linux 顶层Makefile执行的流程！！！(转）_jxywxy_新浪博客

1、make menuconfig

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 26
EXTRAVERSION =
NAME = Rotary Wombat

# *DOCUMENTATION*
# To see a list of typical targets execute "make help"
# More info can be located in ./README
# Comments in this file are targeted only to the developer, do not
# expect to learn how to build the kernel reading this file.

# Do not:
# o  use make's built-in rules and variables
#    (this increases performance and avoids hard-to-debug behaviour);
# o  print "Entering directory ...";
MAKEFLAGS += -rR --no-print-directory
#-r禁止使用build-in规则
#--no-print-directory是:不要再屏幕上打印"Entering directory.."
#记住变量SHELL,MAKEFLAGS在整个make的执行过程中
#始终被自动的传递给所有的子make

# We are using a recursive build, so we need to do a little thinking
# to get the ordering right.
#
# Most importantly: sub-Makefiles should only ever modify files in
# their own directory. If in some directory we have a dependency on
# a file in another dir (which doesn't happen often, but it's often
# unavoidable when linking the built-in.o targets which finy
# turn into vmlinux), we will call a sub make in that other dir, and
# after that we are sure that everything which is in that other dir
# is now up to date.
#
# The only cases where we need to modify files which have global
# effects are thus separated out and done before the recursive
# descending is started. They are now explicitly listed as the
# prepare rule.

# To put more focus on warnings, be less verbose as default
# Use 'make V=1' to see the full commands

ifdef V   #v=1
  ifeq ("$(origin V)", "command line")
    KBUILD_VERBOSE = $(V)   #把V的值作为KBUILD_VERBOSE的值
  endif
endif

ifndef KBUILD_VERBOSE  #即默认我们是不回显的
```

```
    #回显即在命令执行前显示要执行的命令
  KBUILD_VERBOSE = 0
endif
# 函数origin并不操作变量的值，只是告诉你你的这个变量是哪里来的。
# 语法是： $(origin ;)# origin函数的返回值有：
#"undefined"从来没有定义过、“default”是一个默认的定义、“
#"environment"是一个环境变量、
#"file"这个变量被定义在Makefile中
#"command line"这个变量是被命令行定义的
#"override"是被override指示符重新定义的
#"automatic"是一个命令运行中的自动化变量


# Call a source code checker (by default, "sparse") as part of the
# C compilation.调用一个静态分析工具作为c编译器的
#部分
# Use 'make C=1' to enable checking of only re-compiled files.
# Use 'make C=2' to enable checking of *all* source files, regardless
# of whether they are re-compiled or not.
#
# See the file "Documentation/sparse.txt" for more details, including
# where to get the "sparse" utility.
ifdef C
  ifeq ("$(origin C)", "command line")
    KBUILD_CHECKSRC = $(C)
  endif
endif
ifndef KBUILD_CHECKSRC
  KBUILD_CHECKSRC = 0
endif


#用M来指定外部模块的目录
# Use make M=dir to specify directory of external module to build
# Old syntax make ... SUBDIRS=$PWD is still supported
# Setting the environment variable KBUILD_EXTMOD take precedence
#识别这里是定义一个外部模块吗??
#当我们定义了M变量或者SUBDIRS时表示编译一个外部模块

ifdef SUBDIRS
  KBUILD_EXTMOD ?= $(SUBDIRS) #?=为条件赋值操作符仅仅在变量还
  #还没有定义的情况下有效
endif
ifdef M
  ifeq ("$(origin M)", "command line")
    KBUILD_EXTMOD := $(M)
  endif
endif

# kbuild supports saving output files in a separate directory.
# To locate output files in a separate directory two syntaxes are supported.
# In both cases the working directory must be the root of the kernel src.
# 1) O=
# Use "make O=dir/to/store/output/files/"
```

```
#
# 2) Set KBUILD_OUTPUT
# Set the environment variable KBUILD_OUTPUT to point to the directory
# where the output files shall be placed.
# export KBUILD_OUTPUT=dir/to/store/output/files/
# make
#
# The O= assignment takes precedence over the KBUILD_OUTPUT environment
# variable.

# KBUILD_SRC is set on invocation of make in OBJ directory
# KBUILD_SRC is not intended to be used by the regular user (for now)
####################make 的最开始###################
#########################################################
ifeq ($(KBUILD_SRC),)#如果KBUILD_SRC没有定义则进入下面一层即首次进入
        #时一般都需要两次进入顶层Makefile
        #若定义了则不为空直接跳入下一步
# OK, Make called in directory where kernel src resides
# Do we want to locate output files in a separate directory?
ifdef O #把输出文件放在不同的文件夹内
  ifeq ("$(origin O)", "command line")
    KBUILD_OUTPUT := $(O)  #用于指定我们的输出文件的输出目录
  endif
endif

