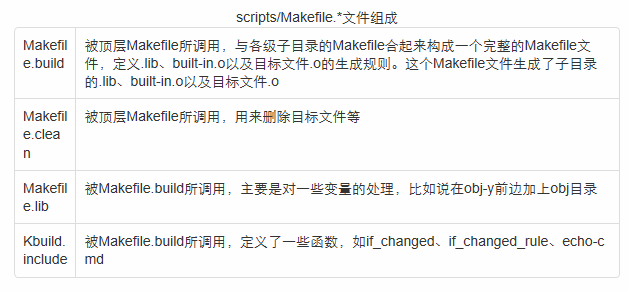
# Linux内核编译流程

## 1.Linux内核makefile文件组成





Makefile.host 本机编译工具（hostprog-y）的编译规则。  
Makefile.modpost 块编译的第二阶段,由.o和.mod生成.ko时使用的规则。

Linux内核Makefile体系核心的Makefile文件就两个：顶层Makefile、scripts/Makefile.build。  
子目录中的Makefile、kbuild不是Makefile文件（完整的Makefile文件），只能算作是Makefile的包含文件。   
顶层Makefile文件负责将各个目录生成的\*.built-in.o、lib.a等文件连接到一起。而scripts/Makefile.build 包含子目录中的Makefile文件来生成这些\*.built-in.o、lib.a、\*.o等文件。

## 2.linux内核make过程分析

### 2.1 整体分析

当执行make uImage时，uImage并不在顶层的makefile中定义，而是在arch/arm/Makefile中定义。

#顶层makefile

ARCH ?**=** arm

CROSS\_COMPILE ?**=** arm-linux-

include $(srctree)/arch/$(ARCH)/Makefile

在arch/arm/makefile中可以找到uImage的定义。

#arch/arm/Makefile

zImage Image xipImage bootpImage uImage**:** vmlinux

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

可见uImage依赖于vmlinux，vmlinux在顶层makefile中定义，最后执行后面的命令完成编译。

### 2.2 vmlinux的生成

#### 2.2.1 vmlinux的依赖

vmlinux依赖$(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) $(kallsyms.o)这些目标。

#顶层makefile

vmlinux**:** $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) $(kallsyms.o) FORCE

下面来逐个分析这些依赖目标。  
**(1)vmlinux-lds：**链接脚本确认，在最后阶段需要使用到。

#顶层makefile

vmlinux-lds **:=** arch/$(ARCH)/kernel/vmlinux.lds

**(2)vmlinux-init：**vmlinux-init依赖$(head-y)和$(init-y)

#顶层makefile

vmlinux-init **:=** $(head-y) $(init-y)

#一开始init-y是个目录

init-y **:=** init/

#后续把init-y变量中所有符合%/的替换为%/built-in.o，这句之后init-y 被赋值为 init/built-in.o

init-y **:=** $(patsubst %/, %/built-in.o, $(init-y))

#arch/arm/Makefile

head-y **:=** arch/arm/kernel/head$(MMUEXT).o arch/arm/kernel/init\_task.o

所以最终vmlinux-init 实际上是:   
1. arch/arm/kernel/head.o（这是Image/vmlinux的入口代码)。   
2. arch/arm/kernel/init\_task.o   
3. init/built-in.o   
三者链接而来的。

**(3)vmlinux-main**

vmlinux-main **:=** $(core-y) $(libs-y) $(drivers-y) $(net-y)

可以看到vmlinux-main有四个依赖选项。

core-y **:=** usr/

core-y +**=** kernel/ mm/ fs/ ipc/ security/ crypto/ block/

core-y +**=** arch/arm/kernel/ arch/arm/mm/ arch/arm/common/

core-$(CONFIG\_ARCH\_S3C2410) +**=** arch/arm/mach-s3c2440/

core-y **:=** $(patsubst %/, %/built-in.o, $(core-y))

