modules_install & install

上一篇文章介绍了 make 的所有过程和产物，即 vmlinux， bzImage 和各 modules 文件，那下一步要做的就是安装你编译的内核。安装内核自然就分为安装内核 image 和各 module，这二者分别由 `make install` 和 `make modules_install` 来完成。

make modules_install

因为 modules_install 的依赖关系比较简单，直接看代码：

```
# In top Makefile
MODLIB  = $(INSTALL_MOD_PATH)/lib/modules/$(KERNELRELEASE)
# 一般情况下，MODLIB 等于 /lib/modules/$(KERNELRELEASE)

modules_install: _modinst_ _modinst_post

_modinst_:
        @rm -rf $(MODLIB)/kernel
        @rm -f $(MODLIB)/source
        @mkdir -p $(MODLIB)/kernel
        @ln -s $(abspath $(srctree)) $(MODLIB)/source
        @if [ ! $(objtree) -ef  $(MODLIB)/build ]; then \
                rm -f $(MODLIB)/build ; \
                ln -s $(CURDIR) $(MODLIB)/build ; \
        fi
        @cp -f $(objtree)/modules.order $(MODLIB)/
        @cp -f $(objtree)/modules.builtin $(MODLIB)/
        $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modinst

# This depmod is only for convenience to give the initial boot a modules.dep even before
# / is mounted read-write.  However the boot script depmod is the master version.
_modinst_post: _modinst_
        $(call cmd,depmod)

# Run depmod only if we have System.map and depmod is executable
cmd_depmod = $(CONFIG_SHELL) $(srctree)/scripts/depmod.sh $(DEPMOD) \
             $(KERNELRELEASE) "$(patsubst y,_,$(CONFIG_HAVE_UNDERSCORE_SYMBOL_PREFIX))"
```

由代码可知，*modinst* 的重点工作转移到 scripts/Makefile.modinst，此 Makefile 的内容仅仅 40 行，也很简单，所以我们看一下重点：

```
# 第一条 rule, default goal.
__modinst:

__modinst: $(modules)
        @:

# 依然是从 .tmp_versions 目录下的 .mod 文件中获知所有 module 的列表，然后经过 wildcard 处理剔除不存在的 .ko
__modules := $(sort $(shell grep -h '\.ko$$' /dev/null $(wildcard $(MODVERDIR)/*.mod)))
modules := $(patsubst %.o,%.ko,$(wildcard $(__modules:.ko=.o)))

cmd_modules_install = \
    mkdir -p $(2) ; \
    cp $@ $(2) ; \
    $(mod_strip_cmd) $(2)/$(notdir $@) ; \
    $(mod_sign_cmd) $(2)/$(notdir $@) $(patsubst %,|| true,$(KBUILD_EXTMOD)) && \
    $(mod_compress_cmd) $(2)/$(notdir $@)

ext-mod-dir = $(INSTALL_MOD_DIR)$(subst $(patsubst %/,%,$(KBUILD_EXTMOD)),,$(@D))
modinst_dir = $(if $(KBUILD_EXTMOD),$(ext-mod-dir),kernel/$(@D))
$(modules):
        $(call cmd,modules_install,$(MODLIB)/$(modinst_dir))
```

看起来也很简单，即在 $(MODLIB) 目录下创建 kernel/bluhbluh 目录，然后把相应路径下的 .ko 拷贝过去。然后还可能这些处理：(mod_strip_cmd, mod_sign_cmd, mod_compress_cmd)，这些变量定义在 top Makefile 中，有条件执行，来看一眼：

```
# INSTALL_MOD_STRIP, if defined, will cause modules to be
# stripped after they are installed.  If INSTALL_MOD_STRIP is '1', then
# the default option --strip-debug will be used.  Otherwise,
# INSTALL_MOD_STRIP value will be used as the options to the strip command.
# 上面的注释可以看出，INSTALL_MOD_STRIP 是用户使用的命令行选项. strip 命令用于剥除 object file
# 中的各种符号，--strip-debug 用于剥除debug符号，目的是为了减小 .ko 的体积。
ifdef INSTALL_MOD_STRIP
    ifeq ($(INSTALL_MOD_STRIP),1)
        mod_strip_cmd = $(STRIP) --strip-debug
    else
        mod_strip_cmd = $(STRIP) $(INSTALL_MOD_STRIP)
    endif # INSTALL_MOD_STRIP=1
else
```

```
    mod_strip_cmd = true
endif # INSTALL_MOD_STRIP

# CONFIG_MODULE_SIG_ALL 是配置选项，在 make *config 时配置。为模块签名？尚未研究过，略过
ifdef CONFIG_MODULE_SIG_ALL
    $(eval $(call config_filename,MODULE_SIG_KEY))
    mod_sign_cmd = scripts/sign-file $(CONFIG_MODULE_SIG_HASH) $(MODULE_SIG_KEY_SRCPREFIX)      $(CONFIG_MODULE_SIG_KEY) certs/sign
ing_key.x509
else
    mod_sign_cmd = true
endif

# CONFIG_MODULE_COMPRESS, if defined, will cause module to be compressed
# after they are installed in agreement with CONFIG_MODULE_COMPRESS_GZIP
# or CONFIG_MODULE_COMPRESS_XZ.
# 上面的注释已经解释的很清楚了，就是把安装的 .ko 文件压缩一下。
mod_compress_cmd = true
ifdef CONFIG_MODULE_COMPRESS
    ifdef CONFIG_MODULE_COMPRESS_GZIP
        mod_compress_cmd = gzip -n -f
    endif # CONFIG_MODULE_COMPRESS_GZIP
    ifdef CONFIG_MODULE_COMPRESS_XZ
        mod_compress_cmd = xz -f
    endif # CONFIG_MODULE_COMPRESS_XZ
endif # CONFIG_MODULE_COMPRESS
```

