[**linux源码Makefile的详细分析**](http://www.cnblogs.com/amanlikethis/p/3675486.html)

**目录**

[**一、概述**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab1)

[**1、本文的意义**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab11)

[**2、Linux内核Makefile文件组成**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab12)

[**二、Linux内核Makefile的“make解析”过程**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab2)

[**1 顶层Makefile阶段**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab21)

[**1、从总目标uImage说起**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab211)

[**2、vmlinux的生成**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab212)

[**3、vmlinux-lds、vmlinux-init、vmlinux-main的生成**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab213)

[**2 scripts/Makefile.build的第一次调用阶段**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab22)

[**1、Makefile.build的包含文件**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab221)

[**2、scripts/Makefile.build的总目标**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab222)

[**3、drivers/built-in.o的生成**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab223)

[**4、drivers/net/built-in.o**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab224)

[**3 scripts/Makefile.build的再一次调用阶段**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab23)

[**1、Makefile.build的包含文件**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab231)

[**2、scripts/Makefile.build的总目标**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab232)

[**3、drivers/net/built-in.o的生成**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab233)

[**4、drivers/net/dm9000.o**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab234)

[**5、drivers/net/e1000/built-in.o的生成**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab235)

[**三、Linux内核整体编译过程**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab3)

[**四、Linux内核编译构成元素**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab4)

[**1、Makefile的目标**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab41)

[**2、Makefile的依赖**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab42)

[**3、Makefile的规则**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab43)

[**4、Makefile的编译连接选项**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab44)

[**五、Linux内核Makefile特点**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab5)

[**1、两个Makefile**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab51)

[**2、编译的目录始终是顶层目录**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab52)

[**3、通用规则**](http://www.cnblogs.com/amanlikethis/p/3675486.html#lab53)

**一、概述**

**1、本文的意义**

　　众多的资料（《嵌入式Linux应用开发完全手册》、Documentation/kbuild/makefiles.txt）已经向我们展示了一个初级Linux用户者应该懂得的知识--怎样添加需要编译的文件、添加编译的规则、多个源文件构成一个目标文件的情况等。

　　但是，一种“找到真相”的冲动迫使我想了解Linux内核编译的整个过程是怎样的。为此，查了很多资料，发现《深度探索Linux操作系统：系统构建和原理解析》一文的第三章对该问题有很详细的论述。

　　经过一番分析，也有自己的想法，写于此。本文主要讨论的是Linux内核编译的整个过程，以此为基础，再来讨论诸如Linux内核Makefile的特点、构成元素等其他的问题。

　　对于分析Linux Makefile的整个编译过程，笔者认为需要对Makefile有一定的基础。如果之前没接触过Makefile的，不建议看本文。另外，对于那些只需要初级认识的Linux用户也无需对此付出过多精力，可以直接飘过。

**2、Linux内核Makefile文件组成**

|  |  |
| --- | --- |
| Linux内核Makefile文件组成 | |
| 名称 | 描述 |
| 顶层 Makefile | 它是所有Makefile文件的核心，从总体上控制着内核的编译、连接 |
| arch/$(ARCH)/Makefile | 对应体系结构的Makefile，它用来决定哪些体系结构相关的文件参与内核的生成，并提供一些规则来生成特定格式的内核映像 |
| scripts/Makefile.\* | Makefile公用的通用规则、脚本等 |
| 子目录kbuild Makefiles | 各级子目录的Makefile相对简单，被上一层Makefile.build调用来编译当前目录的文件。 |
| 顶层.config | 配置文件，配置内核时生成。所有的Makefile文件（包括顶层目录和各级子目录）都是根据.config来决定使用哪些文件的 |

 scripts/Makefile.\*中有公用的通用规则，展示如下：

|  |  |
| --- | --- |
| scripts/Makefile.\*文件组成 | |
| Makefile.build | 被顶层Makefile所调用，与各级子目录的Makefile合起来构成一个完整的Makefile文件，定义.lib、built-in.o以及目标文件.o的生成规则。这个Makefile文件生成了子目录的.lib、built-in.o以及目标文件.o |
| Makefile.clean | 被顶层Makefile所调用，用来删除目标文件等 |
| Makefile.lib | 被Makefile.build所调用，主要是对一些变量的处理，比如说在obj-y前边加上obj目录 |
| Kbuild.include | 被Makefile.build所调用，定义了一些函数，如if\_changed、if\_changed\_rule、echo-cmd |