编译内核的话，core-y最终包含: usr/ kernel/ mm/ fs/ ipc/ security/ crypto/ block/目录下的built-in.o。  
除此之外,core-y还包含体系结构相关的arch/arm/kernel mm、common以及具体芯片相关的:mach-xxx,plat-xxx目录下built-in.o文件。

libs-y **:=** lib/

libs-y1 **:=** $(patsubst %/, %/lib.a, $(libs-y))

libs-y2 **:=** $(patsubst %/, %/built-in.o, $(libs-y))

libs-y **:=** $(libs-y1) $(libs-y2)

libs-y也是包含具体体系结构相关的库，并且包括lib.a和built-in.o两个文件。

drivers-y **:=** drivers/ sound/

drivers-y **:=** $(patsubst %/, %/built-in.o, $(drivers-y))

drivers-y值最终等于drivers/built-in.o sound/built-in.o

net-y **:=** net/

net-y **:=** $(patsubst %/, %/built-in.o, $(net-y))

net-y最终等于net/built-in.o。

vmlinux-main是由以下四个目标组成的:

core-y:

包含体系结构无关的usr/ kernel/ mm/ fs/ ipc/ security/ crypto/ block/ 目录下的built-in.o文件(如果是编译module，则只包含usr/built-in.o)。

包含体系结构相关的文件，在arm下为arch/arm/kernel/ arch/arm/mm/ arch/arm/common/ arch/arm/net/下的built-in.o。

包含具体芯片相关的文件，如arch/arm/plat-%/ arch/arm/mach-%/下的built-in.o。

libs-y: 包含arch/arm/lib/和 lib/目录下的built-in.o和lib.a文件。

drivers-y: 包含drivers/ sound/ firmware/以及arch/arm/oprofile/（可选）目录下的built-in.o文件。

net-y: 包含net/built-in.o文件。

**(4) kallsyms.o**

在2.6内核中，为了更好的调试内核，引入了kallsyms机制。kallsyms把内核中用到的所有函数地址和名称链接到内核文件，当内核启动后，同时加载到内存中。当发生oops时候，内核就会 解析eip位于哪个函数中，然后打印出backtrace信息。

内核编译的最后，make会执行:nm -n vmlinux|scripts/kallsyms,其中：   
1. nm -n vmlinux负责生成所有的内核符号并按地址排序   
2. scripts/kallsyms负责处理这个列表，并生成需要的链接文件tmp\_kallsyms%.s   
也就是说kallsyms实际上是内核编译完了之后，vmlinux中通过nm命令生成的，所以所有符号地址都包括了，实际上是和System.map是一样的。   
而且kallsyms中所有函数的地址，是放在一个全局数组kallsyms\_addresses[]中的，如下:

kallsyms\_addresses**:**

PTR \_text + 0x180

PTR \_text + 0x180

PTR \_text + 0x180

PTR \_text + 0x194

PTR \_text + 0x360

PTR \_text + 0x374

PTR \_text + 0x484

PTR \_text + 0x594

PTR \_text + 0x5f4

PTR \_text + 0x5f8

PTR \_text + 0x60c

PTR \_text + 0x624

kallsyms\_addresses中的每一个表项，都在重定位表中有记录，如果内核发生了重定位，那么kallsyms中的内容也会跟着修改，所以cat /proc/kallsyms的时候总是看到的是真正的函数地址。

kallsyms的整个符号表，最终都会放在kallsyms.o文件中。

kallsyms.o **:=** .tmp\_kallsyms$(last\_kallsyms).o

kallsyms也是最终依赖一个文件，这个是.tmp\_kallsymsX.o。

**(5) FORCE**

它的目的是强迫重建vmlinux(无论上边的原材料是否已经构建，无论上边的原材料是否比已经生成的目标新，都要重建)，这种用法在Linux Makefile体系中经常见到。

PHONY +**=** FORCE

FORCE**:**

.PHONY**:** $(PHONY)

#### 2.2.2 vmlinux的链接

当上述依赖完成对应的编译后，便是vmlinux的链接过程了。

ifdef CONFIG\_HEADERS\_CHECK

$(Q)$(MAKE) -f $(srctree)/Makefile headers\_check

endif

$(call if\_changed\_rule,vmlinux\_\_)