# That's our default target when none is given on the command line
PHONY := _all
_all:

# Cancel implicit rules on top Makefile取消隐式推倒规则
$(CURDIR)/Makefile Makefile: ;#remake 操作查看当前目录Makefile是否为最新
#CURDIR值为当前的内核源码目录current dir
#为GNU make使用的变量,CURDIR设置当前工作目录的路径名

ifneq ($(KBUILD_OUTPUT),) #如果输出目录不为空即设置了输出目录则检测
# Invoke a second make in the output directory, passing relevant variables
# check that the output directory actually exists
saved-output := $(KBUILD_OUTPUT)
KBUILD_OUTPUT := $(shell cd $(KBUILD_OUTPUT) && /bin/pwd)#这里是测试此目录是
否
          #存在,若存在则赋给K BUILD_OUTPUT
$(if $(KBUILD_OUTPUT),, /#这里的为空即表示输出目录不存在
    $(error output directory "$(saved-output)" does not exist))#使用了error函数
#这里的if函数语法为:if,
#或者是if,,

#make的环境变量叫MAKECMDGOALS 这个变量中会存放你所指定的
#终极目标的列表，如果在命令行上，你没有指定目标，
#那么，这个变量是空值。
PHONY += $(MAKECMDGOALS) sub-make
#将任何在命令行中给出的目标放入变量MAKECMDGOALS
#这里filter-out是返回$(MAKECMDGOALS)中除了Makefile _all sub-make以外的
#其他文件;
```

```
$(filter-out _all sub-make $(CURDIR)/Makefile, $(MAKECMDGOALS)) _all: sub-make
 $(Q)@:#这里仅仅只是取消回显没有别的意思


#$(KBUILD_VERBOSE:1=)为变量替换，如果为1就替换为空
#if KBUILD_VERBOSE = 1 or KBUILD_VERBOSE is NULL
#then
#$(MAKE) -C $(KBUILD_OUTPUT) //回显
#else
#@ $(MAKE) -C $(KBUILD_OUTPUT)//即不回显
#endif
sub-make: FORCE
 $(if $(KBUILD_VERBOSE:1=),@)$(MAKE) -C $(KBUILD_OUTPUT)
/#KBUILD_OUTPUT==dir前面测试了
 KBUILD_SRC=$(CURDIR) /
 KBUILD_EXTMOD="$(KBUILD_EXTMOD)" -f $(CURDIR)/Makefile /#
 $(filter-out _all sub-make,$(MAKECMDGOALS))#这里是表示要生成的目标
#make -C dir KBUILD_SRC=`pwd` KBUILD_EXTMOD="" -f `pwd`/Makefile [Targets]
#这里表示再次执行Makefile 但此时我们的KBUILD
#此时再次执行完make后skip-makefile := 1从而
# Leave processing to above invocation of make结束子make返回根目录的make
skip-makefile := 1
endif # ifneq ($(KBUILD_OUTPUT),)这里是ifneq
endif # ifeq ($(KBUILD_SRC),)#这里是ifeq的结束处
########################################################
########################################################

ifeq ($(skip-makefile),)#如果为空则开始分类执行!!!!!!
#endif # skip-makefile在1813行处结束最大的ifeq结构
# If building an external module we do not care about the all: rule
# but instead _all depend on modules
PHONY += all      #声明一个伪目标all
ifeq ($(KBUILD_EXTMOD),) #如果外部模块定义为空则_all依赖于all
_all: all
else
_all: modules                #否则依赖于modules
endif

srctree  := $(if $(KBUILD_SRC),$(KBUILD_SRC),$(CURDIR))#看KBUILD_SRC是否为空,
        #设置源码目录
TOPDIR  := $(srctree)   #顶层目录
# FIXME - TOPDIR is obsolete, use srctree/objtree
objtree  := $(CURDIR)   #CURDIR为make的默认环境变量

src  := $(srctree) #源文件的目录也设置为当前目录
obj  := $(objtree)#目标文件的输出目录设置当前目录

VPATH  := $(srctree)$(if $(KBUILD_EXTMOD),:$(KBUILD_EXTMOD))
#这里是说当前目录找不到源文件时的搜索目录
export srctree objtree VPATH TOPDIR
#export用于要将指定变量输出给子make，
```

```
# SUBARCH tells the usermode build what the underlying arch is.  That is set
# first, and if a usermode build is happening, the "ARCH=um" on the command
# line overrides the setting of ARCH below.  If a native build is happening,
# then ARCH is assigned, getting whatever value it gets normally, and
# SUBARCH is subsequently ignored.
#这里是获取cpu的具体型号存在变量SUBARCH中
#sed -e s表示替换,这里表示出现i.86的用i386替换
#出现sun4u的用sparc64替换
#一般写成sed -e 's/i.86/i386' 用"括起来
SUBARCH := $(shell uname -m | sed -e s/i.86/i386/ -e s/sun4u/sparc64/ /
    -e s/arm.*/arm/ -e s/sa110/arm/ /
    -e s/s390x/s390/ -e s/parisc64/parisc/ /
    -e s/ppc.*/powerpc/ -e s/mips.*/mips/ /
    -e s/sh.*/sh/ )