总结：

　　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。

**二、Linux内核Makefile的“make解析”过程**

　　本文的实验源码是对“linux-2.6.30.4”进行移植后的运行在TQ2440开发板上的源码包。

**1 顶层Makefile阶段**

**1、从总目标uImage说起**

　　内核配置完成后，在顶层目录中执行“#make uImage”便开始编译内核。但是，uImage却不是在顶层Makefile中定义，而是在arch/$(ARCH)/Makefile中定义。

顶层Makefile Line 452:   
include $(srctree)/arch/$(SRCARCH)/Makefile

　　其中srctree为源码绝对路径，以我的环境为例，它的值等于/workspace/linux-2.6.30.4；而SRCARCH := $(ARCH)，即该变量等于架构名称，我们以arm为例进行说明。

arch/arm/Makefile Line 230:   
zImage Image xipImage bootpImage uImage: vmlinux

　　可见uImage依赖于vmlinux，要先生成vmlinux，然后执行下边这条指令完成编译。

arch/arm/Makefile Line 231:   
$(Q)$(MAKE) $(build)=$(boot) MACHINE=$(MACHINE) $(boot)/$@

<1> Q的定义：选择静态编译与否（是否打印编译信息）

[复制代码](javascript:void(0);)

顶层Makefile Line 290:  
ifeq ($(KBUILD\_VERBOSE),1)

quiet =

Q =

else

quiet=quiet\_

Q = @

endif

[复制代码](javascript:void(0);)

<2> MAKE: 系统环境变量，值为make

<3> build:　值为“-f scripts/Makefile.build obj=”实际上就是调用子Makefile--scripts/Makefile.build，然后传递参数目标文件夹。

Kbuild.include Line 141:  
# Shorthand for $(Q)$(MAKE) -f scripts/Makefile.build obj=

# Usage:

# $(Q)$(MAKE) $(build)=dir

build := -f $(if $(KBUILD\_SRC),$(srctree)/)scripts/Makefile.build obj

Kbuild.include被顶层Makefile所包含

顶层Makefile Line 312:

# We need some generic definitions (do not try to remake the file).

$(srctree)/scripts/Kbuild.include: ;

include $(srctree)/scripts/Kbuild.include

**2、vmlinux的生成**

[复制代码](javascript:void(0);)

顶层Makefile Line 845:   
# vmlinux image - including updated kernel symbols

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

ifdef CONFIG\_HEADERS\_CHECK

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

endif

ifdef CONFIG\_SAMPLES

$(Q)$(MAKE) $(build)=samples

endif

ifdef CONFIG\_BUILD\_DOCSRC

$(Q)$(MAKE) $(build)=Documentation

endif

$(call vmlinux-modpost)

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

$(Q)rm -f .old\_version

[复制代码](javascript:void(0);)

　　vmlinux的依赖是“$(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) vmlinux.o $(kallsyms.o) FORCE”，要想生成vmlinux必须要先把这些原材料准备好。

<1> vmlinux-lds

　　在编译之前要把连接脚本先生成，最后的连接阶段会用的着。

顶层Makefile Line 699:   
vmlinux-lds := arch/$(SRCARCH)/kernel/vmlinux.lds

<2> vmlinux-init

顶层Makefile Line 696:   
vmlinux-init := $(head-y) $(init-y)

  ①head-y

arch/arm/Makefile Line 100:   
head-y := arch/arm/kernel/head$(MMUEXT).o arch/arm/kernel/init\_task.o

②init-y

顶层Makefile Line 477:

init-y := init/

顶层Makefile Line 661:

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

　　init-y最终等于init/built-in.o

<3>vmlinux-main

顶层Makefile Line 697:

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

① core-y

[复制代码](javascript:void(0);)

arch/arm/Makefile Line 196:

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

arch/arm/Makefile Line 197:

core-y += $(machdirs) $(platdirs)

顶层Makefile Line 481:

core-y := usr/

顶层Makefile Line 650:

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

顶层Makefile Line 662:

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

[复制代码](javascript:void(0);)

　　core-y最终等于

core-y := usr/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/kernel/built-in.o arch/arm/mm/built-in.o arch/arm/common/built-in.o

② libs-y

[复制代码](javascript:void(0);)

arch/arm/Makefile Line 204:

libs-y := arch/arm/lib/ $(libs-y)

顶层Makefile Line 480:

libs-y := lib/

顶层Makefile Line 665:

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

顶层Makefile Line 666:

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

顶层Makefile Line 667:

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

[复制代码](javascript:void(0);)

　　libs-y最终等于arch/arm/lib/lib.a  lib/lib.a  arch/arm/lib/built-in.o  lib/built-in.o

③drivers-y

顶层Makefile Line 478:

drivers-y := drivers/ sound/ firmware/

顶层Makefile Line 663:

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

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

④net-y

顶层Makefile Line 479:

net-y := net/

顶层Makefile Line 664:

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

　　net-y最终等于net/built-in.o

<4> vmlinux.o

　　目前还未分析

<5> kallsyms.o

　　目前还未分析

<6> 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:

# Declare the contents of the .PHONY variable as phony. We keep that

# information in a variable so we can use it in if\_changed and friends.

.PHONY: $(PHONY)

　　可以看到FORCE为一个伪目标，所以无论如何都要重建vmlinux。

 <7> 通过编译查看最终的依赖文件

[复制代码](javascript:void(0);)

root@daneiqi:/workspace/linux-EmbedSky# make uImage V=1

arm-linux-ld -EL -p --no-undefined -X --build-id -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/plat-s3c24xx/built-in.o

arch/arm/plat-s3c/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 firmware/built-in.o net/built-in.o --end-group .tmp\_kallsyms2.o

[复制代码](javascript:void(0);)

　　命令中的参数V=1的含义是“顶层Makefile Line 36: # Use 'make V=1' to see the full commands”。

**3、vmlinux-lds、vmlinux-init、vmlinux-main的生成**

<1>  vmlinux-dirs

顶层Makefile Line 871:

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

　　vmlinux的主要构成文件vmlinux-init、vmlinux-main、vmlinux-lds的依赖是vmlinux-dirs。

顶层Makefile Line 652: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。笔者在顶层Makefile中添加一个伪目标Debug，使它打印vmlinux-dirs，然后在终端上执行命令"make Debug"得到vmlinux-dirs的值为：

init usr arch/x86/kernel arch/x86/mm arch/x86/crypto arch/x86/vdso kernel mm fs ipc security crypto block drivers sound firmware net lib arch/x86/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？

<2> vmlinux-dirs对应的命令

顶层Makefile Line 879:   
PHONY += $(vmlinux-dirs)

$(vmlinux-dirs): prepare scripts

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

　　vmlinux-dir所代表的目录都是伪目标，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。

[复制代码](javascript:void(0);)

以下代码都是摘自顶层Makefile  
Line 955:

PHONY += prepare archprepare prepare0 prepare1 prepare2 prepare3

Line 957:

# prepare3 is used to check if we are building in a separate output directory,

Line 961:

prepare3: include/config/kernel.release

Line 975:

# prepare2 creates a makefile if using a separate output directory

Line 976:

prepare2: prepare3 outputmakefile

Line 978:

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

Line 982:

archprepare: prepare1 scripts\_basic

Line 984:

prepare0: archprepare FORCE

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

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

Line 989:

prepare: prepare0

[复制代码](javascript:void(0);)

②scripts

　　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。

[复制代码](javascript:void(0);)

以下内容摘自顶层Makefile  
Line 384:

# Basic helpers built in scripts/

PHONY += scripts\_basic

scripts\_basic:

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

Line 472:

PHONY += scripts

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

[复制代码](javascript:void(0);)