$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost $@

$(Q)rm -f .old\_version

其中$(call if\_changed\_rule,vmlinux\_\_)会调用rule\_vmlinux\_,即下面内容：

define rule\_vmlinux\_\_

:

$(if $(CONFIG\_KALLSYMS),,+$(call cmd,vmlinux\_version))

$(call cmd,vmlinux\_\_)

$(Q)echo 'cmd\_$@ := $(cmd\_vmlinux\_\_)' > $(@D)/.$(@F).cmd

$(Q)$(if $(*$(quiet)cmd\_sysmap), \*

echo ' $($(quiet)cmd\_sysmap) System.map' &&) \

$(cmd\_sysmap) $@ System.map; \

if [ $$? -ne 0 ]; then \

rm -f $@; \

/bin/false; \

fi;

$(verify\_kallsyms)

endef

在这上面有命令vmlinux\_\_, \_vmlinux\_version和sysmap定义， 其中vmlinux\_命令中有LD的链接生成vmlinux。

quiet\_cmd\_vmlinux\_\_ ?**=** LD $@

cmd\_vmlinux\_\_ ?**=** $(LD) $(LDFLAGS) $(LDFLAGS\_vmlinux) -o $@ \

-T $(vmlinux-lds) $(vmlinux-init) \

--start-group $(vmlinux-main) --end-group \

$(filter-out $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) FORCE ,$^)

实际生成vmlinux的命令如下：

arm-linux-ld -EL -p --no-undefined -X -o vmlinux -T arch/arm/kernel/vmlinux.lds arch/arm/kernel/head.o arch/arm/kernel/init\_task.o init/built-in.o --start-group usr/built-in.o arch/arm/kernel/built-in.o arch/arm/mm/built-in.o arch/arm/common/built-in.o arch/arm/mach-s3c2410/built-in.o arch/arm/mach-s3c2400/built-in.o arch/arm/mach-s3c2412/built-in.o arch/arm/mach-s3c2440/built-in.o arch/arm/mach-s3c2442/built-in.o arch/arm/mach-s3c2443/built-in.o arch/arm/nwfpe/built-in.o arch/arm/plat-s3c24xx/built-in.o kernel/built-in.o mm/built-in.o fs/built-in.o ipc/built-in.o security/built-in.o crypto/built-in.o block/built-in.o arch/arm/lib/lib.a lib/lib.a arch/arm/lib/built-in.o lib/built-in.o drivers/built-in.o sound/built-in.o net/built-in.o --end-group .tmp\_kallsyms2.o

#### 2.2.3 $(vmlinux-dirs)

上面只讨论了vmlinux依赖哪些文件生成，但是这些依赖文件是如何生成的呢？  
这三个选项又都依赖$(vmlinux-dirs)。

$(sort $(vmlinux-init) $(vmlinux-main)) $(vmlinux-lds)**:** $(vmlinux-dirs) ;

vmlinux-dirs **:=** $(patsubst %/,%,$(filter %/, $(init-y) *$(init-m) \*

$(core-y) $(core-m) $(drivers-y) $(drivers-m) \

$(net-y) $(net-m) $(libs-y) $(libs-m)))

filter首先过滤掉不是文件夹的（带“/”的就是文件夹），然后patsubst将所有的文件夹后边的“/”去掉。$(vmlinux-dirs)的值大致为：

init usr arch/arm/kernel arch/arm/mm arch/arm/crypto arch/arm/vdso kernel mm fs ipc security crypto block drivers sound firmware net lib arch/arm/lib

可见$(vmlinux-dirs)都是源码包本来存在的文件夹，怎么通过这些文件夹的名字来生成它们目录下的\*.built-in.o、lib.a呢？

$(vmlinux-dirs)**:** prepare scripts

$(Q)$(MAKE) $(build)=$@

$(vmlinux-dirs)都是伪目标，这样即使这些文件夹是存在的，也要执行它们对应的命令。

**(1)prepare**

在正式开始编译之前，要先将准备工作做好，检查输出目录是否与源码目录分开、如果分开了要创建这样的目录，还要生成include/config/kernel.release、include/linux/version.h include/linux/utsrelease.h 等文件。

prepare**:** prepare0

prepare0**:** archprepare FORCE

$(Q)$(MAKE) $(build)=.