# Cross compiling and selecting different set of gcc/bin-utils
# ---------------------------------------------------------------------------
#
# When performing cross compilation for other architectures ARCH shall be set
# to the target architecture. (See archMakefile
export KBUILD_BUILDHOST := $(SUBARCH) #这里是把宿主机的cpu结构导出到
        #KBUILD_BUILDHOST变量,并传入子make
ARCH  ?= $(SUBARCH)
# 设置变量 ARCH 的方法有两种，
# 一：在命令行中 如： make ARCH=ia64 ；
# 二：设置环境变量，在环境变量中默认
#的 ARCH 的值是执行 make 的 cpu 架构
#   "?="表示为ARCH 条件赋值如果ARCH前面没有赋值则这里赋值
#成功,否则不重新赋值
CROSS_COMPILE ?=                #交叉编译工具的设置在嵌入式系统
       #经常要修改此处设置交叉编译器的路径

# Architecture as present in compile.h
UTS_MACHINE  := $(ARCH)
SRCARCH  := $(ARCH)

# Additional ARCH settings for x86
ifeq ($(ARCH),i386)
       SRCARCH := x86
endif

ifeq ($(ARCH),x86_64)
       SRCARCH := x86
endif

KCONFIG_CONFIG ?= .config  #这里是生成的配置文件

# SHELL used by kbuild 这里shel中的if [ -x file ]测试file是否可执行
CONFIG_SHELL := $(shell if [ -x "$$BASH" ]; then echo $$BASH; /
  else if [ -x /bin/bash ]; then echo /bin/bash; /
  else echo sh; fi ; fi)

HOSTCC       = gcc
HOSTCXX      = g++
HOSTCFLAGS   = -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer
```

```
HOSTCXXFLAGS = -O2

# Decide whether to build built-in, modular, or both.
# Normally, just do built-in.
KBUILD_MODULES :=
KBUILD_BUILTIN := 1

# If we have only "make modules", don't compile built-in objects.编译内置对象
# 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.

#如果执行”make modules”则这里在这里重新处理KBUILD_BUILTIN :=1
ifeq ($(MAKECMDGOALS),modules)
  KBUILD_BUILTIN := $(if $(CONFIG_MODVERSIONS),1)
endif


# If we have "make modules", compile modules
# in addition to whatever we do anyway.
# Just "make" or "make all" shall build modules as well


#如果执行"make all","make _all","make modules","make"中任一个命令
#则在这里重新处理KBUILD_MODULES
ifneq ($(filter all _all modules,$(MAKECMDGOALS)),)#filter过滤出指定的字符串
        #这里表示如果不为空则编译模块
  KBUILD_MODULES := 1
endif

ifeq ($(MAKECMDGOALS),)
  KBUILD_MODULES := 1
endif

export KBUILD_MODULES KBUILD_BUILTIN
#导出变量给子make
export KBUILD_CHECKSRC KBUILD_SRC KBUILD_EXTMOD


# Beautify output  格式化的回显控制
# ---------------------------------------------------------------------------
#
# Normally, we echo the whole command before executing it. By making
# that echo $($(quiet)$(cmd)), we now have the possibility to set
# $(quiet) to choose other forms of output instead, e.g.
#
#         quiet_cmd_cc_o_c = Compiling $(RELDIR)/$@
#         cmd_cc_o_c       = $(CC) $(c_flags) -c -o $@ $<<BR>#
# If $(quiet) is empty, the whole command will be printed.表示如果为空则完全回显
# If it is set to "quiet_", only the short version will be printed. $(quiet)=quiet_表示只有版本显
示
# If it is set to "silent_", nothing will be printed at all, since如果$(quiet)=silent表示完全不回显
# the variable $(silent_cmd_cc_o_c) doesn't exist.
#
# A simple variant is to prefix commands with $(Q) - that's useful..$(Q)用作前缀,如果$(Q)=@
```

表示
for commands that shall be hidden in non-verbose mode. 不回显,为空表示回显
#
$(Q)ln $@ :<
#
If KBUILD_VERBOSE equals 0 then the above command will be hidden.
If KBUILD_VERBOSE equals 1 then the above command is displayed.

ifeq ($(KBUILD_VERBOSE),1) #这里就是定义是否要回显
  quiet =
  Q =   #不以@开头表示回显
else
  quiet=quiet_
  Q = @   #这里以@开头表示不回显
endif

If the user is running make -s (silent mode), suppress echoing of
commands

ifneq ($(findstring s,$(MAKEFLAGS)),)
#如果MAKEFLAGS中带了-s表示完全不要回显
  quiet=silent_
endif

export quiet Q KBUILD_VERBOSE #导出子make

Look for make include files relative to root of kernel src
MAKEFLAGS += --include-dir=$(srctree)

We need some generic definitions (do not try to remake the file).
$(srctree)/scripts/Kbuild.include: ;
#这里执行remake操作请看GNU make 流程
include $(srctree)/scripts/Kbuild.include #这里是文件包含
#Kbuild.include 这里定义了大量通用的函数和变量
#重点!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Make variables (CC, etc...)
#这里是设置编译连接的默认程序,都是变量赋值操作
AS = $(CROSS_COMPILE)as
LD = $(CROSS_COMPILE)ld
CC = $(CROSS_COMPILE)gcc
CPP = $(CC) -E
AR = $(CROSS_COMPILE)ar
NM = $(CROSS_COMPILE)nm
STRIP = $(CROSS_COMPILE)strip
OBJCOPY = $(CROSS_COMPILE)objcopy
OBJDUMP = $(CROSS_COMPILE)objdump
AWK = awk
GENKSYMS = scripts/genksyms/genksyms
DEPMOD = /sbin/depmod
KALLSYMS = scripts/kallsyms
PERL = perl
CHECK = sparse #默认的字符串检查

CHECKFLAGS   := -D__linux__ -Dlinux -D__STDC__ -Dunix -D__unix__ -Wbitwise $(CF)

