<http://blog.chinaunix.net/uid-20543672-id-3018947.html>

生成的zImage内核的位置在arch/arm/boot目录下。

内核根目录下的vmlinux映像文件是内核Makefile的默认目标。这个vmlinux映像的生成可以通过阅读内核Makefile文件得知，简单的说：Makefile解析内核配置文件.config，递归到各目录下编译出.o文件，最后将其链接成vmlinux。而这个链接成的vmlinux文件是一个包含内核代码的静态可执行ELF文件，你可以通过file命令来验证这一点。她不能通过bootloader引导并启动，如果想要使其可引导，必须使用编译工具链中的objcopy命令把这个ELF格式的vmlinux转化为二进制格式才行。

而平常使用的zImage文件就是这个vmlinux文件经过多次的转换得到的。现在就来仔细研究一下她的生成过程。

### arch/$(ARCH)/Makefile

首先嵌入式中经常使用的编译目标zImage并不在顶层Makefile文件中，而在被顶层Makefile包含的arch/$(ARCH)/Makefile文件中，对于ARM处理器来说就是arch/arm/Makefile文件。其中的部分规则如下：

……

# Default target when executing plain make

ifeq ($(CONFIG\_XIP\_KERNEL),y)

KBUILD\_IMAGE := xipImage

else

KBUILD\_IMAGE := zImage

endif

all: $(KBUILD\_IMAGE)

boot := arch/arm/boot

archprepare:

$(Q)$(MAKE) $(build)=arch/arm/tools include/generated/mach-types.h

# Convert bzImage to zImage

bzImage: zImage

zImage Image xipImage bootpImage uImage: vmlinux

$(Q)$(MAKE) $(build)=$(boot) MACHINE=$(MACHINE) $(boot)/$@

……

从这里可以看出，zImage的依赖是顶层vmlinux文件，下面的命令展开得到：

make -f scripts/Makefile.build obj= arch/arm/boot MACHINE=arch/arm/mach-\* arch/arm/boot/ zImage

**可以看出zImage其实是make解析arch/arm/boot目录下的Makefile文件生成的，而参数传递了目标芯片信息和目标“arch/arm/boot/zImage”。所以zImage其实是在arch/arm/boot目录下完成编译的，这就是为什么可引导zImage映像会在arch/arm/boot目录下。**

### arch/$(ARCH)/boot/Makefile

现在来分析一下arch/arm/boot/Makefile中的部分规则，看看目标zImage的生成：

$(obj)/Image: vmlinux FORCE

$(call if\_changed,objcopy)

@echo ' Kernel: $@ is ready'

$(obj)/compressed/vmlinux: $(obj)/Image FORCE

$(Q)$(MAKE) $(build)=$(obj)/compressed $@

**$(obj)/zImage: $(obj)/compressed/vmlinux FORCE**

$(call if\_changed,objcopy)

@echo ' Kernel: $@ is ready'

先看最后一行，从中可以得知arch/arm/boot/zImage的依赖目标是arch/arm/boot/ compressed/vmlinux，且目标zImage是其二进制化的产物。

而arch/arm/boot/compressed/vmlinux是如何得到的呢？再看上一规则，arch/arm/boot/compressed/vmlinux的依赖目标是arch/arm/boot/Image。这个依赖目标的生成由最上面的规则决定，显然arch/arm/boot/Image是由顶层vmlinux二进制化得到的。而中间这行规则的含义是arch/arm/boot/compressed/vmlinux由make解析arch/arm/boot/compressed/目录下的Makefile文件生成的，这条命令展开得到：

make -f scripts/Makefile.build

obj= arch/arm/boot/compressed arch/arm/boot/compressed/vmlinux

### arch/$(ARCH)/boot/compressed/Makefile

看看arch/arm/boot/compressed/vmlinux 的生成：

......

suffix\_$(CONFIG\_KERNEL\_GZIP) = gzip

suffix\_$(CONFIG\_KERNEL\_LZO) = lzo

suffix\_$(CONFIG\_KERNEL\_LZMA) = lzma

......

$(obj)/vmlinux: $(obj)/vmlinux.lds $(obj)/$(HEAD) $(obj)/piggy.$(suffix\_y).o \

$(addprefix $(obj)/, $(OBJS)) $(lib1funcs) FORCE

$(call if\_changed,ld)

@$(check\_for\_bad\_syms)

**$(obj)/piggy.$(suffix\_y): $(obj)/../Image FORCE**

**$(call if\_changed,$(suffix\_y))**

$(obj)/piggy.$(suffix\_y).o: $(obj)/piggy.$(suffix\_y) FORCE

上面的第一条规则就说明了：其实arch/arm/boot/compressed/vmlinux是由几个部分根据arch/arm/boot/compressed/vmlinux.lds 脚本链接而成的：

$(obj)/$(HEAD)：arch/arm/boot/compressed/head.o，在链接时处于vmlinux的最前面，其主要作用就是做一些必要的初始化工作，如初始化CPU、中断描述符表IDT 和内存页目录表GDT等等，最后跳到misc.c中的decompress\_kernel函数进行内核的自解压工作。

$(addprefix $(obj)/, $(OBJS))：arch/arm/boot/compressed/ misc.o，位于head.o之后，是内核自解压的实现代码。