$(Q)$(MAKE) $(build)=. missing-syscalls

archprepare**:** prepare1 scripts\_basic

prepare1**:** prepare2 include/linux/version.h include/linux/utsrelease.h

include/asm include/config/auto.conf

prepare2**:** prepare3 outputmakefile

prepare3**:** include/config/kernel.release

scripts\_basic**:**

$(Q)$(MAKE) $(build)=scripts/basic

**(2)scripts**

准备工作还包括scripts，这些脚本用来决定编译时的选项，例如include/config/auto.conf。

scripts**:** scripts\_basic include/config/auto.conf

$(Q)$(MAKE) $(build)=$(@)

**(3)$(Q)$(MAKE) $(build)=$@**

编译命令是至关重要的一句话，这句话就是来调用通用子Makefile文件--scripts/Makefile.build。$@会依次 被 $(vmlinux-dirs) 中的文件夹代替，从而依次执行$(Q)$(MAKE) $(build)=$@命令编译该文件夹以生成\*.built-in.o、lib.a等文件。

#### 2.2.4 例子drivers/ built-in.o的生成

我们以$@为drivers为例进行以下内容的说明，下边这条命令最终的结果是生成drivers/built-in.o。

make -f scripts/Makefile.build obj**=**drivers

**(1)Makefile.build的包含文件**

-include include/config/auto.conf

include scripts/Kbuild.include

src := $(obj)

kbuild-dir **:=** $(if $(filter /%,$(src)),$(src),$(srctree)/$(src))

kbuild-file **:=** $(if $(wildcard $(kbuild-dir)/Kbuild),$(kbuild-dir)/Kbuild,$(kbuild-dir)/Makefile)

include $(kbuild-file)

include scripts/Makefile.lib

ifneq ($(hostprogs-y)$(hostprogs-m),)

include scripts/Makefile.host

endif

包含文件有依据.config生成的配置文件include/config/auto.conf、include scripts/Kbuild.include、$(kbuild-file)（也就是子目录的Makefile或者kbuild）、include scripts/Makefile.lib、include scripts/Makefile.host（只在一定情况下才被包含进入）。  
这里着重讲一下$(kbuild-file)值的变化过程。  
kbuild-dir的等式中有$(filter /%,$(src))函数，它的目的是将不是以“/”开头的目录滤除，显然我们的src=drivers，所以就被滤除掉，这个函数的值为空。  
$(if $(filter /%,$(src)),$(src),$(srctree)/$(src))的结果最终等于$(srctree)/$(src)，也就是给src添加绝对路径。所以kbuild-dir值等于/workspace/linux-2.6.22.6/drivers。  
kbuild-file右边的等式，首先查看在kbuild-dir目录中是否有Kbuild存在，如果有就等于这个文件，否则使用这个目录中的Makefile文件。kbuild-file最终等于/workspace/linux-2.6.22.6/drivers/Makefile。

**(2) Makefile.build的总目标**

\_\_build**:** $(if $(KBUILD\_BUILTIN),$(builtin-target) $(lib-target) $(extra-y)) \

$(if $(KBUILD\_MODULES),$(obj-m)) \

$(subdir-ym) $(always)

@

在终端以#make uImage执行Make时,KBUILD\_BUILTIN := 1。  
builtin-target值为drivers/built-in.o。  
lib-target的值为drivers/lib.a。  
subdir-ym的定义：

\_\_subdir-y **:=** $(patsubst %/,%,$(filter %/, $(obj-y)))

subdir-y +**=** $(\_\_subdir-y)

\_\_subdir-m **:=** $(patsubst %/,%,$(filter %/, $(obj-m)))

subdir-m +**=** $(\_\_subdir-m)

subdir-ym **:=** $(sort $(subdir-y) $(subdir-m))

subdir-ym **:=** $(addprefix $(obj)/,$(subdir-ym))

