Building the Kernel

Advanced Operating Systems and Virtualization

Alessandro Pellegrini

A.Y. 2019/2020

How to Compile the Kernel

```
$ make menuconfig / nconfig / xconfig /
  gconfig
$ make -jX
$ make modules
# make modules_install
# make install
# make headers_install
# <build an init ramdisk>
# grub-mkconfig -o /boot/grub/grub.cfg
```

initrd

- A way to load a temporary root file system into memory
- This mainly hosts kernel modules to be loaded during the boot to interact with the specific hardware, and early-life applications
- This file system is then replaced by the actual root file system, moved to a folder, and unmounted
- Typically it is an archive in the CPIO format
  - `lsinitcpio /boot/initramfs-linux.img`

Manually Create an `initrd`

- `mkdir -p init/initramfs/{bin,sbin,etc,proc,sys,newroot}`
- `cd init`
- `touch initramfs/etc/mdev.conf`
- Copy in the archive all tools which respect the Single UNIX Specification (e.g., BusyBox)
- Provide an implementation of `/init`

Manually Create an `initrd`

```
#!/bin/sh

mount -t proc proc /proc
mount -t sysfs sysfs /sys

echo 0 > /proc/sys/kernel/printk
clear

busybox --install -s

mknod /dev/null c 1 3
mknod /dev/tty c 5 0
mdev -s

# You can parse /proc/cmdline
# to get actual values
init="/sbin/init"
root="/dev/hda1"

mount "${root}" /newroot

if [[ -x "/newroot/${init}" ]] ;
then
        umount /sys /proc
        exec switch_root /newroot
        "${init}"
fi

echo "Failed to switch_root,
dropping to a shell"
exec sh
```

switch_root

NAME

switch_root - switch to another filesystem as the root of the mount tree

SYNOPSIS

switch_root newroot init [arg...]

DESCRIPTION

switch_root moves already mounted /proc, /dev, /sys and /run to newroot and makes newroot the new root filesystem and starts init process.

WARNING: switch_root removes recursively all files and directories on the current root filesystem.

switch_root

- You cannot actually unmount /
- `mount(newroot, "/", NULL, MS_MOVE, NULL)`

- If mountflags contains the flag `MS_MOVE` (available since Linux 2.4.18), then move a subtree: `source` specifies an existing mount point and `target` specifies the new location to which that mount point is to be relocated. The move is atomic: at no point is the subtree unmounted.

Kernel Build System

- **Kconfig files**: define each config symbol and its attributes, such as its type, description and dependencies. Programs that generate an option menu tree (for example, make menuconfig) read the menu entries from these files.

- **.config file**: stores each config symbol's selected value. You can edit this file manually or use one of the many make configuration targets, such as menuconfig and xconfig, that call specialized programs to build a tree-like menu and automatically update (and create) the .config file for you.

Kernel Build System

- **Upper-Level Makefiles**: normal GNU makefiles that describe the relationship between source files and the commands needed to generate each make target, such as kernel images and modules.

- **Kbuild Files**: a "special flavor" of Makefiles used by the kernel, to instruct how to build subsystems

Kconfig Example

```
menu "Character devices"

config DUMMY_CHAR
       tristate "DummyChar device support"
       ---help---
         Say Y here if you want to add support for the
         DummyChar device.
         If unsure, say N.
         To compile this driver as a module, choose M here:
         the module will be called dummy.

config DUMMY_STAT
        bool "dummy statistics"
        depends on DUMMY_CHAR
       ---help---
        Say Y here if you want to enable statistics about
        the DummyChar device.
```

Kbuild Files

- A Kbuild file is named "Makefile" or "Kbuild"
- **Goal definitions**: they define the files to be built, any special compilation options, and any subdirectories to be entered recursively.

```
obj-y += foo.o
```

Built from `foo.c` or `foo.S`

Compile as a module

```
obj-m += foo.o
```

It can be also a directory

```
obj-$(CONFIG_FOO) += foo.o
```

Kbuild for `obj-y`

- The kbuild Makefile specifies object files for `vmlinux` in the `$(obj-y)` lists.
- Kbuild compiles all the `$(obj-y)` files.
- It then calls "`$(LD) -r`" to merge these files into one `built-in.o` file

Final Kernel Linking

- The final linking is carried out via the `link-vmlinux.sh` script
- vmlinux is linked from the objects selected by `$(KBUILD_VMLINUX_INIT)` and `$(KBUILD_VMLINUX_MAIN)`
- Order is important! `$(KBUILD_VMLINUX_INIT)` must come first
- This is the point where `kallsyms` is placed in the kernel image

Kernel System Map

- It contains a mapping between symbols and virtual memory locations (determined at compile/link time) for:
  - Steady-state Kernel functions (steady-state ones)
  - Kernel data structures
- Symbols are associated with ‘storage class’:
  - `T`: global (non-static but not necessarily exported) function;
  - `t`: a function local to the compilation unit (i.e. static)
  - `D`: global data;
  - `d`: data local to the compilation unit.
  - `R`/`r`: same as `D`/`d`, but for read-only data

System map applications

- Kernel debugging
- Kernel run-time hacking
- The system map is also (partially) reported by the (pseudo) file `/proc/kallsysm`

- This is access-protected from non-root users:
  - Kernel printing function uses the %pK format specifier
  - This checks whether the current command is being run by root
  - In the negative case, it returns zero

An example

2.6.5-7.282-smp #1 SMP ……. i686 i686 i386 GNU/Linux

c03a8a00 D sys_call_table

Read/write data

2.6.32-5-amd64 #1 SMP …… x86_64 GNU/Linux

ffffffff81308240 R sys_call_table

Read-only data