以下假定是gzip模式压缩：

$(obj)/piggy.$(suffix\_y).o：arch/arm/boot/compressed/ piggy.gzip.o，其实是arch/arm/boot/Image经过gzip压缩后（piggy.gzip），再借助piggy.gzip.S一起编译出的ELF可链接文件。其中的原理可以看看piggy.gzip.S源码：

.section .piggydata,#alloc

.globl input\_data

input\_data:

.incbin "arch/arm/boot/compressed/piggy.gzip"

.globl input\_data\_end

input\_data\_end:

这里我还是要额外的提一下gzip压缩，也就是$(call if\_changed,$(suffix\_y))这个过程。这个命令认真解析起来比较麻烦，这里如果有兴趣的读者可以自行分析。这里介绍两篇经典的分析文档：《kbuild实现分析》、《Kbuild系统原理分析》，读者可自行上网下载学习。这里我直接给出了结果，这条命令执行了在Makefile.lib(scripts)中定义的：

cmd\_gzip = (cat $(filter-out FORCE,$^) | gzip -n -f -9 > $@) || \

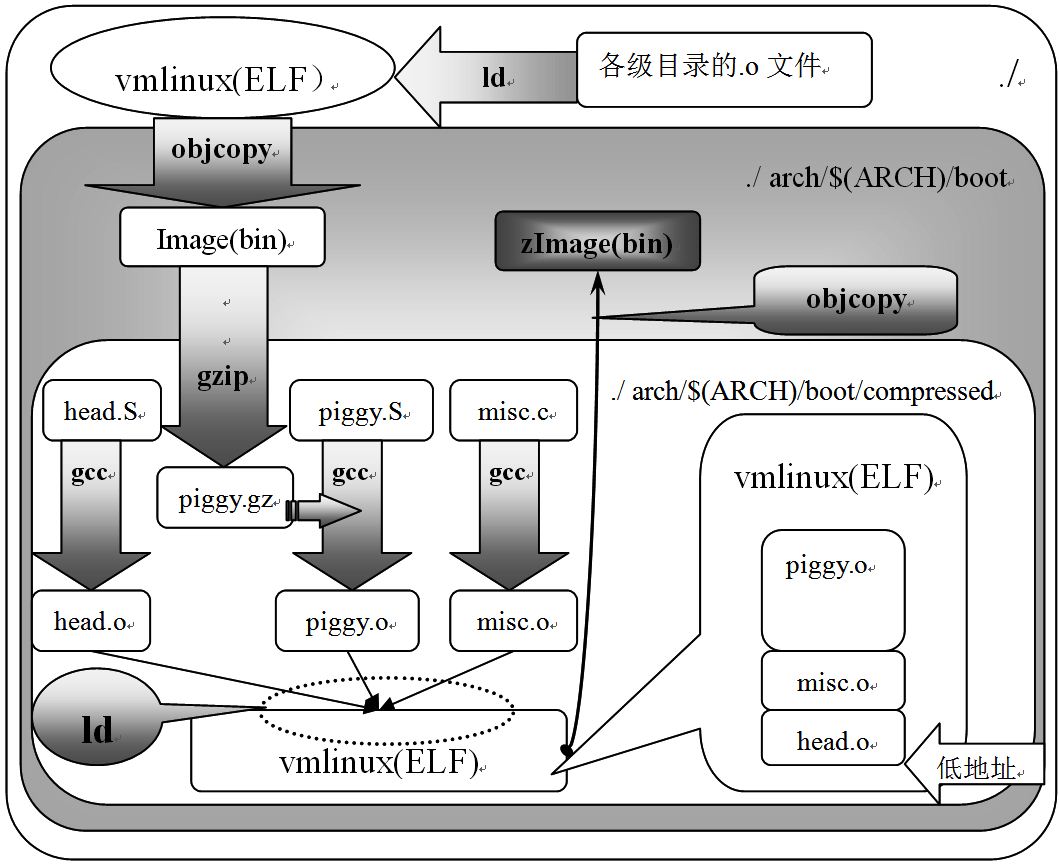
(rm -f $@ ; false)

也就是说，piggy.gzip是将arch/arm/boot/Image文件cat到标准输出，并通过管道传入gzip命令（gzip -n -f -9 ）的标准输入，最后将gzip的输出重定向到目标piggy.gzip

而这个piggy.gzip文件有一个重要的特性：最后的四个字节，是文件压缩前的大小数据，存放格式是小端模式。这个数据在zImage自解压时会被用于程序得到内核解压后所需要的空间！！！

感兴趣的朋友可以自己随便用“gzip -n -f -9”压缩一个文件试试，验证一下，我已亲自验证过了。

这样跟踪下来，zImage的产生过程已经看完了，但是读者可能会被这有点复杂的关系绕晕了，所以现在可以结合一下的流程图简单地总结一下：



首先顶层vmlinux是ELF格式的可执行文件，必须将其二进制化生成Image后才可以被bootloader引导。为了实现压缩的内核映像，arch/arm/boot/compressed/Makefile又将这个非压缩映像Image做gzip压缩，生成了piggy.gzip。但要实现在启动时自解压，必须将这个piggy.gzip转化为.o文件，并同初始化程序head.o和自解压程序misc.o一同链接，生成arch/arm/boot/compressed/vmlinux。最后arch/arm/boot/Makefile将这个ELF格式的arch/arm/boot/compressed/vmlinux二进制化得到可被bootloader引导的映像文件zImage。