\_\_subdir-y和\_\_subdir-m首先将obj-y、obj-m中的非文件夹滤除，然后通过patsubst函数将最后的“/”去除。注意到drivers/Makefile中的obj-y、obj-m通通都是文件夹。我们以obj-y = net/为例进行以下内容的说明，所以\_\_subdir-y=net。  
subdir-ym就是在\_\_subdir-y和\_\_subdir-m前边添加obj的前缀，所以最终等于drivers/net。

**(3) drivers/built-in.o的生成**

scripts/Makefile.build的总目标\_\_build中的一个目标是 builtin-target，而它的值为drivers/built-in.o，现在就来看看它是怎样生成的。

$(builtin-target)**:** $(obj-y) FORCE

*$(call if\_changed,link\_o\_target*

drivers/built-in.o依赖于obj-y（子目标），然后通过调用一个if\_changed函数，将这些子目标连接起来，生成drivers/built-in.o。通过命令打印查看连接命令如下：

arm-linux-ld -EL -r -o drivers/built-in.o drivers/video/built-in.o drivers/char/built-in.o drivers/serial/built-in.o drivers/parport/built-in.o drivers/base/built-in.o drivers/block/built-in.o drivers/misc/built-in.o drivers/mfd/built-in.o drivers/net/built-in.o drivers/media/built-in.o drivers/macintosh/built-in.o drivers/scsi/built-in.o drivers/cdrom/built-in.o drivers/auxdisplay/built-in.o drivers/mtd/built-in.o drivers/spi/built-in.o drivers/usb/built-in.o drivers/input/serio/built-in.o drivers/input/built-in.o drivers/rtc/built-in.o drivers/i2c/built-in.o drivers/hwmon/built-in.o drivers/mmc/built-in.o drivers/leds/built-in.o drivers/firmware/built-in.o drivers/crypto/built-in.o drivers/hid/built-in.o

**(4)drivers/net/built-in.o的生成**

drivers/built-in.o的依赖如何生成呢，比如说drivers/net/built-in.o。  
还记得subdir-ym吗？subdir-ym记录了当前目录里边要参与编译连接的子文件夹。

$(subdir-ym)**:**

$(Q)$(MAKE) $(build)**=**$

再一次调用scripts/Makefile.build，并传递参数drivers/net，这和生成drivers/built-in.o没什么差别。  
再次转战到scripts/Makefile.build，obj=drivers/net被带到该文件，大致编译流程为：

arm-linux-ld -EL -r -o drivers/net/built-in.o drivers/net/mii.o drivers/net/Space.o drivers/net/loopback.o drivers/net/dm9dev9000c.o drivers/net/arm/built-in.o drivers/net/wireless/built-in.o

### 2.3 uImage zImage的生成

zImage Image xipImage bootpImage uImage**:** vmlinux

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

上面的make命令等效为：

make -f scripts/Makefile.build obj**=**arch/arm/boot MACHINE=arch/arm/mach-s3c2410/ arch/arm/boot/uImage

下面来看一下arm/arm/boot下的makefile。

$(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'

$(obj)/uImage**:** $(obj)/zImage FORCE

$(call if\_changed,uimage)

@echo ' Image $@ is ready'

可以看出要生成uImage依赖于zImage，步骤如下：

**(1)Image生成**

将vmlinux转成二进制文件并去除调试信息，注释和符号表等内容。

arm-linux-objcopy -O binary -R .note -R .comment -S vmlinux arch/arm/boot/Image

**(2) compressed/vmlinux生成**

compressed/vmlinux为压缩过的linux内核。

make -f scripts/Makefile.build obj=arch/arm/boot/compressed arch/arm/boot/compressed/vmlinux

arch/arm/boot/compressed下的makefile内容：

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

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

$(call if\_changed,ld)

@:

$(obj)/piggy.gz**:** $(obj)/../Image FORCE

$(call if\_changed,gzip)