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。$@会依次被vmlinux-dirs中的文件夹代替，从而依次执行命令"make -f scripts/Makefile.build obj=$@"

编译该目录中的文件以生成\*.built-in.o、lib.a等文件。

　　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

**2 scripts/Makefile.build的第一次调用阶段**

　　转战到scripts/Makefile.build，obj=drivers被带到该文件，首先看一下这个文件都包含了哪些文件

**1、Makefile.build的包含文件**

[复制代码](javascript:void(0);)

以下内容皆来自于scripts/Makefile.build

Line 34:

-include include/config/auto.conf

Line 36:

include scripts/Kbuild.include

Line 41:

# The filename Kbuild has precedence over Makefile

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

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

include $(kbuild-file)

Line 52:

include scripts/Makefile.lib

Line 61:

# Do not include host rules unless needed

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

include scripts/Makefile.host

endif

[复制代码](javascript:void(0);)

　　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。

scripts/Makefile.build Line 5:   
src := $(obj)

　 $(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-2.6.30.4/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-2.6.30.4/drivers/Makefile。

**2、scripts/Makefile.build的总目标**

[复制代码](javascript:void(0);)

scripts/Makefile.build Line 7:

PHONY := \_\_build

\_\_build:

scripts/Makefile.build Line 93:

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

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

$(subdir-ym) $(always)

@:

[复制代码](javascript:void(0);)

 <1> KBUILD\_BUILTIN

　　在终端以#Make uImage执行Make时,KBUILD\_BUILTIN := 1。

[复制代码](javascript:void(0);)

顶层Makefile Line 241:  
KBUILD\_BUILTIN := 1

# If we have only "make modules", don't compile built-in objects.

# When we're building modules with modversions, we need to consider

# the built-in objects during the descend as well, in order to

# make sure the checksums are up to date before we record them.

ifeq ($(MAKECMDGOALS),modules)

KBUILD\_BUILTIN := $(if $(CONFIG\_MODVERSIONS),1)

endif

顶层Makefile Line 264:   
export KBUILD\_MODULES KBUILD\_BUILTIN

[复制代码](javascript:void(0);)

 <2> builtin-target

scripts/Makefile.build Line 85:

ifneq ($(strip $(obj-y) $(obj-m) $(obj-n) $(obj-) $(lib-target)),)

builtin-target := $(obj)/built-in.o

endif

　　builtin-target值为drivers/built-in.o

 <3> lib-target

scripts/Makefile.build Line 81:

ifneq ($(strip $(lib-y) $(lib-m) $(lib-n) $(lib-)),)

lib-target := $(obj)/lib.a

endif

　　lib-target的值为drivers/lib.a

<4> $(if $(KBUILD\_MODULES),$(obj-m) $(modorder-target))

　　初步认识时定义编译的模块

 <5> subdir-ym

subdir-ym的定义

[复制代码](javascript:void(0);)

scripts/Makefile.lib Line 38:

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

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

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

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

scripts/Makefile.lib Line 47:   
subdir-ym := $(sort $(subdir-y) $(subdir-m))

scripts/Makefile.lib Line 89:   
subdir-ym := $(addprefix $(obj)/,$(subdir-ym))

[复制代码](javascript:void(0);)

　\_\_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。

scripts/Makefile.build Line 288:

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

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

<1> drivers/built-in.o的依赖

　　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：

[复制代码](javascript:void(0);)

arm-linux-ld -EL -r -o drivers/built-in.o drivers/gpio/built-in.o drivers/video/built-in.o

drivers/char/built-in.o drivers/gpu/built-in.o drivers/serial/built-in.o drivers/base/built-in.o

drivers/block/built-in.o drivers/misc/built-in.o drivers/mfd/built-in.o drivers/macintosh/built-in.o

drivers/scsi/built-in.o drivers/net/built-in.o drivers/ieee1394/built-in.o drivers/cdrom/built-in.o

drivers/auxdisplay/built-in.o drivers/mtd/built-in.o drivers/usb/built-in.o drivers/usb/gadget/built-in.o

drivers/input/built-in.o drivers/rtc/built-in.o drivers/i2c/built-in.o drivers/media/built-in.o

drivers/watchdog/built-in.o drivers/lguest/built-in.o drivers/idle/built-in.o drivers/mmc/built-in.o

drivers/firmware/built-in.o drivers/crypto/built-in.o drivers/hid/built-in.o drivers/platform/built-in.o