```
MODFLAGS = -DMODULE
CFLAGS_MODULE   = $(MODFLAGS)
AFLAGS_MODULE   = $(MODFLAGS)
LDFLAGS_MODULE  =
CFLAGS_KERNEL =
AFLAGS_KERNEL =


# Use LINUXINCLUDE when you must reference the include/ directory.
# Needed to be compatible with the O= option
LINUXINCLUDE    := -Iinclude /
                $(if $(KBUILD_SRC),-Iinclude2 -I$(srctree)/include) /
    -include include/linux/autoconf.h

KBUILD_CPPFLAGS := -D__KERNEL__ $(LINUXINCLUDE)

KBUILD_CFLAGS   := -Wall -Wundef -Wstrict-prototypes -Wno-trigraphs /
    -fno-strict-aliasing -fno-common /
    -Werror-implicit-function-declaration
KBUILD_AFLAGS   := -D__ASSEMBLY__

# Read KERNELRELEASE from include/config/kernel.release (if it exists)
KERNELRELEASE = $(shell cat include/config/kernel.release 2> /dev/null)#
KERNELVERSION = $(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)

export VERSION PATCHLEVEL SUBLEVEL KERNELRELEASE KERNELVERSION
export ARCH SRCARCH CONFIG_SHELL HOSTCC HOSTCFLAGS CROSS_COMPILE AS
LD CC
export CPP AR NM STRIP OBJCOPY OBJDUMP MAKE AWK GENKSYMS PERL
UTS_MACHINE
export HOSTCXX HOSTCXXFLAGS LDFLAGS_MODULE CHECK CHECKFLAGS

export KBUILD_CPPFLAGS NOSTDINC_FLAGS LINUXINCLUDE OBJCOPYFLAGS
LDFLAGS
export KBUILD_CFLAGS CFLAGS_KERNEL CFLAGS_MODULE
export KBUILD_AFLAGS AFLAGS_KERNEL AFLAGS_MODULE

# When compiling out-of-tree modules, put MODVERDIR in the module
# tree rather than in the kernel tree. The kernel tree might
# even be read-only.
export MODVERDIR := $(if $(KBUILD_EXTMOD),$(firstword
$(KBUILD_EXTMOD))/).tmp_versions

# Files to ignore in find ... statements

RCS_FIND_IGNORE := /( -name SCCS -o -name BitKeeper -o -name .svn -o -name CVS -o
-name .pc -o -name .hg -o -name .git /) -prune -o
export RCS_TAR_IGNORE := --exclude SCCS --exclude BitKeeper --exclude .svn --exclude
CVS --exclude .pc --exclude .hg --exclude .git

#
# Rules shared between *config targets and build targets

# Basic helpers built in scripts这里生成基本的scripts配置工具fixdep docproc两个/
PHONY += scripts_basic   #定义一个伪目标
scripts_basic: #该伪目标要执行的操作
 $(Q)$(MAKE) $(build)=scripts/basic
```

```
#build := -f $(if $(KBUILD_SRC),$(srctree)/)scripts/Makefile.build obj 定义在Kbuild.include
中我们在前面
#包含了该文件
#src := $(obj)因为我们Makefile.build中需要带入参数obj用于赋给src
#这里会执行.../scripts/Makefile.build 文件==========================
#Makefile.build中包含了scripts/basic/Makefile

# To avoid any implicit rule to kick in, define an empty command.
scripts/basic/%: scripts_basic ;

PHONY += outputmakefile
# outputmakefile generates a Makefile in the output directory, if using a
# separate output directory. This allows convenient use of make in the
# output directory.

outputmakefile:
ifneq ($(KBUILD_SRC),)  #$(CONFIG_SHELL)即被设置为/bin/sh或者其它shell
 $(Q)$(CONFIG_SHELL) $(srctree)/scripts/mkmakefile /#这里的用shell执行mkmakefile
    $(srctree) $(objtree) $(VERSION) $(PATCHLEVEL) #并在输出目录中输出
#这个规则的命令运行一个shell脚本scripts/mkmakefile，
#并传递四个参数。这个脚本主要是在$(objtree)参数指定的目录中
#生成一个Makefile文件。由于这里KBUILD_SRC为空，
#所以这个脚本并不会被执行
endif


# To make sure we do not include .config for any of the *config targets
# catch them early, and hand them over to scripts/kconfig/Makefile
# It is allowed to specify more targets when calling make, including
# mixing *config targets and build targets.
# For example 'make oldconfig all'.
# Detect when mixed targets is specified, and make a second invocation
# of make so .config is not included in this case either (for *config).