$(obj)/piggy.o**:** $(obj)/piggy.gz FORCE

CFLAGS\_font\_acorn\_8x8.o **:=** -Dstatic=

$(obj)/vmlinux.lds**:** $(obj)/vmlinux.lds.in arch/arm/boot/Makefile .config

@sed "$(SEDFLAGS)" < $< > $@

$(obj)/misc.o**:** $(obj)/misc.c include/asm/arch/uncompress.h lib/inflate.c

先将arch/arm/boot/compressed/head.S编译成arch/arm/boot/compressed/head.o。

arm-linux-gcc -Wp,-MD,arch/arm/boot/compressed/.head.o.d -nostdinc -isystem /opt/JZ2440\_TOOLS/gcc-3.4.5-glibc-2.3.6/bin/../lib/gcc/arm-linux/3.4.5/include -D\_\_KERNEL\_\_ -Iinclude -include include/linux/autoconf.h -mlittle-endian -D\_\_ASSEMBLY\_\_ -mapcs-32 -mno-thumb-interwork -D\_\_LINUX\_ARM\_ARCH\_\_=4 -march=armv4t -mtune=arm9tdmi -msoft-float -gdwarf2 -c -o arch/arm/boot/compressed/head.o arch/arm/boot/compressed/head.S

将 arch/arm/boot/Image 用gzip -9 压缩生成arch/arm/boot/compressed/piggy.gz

gzip -f -9 < arch/arm/boot/compressed/../Image > arch/arm/boot/compressed/piggy.gz

编译arch/arm/boot/compressed/piggy.S 生成arch/arm/boot/compressed/piggy.o

这里实际上是将piggy.gz通过piggy.S编译进piggy.o文件中。而piggy.S文件仅有6行，只是包含了文件piggy.gz。  
编译arch/arm/boot/compressed/misc.c 生成arch/arm/boot/compressed/misc.o。

arm-linux-gcc -Wp,-MD,arch/arm/boot/compressed/.piggy.o.d -nostdinc -isystem /opt/JZ2440\_TOOLS/gcc-3.4.5-glibc-2.3.6/bin/../lib/gcc/arm-linux/3.4.5/include -D\_\_KERNEL\_\_ -Iinclude -include include/linux/autoconf.h -mlittle-endian -D\_\_ASSEMBLY\_\_ -mapcs-32 -mno-thumb-interwork -D\_\_LINUX\_ARM\_ARCH\_\_=4 -march=armv4t -mtune=arm9tdmi -msoft-float -gdwarf2 -c -o arch/arm/boot/compressed/piggy.o arch/arm/boot/compressed/piggy.S

arm-linux-gcc -Wp,-MD,arch/arm/boot/compressed/.misc.o.d -nostdinc -isystem /opt/JZ2440\_TOOLS/gcc-3.4.5-glibc-2.3.6/bin/../lib/gcc/arm-linux/3.4.5/include -D\_\_KERNEL\_\_ -Iinclude -include include/linux/autoconf.h -mlittle-endian -Wall -Wundef -Wstrict-prototypes -Wno-trigraphs -fno-strict-aliasing -fno-common -Os -marm -fno-omit-frame-pointer -mapcs -mno-sched-prolog -mapcs-32 -mno-thumb-interwork -D\_\_LINUX\_ARM\_ARCH\_\_=4 -march=armv4t -mtune=arm9tdmi -malignment-traps -msoft-float -Uarm -fno-omit-frame-pointer -fno-optimize-sibling-calls -g -Wdeclaration-after-statement -fpic -Dstatic= -D"KBUILD\_STR(s)=#s" -D"KBUILD\_BASENAME=KBUILD\_STR(misc)" -D"KBUILD\_MODNAME=KBUILD\_STR(misc)" -c -o arch/arm/boot/compressed/misc.o arch/arm/boot/compressed/misc.c

依据arch/arm/boot/compressed/vmlinux.lds将arch/arm/boot/compressed/目录下的文件head.o 、piggy.o 、misc.o链接生成 arch/arm/boot/compressed/vmlinux，这个vmlinux是经过压缩且含有自解压代码的内核。