[复制代码](javascript:void(0);)

　　看下边的代码，原来drivers/Makefile中的obj-y是一群目录，通过第一行代码后就给这些目录后边加上了built-in.o，再经过第二行代码，给它们冠以前缀。我们以obj-y=net/为例说明，经过第一行代码obj-y=net/built-in.o，经过第二行代码obj-y=drivers/net/built-in.o

scripts/Makefile.lib Line 42:  
obj-y := $(patsubst %/, %/built-in.o, $(obj-y))

scripts/Makefile.lib Line 78:   
obj-y := $(addprefix $(obj)/,$(obj-y))

<2> if\_changed函数

　　通过if\_changed函数，完成了把这些子目录中的目标连接生成dirvers/built-in.o。

Kbuild.include Line 192:

if\_changed = $(if $(strip $(any-prereq) $(arg-check)), \

@set -e; \

$(echo-cmd) $(cmd\_$(1)); \

echo 'cmd\_$@ := $(make-cmd)' > $(dot-target).cmd)

　　既然是函数，就会传递参数。相对于C语言来说形参就是(1)，而传递实参的方法就是(1)，而传递实参的方法就是(call if\_changed,link\_o\_target)，其中call是调用函数命令，if\_changed是调用函数名，后边的link\_o\_target就是实参。

①$(if $(strip $(any-prereq) $(arg-check))  
any-prereq检查是否有依赖比目标新，或者依赖还没有创建；arg-check检查编译目标的命令相对上次是否发生变化。如果两者中只要有一个发生改变，if\_changed的值就等于

@set -e; \

$(echo-cmd) $(cmd\_$(1)); \

echo 'cmd\_$@ := $(make-cmd)' > $(dot-target).cmd

　　　否则，if\_changed的值为空，也就是说$(call if\_changed,link\_o\_target)将什么也不执行，因为没必要执行。

②set -e 是说下边的命令如果出错了就直接返回，不再继续执行

③echo-cmd

Kbuild.include Line 156:

echo-cmd = $(if $($(quiet)cmd\_$(1)),\

echo ' $(call escsq,$($(quiet)cmd\_$(1)))$(echo-why)';)

　　echo-cmd就是打印出调用的命令。if (((quiet)cmd\_(1))是判断命令(1))是判断命令(quiet)cmd\_(1)或者cmd\_(1)或者cmd\_(1)是否定义，也就是quiet\_cmd\_link\_o\_target或者cmd\_link\_o\_target是否定义。如果有定义，echo-cmd就会将这个命令打印出来，也就是打印。

④$(cmd\_$(1))

　　对于 $(call if\_changed,link\_o\_target)，该命令就是cmd\_link\_o\_target。

scripst/Makefile.build Line 283:

cmd\_link\_o\_target = $(if $(strip $(obj-y)),\

$(LD) $(ld\_flags) -r -o $@ $(filter $(obj-y), $^) \

$(cmd\_secanalysis),\

rm -f $@; $(AR) rcs $@)

$(LD) $(ld\_flags) -r -o $@ $(filter $(obj-y), $^) ,这个就是连接命令。  
  
总结：  
　　if\_changed函数的核心功能就是判断是否需要更新目标，如果需要就执行表达式$(cmd\_$(1))展开后的值来完成重建目标。

**4、drivers/net/built-in.o**  
 　　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。在 “2 scripts/Makefile.build阶段  2、scripts/Makefile.build的总目标 <5> $(subdir-ym)”中，我们假定subdir-ym=drivers/net。

Makefile.build Line 356:   
PHONY += $(subdir-ym)

$(subdir-ym):

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

　　$(Q)$(MAKE) $(build)=$@就是再一次调用scripts/Makefile.build，并传递参数drivers/net，这和生成drivers/built-in.o没什么差别。

