s ldd si lze odpustit a prelozit busybox jako statickou binarku…

```
Busybox Settings  --->
  Build Options  --->
    [*] Build BusyBox as a static binary (no shared libs)
```

*Comment by **kufnejos** — 11 years and 11 months ago*

Re: Static binary

Toto řešení má smysl a používá se v některých specifických případech. Pokud však chcete později systém doplnit o nějakou uživatelskou aplikaci (např. e-book reader) tak statické linkování více binárních programů vede k velkému nárůstu zabraného místa. Přitom postup s `ldd` vám snadno umožňuje image doplnit o `bash`, `mc` a cokoliv dalšího a to v běžných případech bez rekompilace. Je také možné načíst ovladače FS, připojit například oddíl z disku (např. při řešení chyby boot sektoru) a provozovat aplikace odtud. Je tedy dobré si vyzkoušet, jak běžné závislosti vypadají a jak je zjistit a splnit.

*Comment by **pisa** — 11 years and 11 months ago*

64bitovy system

Na skolnich pocitacich je dulezite, aby byl v cilovem filesystemu (ramdisk) vytvoren adresar /lib64, do ktereho musi byt zkopirovana ona knihovna ld-linux.

*Comment by **Anonym** — 8 years and 11 months ago*

No available video device

postel:~/busybox$ qemu-system-x86_64 -kernel /boot/vmlinuz-3.2.0-4-amd64 -initrd ramdisk

—> Could not initialize SDL(No available video device) - exiting

☹ prosím o radu, jel jsem přesně podle návodu

*Comment by **Anonym** — 8 years and 10 months ago*

Re: No available video device

Spusťte qemu s parameterm `-nographic`. Viz sekce Tipy a triky.

*Comment by **sojkam1** — 8 years and 10 months ago*

fatal error: curses.h

Pokud na Ubuntu narazíte na *fatal error: curses.h*, bude potřeba

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

viz http://ubuntuforums.org/showthread.php?t=2036205

*Comment by **stankmic** — 6 years and 10 months ago*

comment 7

Jestli pri zpusteni prikazu „**insmod khello.ko**„ vam se objevuje chyba „**no symbol version for module_layout**„, pouzijte reseni z nasledujiciho odkazu: http://askubuntu.com/a/171633

*Comment by **sokolnik** — 6 years and 10 months ago*

Last edited 6 years and 10 months ago

Webmaster: sojkam1 at fel cvut cz