Ok, branch 了那么远，*modinst* 做的事情了解清楚了，回头看看 modules_install 的第二个 prerequisite: _modinst_post

```
cmd_depmod = $(CONFIG_SHELL) $(srctree)/scripts/depmod.sh $(DEPMOD) \
             $(KERNELRELEASE) "$(patsubst y,_,$(CONFIG_HAVE_UNDERSCORE_SYMBOL_PREFIX))"
```

scripts/depmod.sh 是 depmod 命令的 wrapper。看一下 depmod 的 manual, 它做的事情看起来不少：生成 /lib/modules/version 目录下的 modules.dep, modules.dep.bin, modules.symbols, modules.symbols.bin, modules.devname 等几个文件。

make install

Target "install" 定义在 arch Makefile 中，以 x86 为例：

```
boot=arch/x86/boot
install:
        $(Q)$(MAKE) $(build)=$(boot) $@
```

又进入到 arch/x86/boot/Makefile:

```
install:
        sh $(srctree)/$(src)/install.sh $(KERNELRELEASE) $(obj)/bzImage \
                System.map "$(INSTALL_PATH)"

# 变量 KERNELRELEASE, INSTALL_PATH, INSTALLKERNEL 定义在 top Makefile 中
# INSTALL_PATH specifies where to place the updated kernel and system map
# images. Default is /boot, but you can set it to other values
export  INSTALL_PATH ?= /boot

INSTALLKERNEL  := installkernel
```

所以, x86 架构下执行 `make install` 时，最终执行的命令是：

```
arch/x86/boot/install.sh $(KERNELRELEASE) arch/x86/boot/bzImage System.map "$(INSTALL_PATH)"
```

进入 install.sh 里面看一眼，看起来也是很简单的一段 shell script。大部分情况下，如果系统中有 installkernel 的程序，则执行它来安装刚编译好的 kernel，传递给 install.sh 的参数原样传给系统中的 installkernel 脚本：

```
if [ -x /sbin/${INSTALLKERNEL} ]; then exec /sbin/${INSTALLKERNEL} "$@"; fi
```

installkernel 是位于 /sbin 目录下的一个 bash script。 在我的 Fedora Workstation 27 的环境中，/sbin 和 /usr/sbin 目录下有一模一样的 installkernel 脚本。

所以， `make install` 的过程是包了一层又一层。最后来看一眼 installkernel 脚本，也是很简单，做了各种环境检查后，最实质的动作包括这么几个：

i. 把 root source 目录下的 System.map 拷贝为 /boot/System.map-$KERNEL_VERSION 并在 /boot 目录下为它建立符号链接 /boot/System.map
ii. 把 arch/x86/boot/bzImage 拷贝为 /boot/vmlinuz-$KERNEL_VERSION 并在 /boot 目录下为它建立符号链接 /boot/vmlinuz
iii. 执行：
`new-kernel-pkg --mkinitrd --dracut --host-only --depmod --install --kernel-name $KERNEL_NAME $KERNEL_VERSION`
`new-kernel-pkg --rpmposttrans --kernel-name $KERNEL_NAME $KERNEL_VERSION`

> new-kernel-pkg is a tool used in packaging to automate the installation of a new kernel, including the creation of an initial ram filesystem image, updating of bootloader configuration, and other associated tasks.

new-kernel-pkg 是个脚本，请参考它 的 manual 来理解 iii. 中的命令行具体做了什么，如果想了解的更清楚，就只能阅读它的代码了。简单扫了一眼发现，此脚本中还会调用其他的命令，一股深似海的感觉扑面而来:)

modules_install 和 install 的过程就是酱紫咯～