**3 scripts/Makefile.build的再一次调用阶段**

再次转战到scripts/Makefile.build，obj=drivers/net被带到该文件。

**1、Makefile.build的包含文件**

　　其包含文件与第一次调用阶段没什么大的区别，最为重要的区别就是包含子目录Makefile的改变，这次包含的文件是/workspace/linux-2.6.30.4/drivers/net/Makefile。

**2、scripts/Makefile.build的总目标**

　总目标中的builtin-target值为drivers/net/built-in.o

**3、drivers/net/built-in.o的生成**

scripts/Makefile.build Line 288:

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

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

　　由于子目录的Makefile变成了/workspace/linux-2.6.30.4/drivers/net/Makefile，所以obj-y对应的发生了变化。通过执行编译查看这个连接过程。

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/dm9000.o drivers/net/arm/built-in.o drivers/net/e1000/built-in.o

<1> drivers/net/built-in.o的依赖

　　从drivers/net/Makefile中摘出两个典型的obj-y组成部分如下所示：

drivers/net/Makefile Line 5:

obj-$(CONFIG\_E1000) += e1000/

drivers/net/Makefile Line 230:   
obj-$(CONFIG\_DM9000) += dm9000.o

　　drivers/net/built-in.o的依赖包含了文件夹，并且包含了直接由C文件生成的目标文件.o。

scripts/Makefile.lib Line 42:

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

scripts/Makefile.lib Line 78:

obj-y := $(addprefix $(obj)/,$(obj-y))

　　第一行代码让obj-y如果是文件夹，就添加built-in.o，如果是普通的目标文件\*.o，什么也不操作；第二行代码冠以目录obj。所以obj-y最终等于drivers/net/e1000/built-in.o、drivers/net/dm9000.o。

　　我们越来越接近最终的目标了--分析到最底层的C文件生成目标文件。

<2> if\_changed函数

　　drivers/net/built-in.o也是一个由众多的文件组成的库文件，所以也是通过调用if\_changed函数完成连接。

**4、drivers/net/dm9000.o**

[复制代码](javascript:void(0);)

scripts/Makefile.build Line 226:

# Built-in and composite module parts

$(obj)/%.o: $(src)/%.c FORCE

$(call cmd,force\_checksrc)

$(call if\_changed\_rule,cc\_o\_c)

scripts/Makefile.build Line 263:

$(obj)/%.o: $(src)/%.S FORCE

$(call if\_changed\_dep,as\_o\_S)

[复制代码](javascript:void(0);)

$(obj)/%.o: $(src)/%.c FORCE是编译C程序，$(obj)/%.o: $(src)/%.S FORCE是编译汇编程序

<1> drivers/net/dm9000.o的依赖

　　drivers/net/dm9000.o的依赖是对应的$(src)/%.c，也就是drivers/net/dm9000.c。

<2>cmd函数

scripts/Kbuild Line 160:

cmd = @$(echo-cmd) $(cmd\_$(1))

　　此函数还未做深入分析。

<2>if\_changed\_rule函数

[复制代码](javascript:void(0);)

scripts/Kbuild Line 205:

# Usage: $(call if\_changed\_rule,foo)

# Will check if $(cmd\_foo) or any of the prerequisites changed,

# and if so will execute $(rule\_foo).

if\_changed\_rule = $(if $(strip $(any-prereq) $(arg-check) ), \

@set -e; \

$(rule\_$(1)))

[复制代码](javascript:void(0);)

　　$(call if\_changed\_rule,cc\_o\_c)，从而会调用函数if\_changed\_rule，接着会执行命令$(rule\_$(1)))，也就是rule\_cc\_o\_c。

[复制代码](javascript:void(0);)

scripts/Makefile.build Line 215:

define rule\_cc\_o\_c

$(call echo-cmd,checksrc) $(cmd\_checksrc) \

$(call echo-cmd,cc\_o\_c) $(cmd\_cc\_o\_c); \

$(cmd\_modversions) \

$(cmd\_record\_mcount) \

scripts/basic/fixdep $(depfile) $@ '$(call make-cmd,cc\_o\_c)' > \

$(dot-target).tmp; \

rm -f $(depfile); \

mv -f $(dot-target).tmp $(dot-target).cmd

endef

[复制代码](javascript:void(0);)