#GNU configure 中的build/target和host的区别
#build就是现在使用的机器,即编译平台
#host就是我们的目标机,编译好的程序运行的平台
#target一般在构建编译本身的时候(gcc),采用target
no-dot-config-targets := clean mrproper distclean /
   cscope TAGS tags help %docs check% /
   include/linux/version.h headers_% /
   kernelrelease kernelversion

#这里是默认值,
config-targets := 0#如果有config %config则为1
mixed-targets  := 0#当有config %config 还有其它目标时为1
dot-config     := 1  #当有config %config时肯定为1
#根据输出参数修改

#make后面的目标包括三类:
#(1)和.config无关的目标即(no_dot_config-targets)比如:clean,mrproper,distclean,
#     headers_install,kernelrelease,kernelversion等等；
#(2)和.config相关的目标dot_config
```

```
#    a:配置目标(config targets)主要是产生.config文件
#    b:构建目标(build targets)主要是使用.config像all,vmlinux,modules,zImage,等等
#(3)single targets
#    上面没有列出的,如:dir/,dir/file，dir/file.ko


ifneq ($(filter $(no-dot-config-targets), $(MAKECMDGOALS)),)#如果我们命令行目标
CMDGOALS
          #有任何clean之类的命令
 ifeq ($(filter-out $(no-dot-config-targets), $(MAKECMDGOALS)),)#且除了clean之类目标外
为空
         #那么标识应该删除.config文件
  dot-config := 0
 endif
endif


ifeq ($(KBUILD_EXTMOD),)
      ifneq ($(filter config %config,$(MAKECMDGOALS)),)#这里就是匹配我们命令行
           #中是否有menuconfig/xconfig等有则表示
           #这是一条配置命令,因而config-targets置1
           #标识应该生成.config文件
            config-targets := 1
            ifneq ($(filter-out config %config,$(MAKECMDGOALS)),)#如果不仅有config目标
                     #还有别的目标,表示为混合目标模式
                mixed-targets := 1
            endif
      endif
endif

# ============混合模式则则执行以下规则===========
ifeq ($(mixed-targets),1) #如果是混合模式则则执行以下规则
# We're called with mixed targets (*config and build targets).
# Handle them one by one.
#%代表make 后带任何目标都执行该命令
#那就是
%:: FORCE
 $(Q)$(MAKE) -C $(srctree) KBUILD_SRC= $@
#从代码中可以看出，这里使用了一个双冒号的模式匹配规则。
#百分号代表任何目标都使用这个规则，其中
#$(srctree) 为内核代码树所在目 录 ， KBUILD_SRC 定义为空 。
#所以如果 make 命令为 ： make s3c2410_defconfig all
#那么构建系统就会分别执行下面两条命令：
#make -C $(srctree) KBUILD_SRC= s3c2410_defconfig
#make -C $(srctree) KBUILD_SRC= all
#这其实和简单的用手动的输入两条连续命令
#(make s3c2410_defconfig 和 make all) 是一样效果的。

# ============%config targe配置时执行以下规则===========
#这里是%config target时执行的规则!!!!!!!!!!!!!!!!!!!!!!!!!!
else
ifeq ($(config-targets),1)
```

```
#
# ==========================================================================
# *config targets only - make sure prerequisites are updated, and descend
# in scripts/kconfig to make the *config target

# Read arch specific Makefile to set KBUILD_DEFCONFIG as needed.
# KBUILD_DEFCONFIG may point out an alternative default configuration
# used for 'make defconfig'
include $(srctree)/arch/$(SRCARCH)/Makefile#直接包含平台架构下
        #的makefile
export KBUILD_DEFCONFIG

###############执行make menuconfig的入口点####################
#生配置工具,根据我们的kconfig
config %config: scripts_basic outputmakefile FORCE
 $(Q)mkdir -p include/linux include/config
 $(Q)$(MAKE) $(build)=scripts/kconfig $@
#在顶层目录执行”make menuconfig”会执行顶层Makefile 第415行的规则
#config %config: scripts_basic outputmakefile FORCE
#$(Q)mkdir -p include/linux include/config
#$(Q)$(MAKE) $(build)=scripts/kconfig $@
#这里"menuconfig"与模式"%config”匹配。所以其执行的规则如下：

#menuconfig: scripts_basic outputmakefile FORCE
#$(Q)mkdir -p include/linux include/config
#$(Q)$(MAKE) $(build)=scripts/kconfig menuconfig

#$@目标变量
#build := -f $(if $(KBUILD_SRC),$(srctree)/)scripts/Makefile.build obj
#之所以我们引用$(build)=...每次都要是build带一个等于号的
#的原因就是因为我们的变量后面带了一个obj用于指定
#我们的obj目录是哪里!!!!!!!!!!!
#这个规则有三个依赖：scripts_basic、outputmakefile、FORCE


#PHONY += scripts_basic   #定义一个伪目标
#scripts_basic: #该伪目标要执行的操作
# $(Q)$(MAKE) $(build)=scripts/basic
#build := -f $(if $(KBUILD_SRC),$(srctree)/)scripts/Makefile.build obj
#之所以我们引用$(build)=...每次都要是build带一个等于号的
#的原因就是因为我们的变量后面带了一个obj用于指定
#我们的obj目录是哪里!!!!!!!!!!!