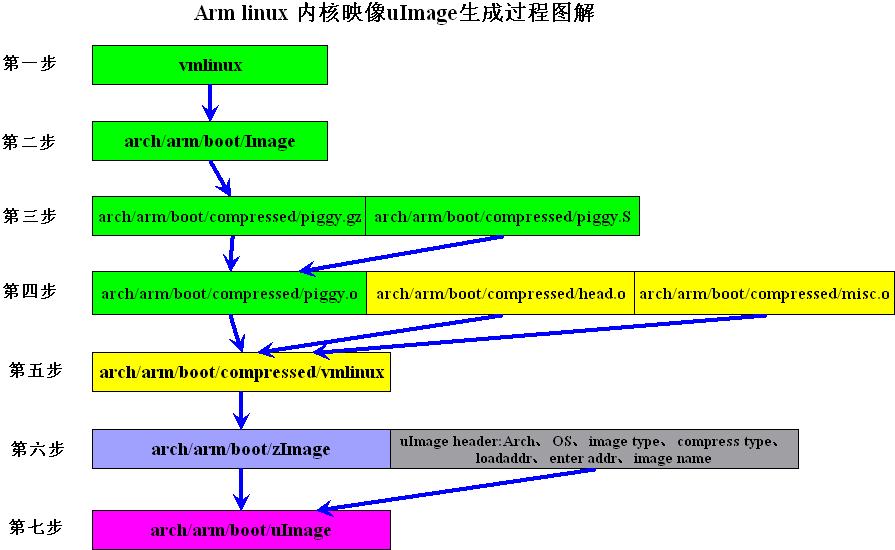
arm-linux-ld -EL --defsym zreladdr=0x30008000 --defsym params\_phys=0x30000100 -p --no-undefined -X /opt/JZ2440\_TOOLS/gcc-3.4.5-glibc-2.3.6/bin/../lib/gcc/arm-linux/3.4.5/libgcc.a -T arch/arm/boot/compressed/vmlinux.lds arch/arm/boot/compressed/head.o arch/arm/boot/compressed/piggy.o arch/arm/boot/compressed/misc.o -o arch/arm/boot/compressed/vmlinux

生成zImage二进制文件

arm-linux-objcopy -O binary -R .note -R .comment -S arch/arm/boot/compressed/vmlinux arch/arm/boot/zImage

uImage二进制文件生成通过mkimage完成，在zImage前添加64个字节的头部。

/bin/bash /home/jay/Code/JZ2440/linux-2.6.22.6/scripts/mkuboot.sh -A arm -O linux -T kernel -C none -a 0x30008000 -e 0x30008000 -n 'Linux-2.6.22.6' -d arch/arm/boot/zImage arch/arm/boot/uImage



## 3.linux的make menuconfig

执行make menuconfig会执行顶层的makefile。

%config**:** scripts\_basic outputmakefile FORCE

    $(Q)mkdir -p include/linux include/config

    $(Q)$(MAKE) $(build)**=**scripts/kconfig $@

$(Q)$(MAKE) $(build)=scripts/kconfig $@ 等效为

$(Q) make -f $(srctree)/scripts/Makefile.build obj**=**scripts/kconfig menuconfig

build **:=** -f $(srctree)/scripts/Makefile.build obj

也就是在Makefile.build的makefile中再include scripts/kconfig/makefile。

在scripts/kconfig/Makefile找到如下代码：

menuconfig**:** $(obj)/mconf

$< arch/$(ARCH)/Kconfig

该部分代码等效为：

menuconfig**:** scripts/kconfig/mconf

      scripts/kconfig/mconf  arch/arm/Kconfig

当我们执行命令make menuconfig时  
(1)先创建两个二级目录include /linux 和 include/config。  
(2)接着把scripts/kconfig/里面的可执行文件mconf运行起来，有能力的话可以详看scripts/kconfig/mconf.c，根据文件arch/arm/kconfig内容，显示出一个菜单界面。其实，mconf运行起来，首先是“画出”菜单界面（显示内容则根据各级目录的Kconfig文件），然后查看当前目录有没有存在.config文件，若没有，则按照默认显示到菜单里面；若存在.config，则会把它读出来存到内存某块区域，在逐行逐行解析它的内容，再把读出的内容更新到菜单。  
最后会保存在.config文件（当前目录中）。