　　其中$(cmd\_cc\_o\_c)命令的定义是

scripts/Makefile.build Line 179:   
cmd\_cc\_o\_c = $(CC) $(c\_flags) -c -o $@ $<

**5、drivers/net/e1000/built-in.o的生成**

　　subdir-ym肯定包含有文件夹drivers/net/e1000，于是继续再一次调用scripts/Makefile.build来完成drivers/net/e1000/built-in.o的生成。

Makefile.build Line 356:

PHONY += $(subdir-ym)

$(subdir-ym):

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

 **三、Linux内核整体编译过程**

　经过解析过程的分析，编译过程就是解析过程的相反操作。

**1、生成准备文件**

①控制C程序的头文件

　　include/linux/version.h include/linux/utsrelease.h、include/linux/autoconf.h

②控制编译连接的文件

　　arch/arm/kernel/vmlinux.lds、include/config/auto.conf等文件。

**2、 由C程序源码和汇编语言源码生成目标文件（\*.o）**

**3、将目标文件连接成\*.built-in.o、\*/lib.a等文件**

**4、将紧接着顶层目录的子目录中的\*.built-in.o以及部分重要的\*.o文件连接生成vmlinux**

**5、根据arch/arm/Makefile的规则生成zImage、uImage等**

**四、Linux内核编译构成元素**

**1、Makefile的目标**

**<1> 总目标**

　　总目标实际上是在arch/arm/Makefile中定义了，比方说zImage、uImage，顶层Makefile紧接着定义了这些终极目标直接的依赖目标vmlinux。

**<2> 各级子目标**

　　各级子目标是在scripts/Makefile.build中的\_\_build中定义的，例如传递参数obj=drivers后的目标是drivers/built-in.o。

　　这些目标的依赖其实又成为了新的目标，例如drivers/net/built-in.o、drivers/net/dm9000.o。

**2、Makefile的依赖**

**<1> 总目标的依赖**

vmlinux-lds  vmlinux-init vmlinux-main vmlinux.o kallsyms.o

**<2> 各级子目标的依赖**

　　各级子目标的依赖是由子目录中的Makefile（实际是scripts/Makefile.build的包含文件）和scripts/Makefile.lib共同完成确定的。

　　子目录中的Makefile负责选材，而scripts/Makefile.lib负责加工。

**3、Makefile的规则**

**<1> 总目标的连接规则**

　　总目标vmlinux的连接规则就是在顶层Makefile中定义的，至于zImage、uImage则是在arch/arm/Makefile中定义的。

**<2>子目标的编译连接规则**

　　主要是在scripts/Makefile.build、scripts/Kbuild.include中定义的，其中scripts/Kbuild.include定义了许多诸如if\_changed的函数。

**4、Makefile的编译连接选项**

本文并没有讨论。

**五、Linux内核Makefile特点**

**1、两个Makefile**

顶层Makefile文件负责将各个目录生成的\*.built-in.o、lib.a等文件连接到一起生成vmlinux。而scripts/Makefile.build 包含子目录中的Makefile文件以及scripts/中的众多脚本来生成这些\*.built-in.o、lib.a、\*.o等文件。

　　通过“make -f scripts/Makefile.build obj=”的方法完成了顶层Makefile到scripts/Makefile.build的调用生成\*/built-in.o，以及scripts/Makefile.build的自调用生成更低一级子目录的\*/built-in.o。

**2、编译的目录始终是顶层目录**

　　“make -C”命令会先切换工作目录，然后执行该目录中的Makefile，u-boot就是采用这种方法。而linux则是利用“make -f”的方法，所以编译的目录始终是顶层目录。

**3、通用规则**

　　Linux内核Makefile的通用子Makefile是scripts/Makefile.build，而通用的其他规则则是scripts中的其他文件。

参考资料：《深度探索Linux操作系统：系统构建和原理解析》