#============================================
#============================================
#这个命令依然是执行scripts/Makefile.build这个makefile文件。
#并执行它里面$@表示menuconfig的规则。根据上面的分析，
#在Makefile.build会包含scripts/kconfig/Makefile文件!!!!!!!!!!!!。
#然后执行以menuconfig为目标的规则，
#在scripts/kconfig/Makefile的13行定义
#menuconfig: $(obj)/mconf
 #      $< arch/$(ARCH)/Kconfig
#即$<表示依赖集$(obj)/mconf这个是一个解析Kconfig的程序
```

```
#arch/$(ARCH)/Kconfig 是被解析的文件

# ===============编译模块或者目标时目标=========================
# Build targets only - this includes vmlinux, arch specific targets, clean
# targets and others. In general all targets except *config targets.
else
ifeq ($(KBUILD_EXTMOD),) #
# Additional helpers built in scripts/
# Carefully list dependencies so we do not try to build scripts twice
# in parallel
PHONY += scripts
scripts: scripts_basic include/config/auto.conf
 $(Q)$(MAKE) $(build)=$(@)
#这里生成scripts_basic include/config/auto.conf

# Objects we will link into vmlinux / subdirs we need to visit
init-y  := init/
drivers-y := drivers/ sound/
net-y  := net/
libs-y  := lib/
core-y  := usr/
endif # KBUILD_EXTMOD


ifeq ($(dot-config),1)#若.config已经被配置,则可包含产生的auto.conf文件
# Read in config
-include include/config/auto.conf

ifeq ($(KBUILD_EXTMOD),)
# Read in dependencies to all Kconfig* files, make sure to run
# oldconfig if changes are detected.
-include include/config/auto.conf.cmd

# To avoid any implicit rule to kick in, define an empty command
$(KCONFIG_CONFIG) include/config/auto.conf.cmd: ;

# If .config is newer than include/config/auto.conf, someone tinkered
# with it and forgot to run make oldconfig.
# if auto.conf.cmd is missing then we are probably in a cleaned tree so
# we execute the config step to be sure to catch updated Kconfig files
#这里是生成我们的需要的三个文件
include/config/auto.conf: $(KCONFIG_CONFIG) include/config/auto.conf.cmd
 $(Q)$(MAKE) -f $(srctree)/Makefile silentoldconfig
else
# external modules needs include/linux/autoconf.h and include/config/auto.conf
# but do not care if they are up-to-date. Use auto.conf to trigger the test
PHONY += include/config/auto.conf

include/config/auto.conf:
 $(Q)test -e include/linux/autoconf.h -a -e $@ || (  /
 echo;       /
 echo "  ERROR: Kernel configuration is invalid.";  /
 echo "         include/linux/autoconf.h or $@ are missing."; /
 echo "         Run 'make oldconfig && make prepare' on kernel src to fix it."; /
 echo;       /
 /bin/false)
endif # KBUILD_EXTMOD
```

```
else
# Dummy target needed, because used as prerequisite
include/config/auto.conf: ;
endif # $(dot-config)

#==============the read targets starts======================
#==============vmlinux all modules and other targets ============
# The all: target is the default when no target is given on the
# command line.
# This allow a user to issue only 'make' to build a kernel including modules
# Defaults vmlinux but it is usually overridden in the arch makefile
all: vmlinux    #这个是默认的目标

ifdef CONFIG_CC_OPTIMIZE_FOR_SIZE
KBUILD_CFLAGS += -Os
else
KBUILD_CFLAGS += -O2
endif

ifneq (CONFIG_FRAME_WARN,0)
KBUILD_CFLAGS += $(call cc-option,-Wframe-larger-than=${CONFIG_FRAME_WARN})
endif

# Force gcc to behave correct even for buggy distributions
# Arch Makefiles may override this setting
KBUILD_CFLAGS += $(call cc-option, -fno-stack-protector)

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

ifdef CONFIG_FRAME_POINTER
KBUILD_CFLAGS += -fno-omit-frame-pointer -fno-optimize-sibling-calls
else
KBUILD_CFLAGS += -fomit-frame-pointer
endif

ifdef CONFIG_DEBUG_INFO
KBUILD_CFLAGS += -g
KBUILD_AFLAGS += -gdwarf-2
endif

# We trigger additional mismatches with less inlining
ifdef CONFIG_DEBUG_SECTION_MISMATCH
KBUILD_CFLAGS += $(call cc-option, -fno-inline-functions-called-once)
endif

# arch Makefile may override CC so keep this after arch Makefile is included
NOSTDINC_FLAGS += -nostdinc -isystem $(shell $(CC) -print-file-name=include)
CHECKFLAGS     += $(NOSTDINC_FLAGS)

# warn about C99 declaration after statement
KBUILD_CFLAGS += $(call cc-option,-Wdeclaration-after-statement,)

# disable pointer signed / unsigned warnings in gcc 4.0
KBUILD_CFLAGS += $(call cc-option,-Wno-pointer-sign,)

# Add user supplied CPPFLAGS, AFLAGS and CFLAGS as the last assignments
# But warn user when we do so
warn-assign = /
$(warning "WARNING: Appending $$K$(1) ($(K$(1))) from $(origin K$(1)) to kernel $$$(1)")
```