下面来分析一下mconf可执行文件是如何生成的。

lxdialog **:=** lxdialog/checklist.o lxdialog/util.o lxdialog/inputbox.o

lxdialog +**=** lxdialog/textbox.o lxdialog/yesno.o lxdialog/menubox.o

conf-objs **:=** conf.o zconf.tab.o

mconf-objs **:=** mconf.o zconf.tab.o $(lxdialog)

kxgettext-objs **:=** kxgettext.o zconf.tab.o

hostprogs-y **:=** conf qconf gconf kxgettext

ifeq ($(MAKECMDGOALS),menuconfig)

hostprogs-y += mconf

endif

Kbuild 框架中，专门使用hostprogs-y变量来指示在内核编译阶段需要使用的一些可执行文件，通过hostprogs-y += mconf，就向make程序指明mconf是一个编译阶段需要使用的可执行文件。当hostprogs-y存在时，在makefile.build中就会include makefile.host脚本，完成对应bin文件的编译。

ifneq ($(hostprogs-y)$(hostprogs-m),)

include scripts/Makefile.host

endif

## 4.linux内核clean操作

make clean  
删除大多数的编译生成文件，但是会保留内核的配置文件.config，还有足够的编译支持来建立扩展模块

make mrproper   
删除所有的编译生成文件，还有内核配置文件，再加上各种备份文件

make distclean   
mrproper删除的文件，加上编辑备份文件和一些补丁文件。

删除的文件范围从小到大依次为: make clean < make mrproper < make distclean。

jay@jay-VirtualBox**:**~/Code/JZ2440/linux-2.6.22.6$ make clean

CLEAN arch/arm/boot/compressed

CLEAN arch/arm/boot

CLEAN /home/jay/Code/JZ2440/linux-2.6.22.6

CLEAN arch/arm/kernel

CLEAN drivers/char

CLEAN drivers/video/logo

CLEAN init

CLEAN lib

CLEAN usr

CLEAN include/asm-arm/mach-types.h include/asm-arm/arch include/asm-arm/.arch vmlinux System.map .tmp\_kallsyms2.S .tmp\_kallsyms1.o .tmp\_kallsyms2.o .tmp\_kallsyms1.S .tmp\_vmlinux1 .tmp\_vmlinux2 .tmp\_System.map

## 5.其他常用知识

### 5.1 make V=1 uImage

在编译的过程中如果想看到更多的打印，使用V=1选项。

ifdef V

ifeq ("$(origin V)", "command line")

KBUILD\_VERBOSE **=** $(V)

endif

endif

ifndef KBUILD\_VERBOSE

KBUILD\_VERBOSE **=** 0

endif

ifeq ($(KBUILD\_VERBOSE),1)

quiet **=**

Q **=**

else

quiet**=**quiet\_

Q **=** @

endif

### 5.2 EXTRA\_CFLAGS

各级子目录下的makefile中可以设置能够影响当前目录下所有文件编译、连接选项：EXTRA\_CFLAGS、EXTRA\_AFLAGS、EXTRA\_LDFLAGS、EXTRA\_ARFLAGS；还可以设置影响某个文件的编译选项：CFLAGS\_$@,AFLAGS\_$@。

例如：

EXTRA\_CFLAGS +**=** -I$(obj)

ifdef DEBUG

EXTRA\_CFLAGS +**=** -DEMU10K1\_DEBUG

endif

CFLAGS\_aha152x.o **=** -DAHA152X\_STAT -DAUTOCONF

### 5.3 一些重要的文件

include/config/auto.conf 顶层makefile包含的文件，由.config生成。

include/linux/autoconf.h 给linux源码使用的头文件。在编译的过程中头文件指定如下：-Iinclude -include include/linux/autoconf.h