```
ifneq ($(KCPPFLAGS),)
	$(call warn-assign,CPPFLAGS)
	KBUILD_CPPFLAGS += $(KCPPFLAGS)
endif
ifneq ($(KAFLAGS),)
	$(call warn-assign,AFLAGS)
	KBUILD_AFLAGS += $(KAFLAGS)
endif
ifneq ($(KCFLAGS),)
	$(call warn-assign,CFLAGS)
	KBUILD_CFLAGS += $(KCFLAGS)
endif

# Use --build-id when available.
LDFLAGS_BUILD_ID = $(patsubst -Wl$(comma)%,%,/
	 $(call ld-option, -Wl$(comma)--build-id,))
LDFLAGS_MODULE += $(LDFLAGS_BUILD_ID)
LDFLAGS_vmlinux += $(LDFLAGS_BUILD_ID)

# Default kernel image to build when no specific target is given.
# KBUILD_IMAGE may be overruled on the command line or
# set in the environment
# Also any assignments in arch/$(ARCH)/Makefile take precedence over
# this default value
export KBUILD_IMAGE ?= vmlinux

#
# 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
#默认的安装目录
#
# INSTALL_MOD_PATH specifies a prefix to MODLIB for module directory
# relocations required by build roots.  This is not defined in the
# makefile but the argument can be passed to make if needed.
#

MODLIB = $(INSTALL_MOD_PATH)/lib/modules/$(KERNELRELEASE)
export MODLIB

#
#  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 will used as the options to the strip command.

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
export mod_strip_cmd

#这里是编译的主体!!!!!!!!!!!!!!!!!!!!!!
```

```
ifeq ($(KBUILD_EXTMOD),)#如果外部模块为空,即编译内核
core-y  += kernel/ mm/ fs/ ipc/ security/ crypto/ block/
#此时我们的core-y目标需要添加各个模块
#vmlinux-dirs即我们编译内核需要添加的各个模块
#所在的内核目录
vmlinux-dirs := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) /
      $(core-y) $(core-m) $(drivers-y) $(drivers-m) /
      $(net-y) $(net-m) $(libs-y) $(libs-m)))
#这里的vmlinux即取目录名称
vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, /#sort is a sequence!
      $(init-n) $(init-) /
      $(core-n) $(core-) $(drivers-n) $(drivers-) /
      $(net-n)  $(net-)  $(libs-n)    $(libs-))))


#################重点################################
#################################################
#################################################
init-y  := $(patsubst %/, %/built-in.o, $(init-y))#这里所有的init/都被
            #init/built-in.o所替代
core-y  := $(patsubst %/, %/built-in.o, $(core-y))#同上
 #vmlinux-main即由各个目录下的build-in.o链接而成
drivers-y := $(patsubst %/, %/built-in.o, $(drivers-y))
net-y  := $(patsubst %/, %/built-in.o, $(net-y))

libs-y1  := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2  := $(patsubst %/, %/built-in.o, $(libs-y))
libs-y  := $(libs-y1) $(libs-y2)

# Build vmlinux
# ---------------------------------------------------------------------------
# vmlinux is built from the objects selected by $(vmlinux-init) and
# $(vmlinux-main). Most are built-in.o files from top-level directories
# in the kernel tree, others are specified in arch/$(ARCH)/Makefile.
# Ordering when linking is important, and $(vmlinux-init) must be first.
#
# vmlinux
#   ^
#   |
#   +-< $(vmlinux-init)
#   |   +--< init/version.o + more
#   |
#   +--< $(vmlinux-main)
#   |    +--< driver/built-in.o mm/built-in.o + more
#   |
#   +-< kallsyms.o (see description in CONFIG_KALLSYMS section)
#
# vmlinux version (uname -v) cannot be updated during normal
# descending-into-subdirs phase since we do not yet know if we need to
# update vmlinux.
# Therefore this step is delayed until just before final link of vmlinux -
# except in the kallsyms case where it is done just before adding the
# symbols to the kernel.
#
# System.map is generated to document addresses of all kernel symbols
```

```
vmlinux-init := $(head-y) $(init-y)#head-y定义在/arch/xx/Makefile
  #head-y := arch/x86/kernel/head_$(BITS).o
  #head-y += arch/x86/kernel/head$(BITS).o
  #head-y += arch/x86/kernel/init_task.o
vmlinux-main := $(core-y) $(libs-y) $(drivers-y) $(net-y)#the main body of linux kernel;
vmlinux-all  := $(vmlinux-init) $(vmlinux-main)
vmlinux-lds  := arch/$(SRCARCH)/kernel/vmlinux.lds
#使用与架构相关的
#链接文件lds
export KBUILD_VMLINUX_OBJS := $(vmlinux-all)

# Rule to link vmlinux - also used during CONFIG_KALLSYMS
# May be overridden by arch/$(ARCH)/Makefile
quiet_cmd_vmlinux__ ?= LD      $@  #?=means the defualts if without other operation
      cmd_vmlinux__ ?= $(LD) $(LDFLAGS) $(LDFLAGS_vmlinux) -o $@ /
      -T $(vmlinux-lds) $(vmlinux-init)                        /
      --start-group $(vmlinux-main) --end-group                /
      $(filter-out $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) vmlinux.o FORCE ,$^)

# Generate new vmlinux version
quiet_cmd_vmlinux_version = GEN     .version
      cmd_vmlinux_version = set -e;                    /
 if [ ! -r .version ]; then   /
   rm -f .version;    /
   echo 1 >.version;    /
 else      /
   mv .version .old_version;   /
   expr 0$$(cat .old_version) + 1 >.version; /
 fi;      /
 $(MAKE) $(build)=init

# Generate System.map
quiet_cmd_sysmap = SYSMAP
      cmd_sysmap = $(CONFIG_SHELL) $(srctree)/scripts/mksysmap

# Link of vmlinux
# If CONFIG_KALLSYMS is set .version is already updated
# Generate System.map and verify that the content is consistent
# Use + in front of the vmlinux_version rule to silent warning with make -j2
# First command is ':' to allow us to use + in front of the rule

#以下是一个命令包,即一组命令可能在多个规则中使用
#相同的一组命令,通过define来定义这组命令作为一个命令,以后
#在需要的规则中通过变量名直接引用!!!!!!!
define rule_vmlinux__
#这里是执行vmlinux的规则
 :
 $(if $(CONFIG_KALLSYMS),,+$(call cmd,vmlinux_version))

 $(call cmd,vmlinux__)
#cmd = @$(echo-cmd) $(cmd_$(1))
#即被展开为cmd_vmlinux__!!!!!!!!!!!!!!=======
#而此命令为一个连接命令,如下!!!!!!!!========即我们连接命令
#cmd_vmlinux__ ?= $(LD) $(LDFLAGS) $(LDFLAGS_vmlinux) -o $@ /
#      -T $(vmlinux-lds) $(vmlinux-init)                        /
#      --start-group $(vmlinux-main) --end-group                /
```

```
#      $(filter-out $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) vmlinux.o FORCE ,$^)

 $(Q)echo 'cmd_$@ := $(cmd_vmlinux__)' > $(@D)/.$(@F).cmd

 $(Q)$(if $($(quiet)cmd_sysmap),   #生成System.map                          /
   echo '  $($(quiet)cmd_sysmap)  System.map' &&)                 /
 $(cmd_sysmap) $@ System.map;                                  /
 if [ $$? -ne 0 ]; then                                  /
  rm -f $@;                                    /
  /bin/false;                                  /
 fi;
 $(verify_kallsyms)
endef

#至此我们的vmlinux就链接完成了,接下来就是我们的bzImage了!!!

ifdef CONFIG_KALLSYMS
# Generate section listing all symbols and add it into vmlinux $(kallsyms.o)
# It's a three stage process:
# o .tmp_vmlinux1 has all symbols and sections, but __kallsyms is
#   empty
#   Running kallsyms on that gives us .tmp_kallsyms1.o with
#   the right size - vmlinux version (uname -v) is updated during this step
# o .tmp_vmlinux2 now has a __kallsyms section of the right size,
#   but due to the added section, some addresses have shifted.
#   From here, we generate a correct .tmp_kallsyms2.o
# o The correct .tmp_kallsyms2.o is linked into the final vmlinux.
# o Verify that the System.map from vmlinux matches the map from
#   .tmp_vmlinux2, just in case we did not generate kallsyms correctly.
# o If CONFIG_KALLSYMS_EXTRA_PASS is set, do an extra pass using
#   .tmp_vmlinux3 and .tmp_kallsyms3.o.  This is only meant as a
#   temporary bypass to allow the kernel to be built while the
#   maintainers work out what went wrong with kallsyms.

ifdef CONFIG_KALLSYMS_EXTRA_PASS
last_kallsyms := 3
else
last_kallsyms := 2
endif

kallsyms.o := .tmp_kallsyms$(last_kallsyms).o


#以下是一个命令包,即一组命令可能在多个规则中使用
#相同的一组命令,通过define来定义这组命令作为一个命令,以后
#在需要的规则中通过变量名直接引用!!!!!!!
define verify_kallsyms  #这里是命令包名称
 $(Q)$(if $($(quiet)cmd_sysmap),                                  /
   echo '  $($(quiet)cmd_sysmap)  .tmp_System.map' &&)              /
   $(cmd_sysmap) .tmp_vmlinux$(last_kallsyms) .tmp_System.map
 $(Q)cmp -s System.map .tmp_System.map ||                         /
  (echo Inconsistent kallsyms data;                       /
   echo Try setting CONFIG_KALLSYMS_EXTRA_PASS;              /
   rm .tmp_kallsyms* ; /bin/false )
endef

# Update vmlinux version before link
```

```
# Use + in front of this rule to silent warning about make -j1
# First command is ':' to allow us to use + in front of this rule
cmd_ksym_ld = $(cmd_vmlinux__)
define rule_ksym_ld
 :
 +$(call cmd,vmlinux_version)
 $(call cmd,vmlinux__)
 $(Q)echo 'cmd_$@ := $(cmd_vmlinux__)' > $(@D)/.$(@F).cmd
endef

# Generate .S file with all kernel symbols
quiet_cmd_kallsyms = KSYM    $@
     cmd_kallsyms = $(NM) -n $< | $(KALLSYMS) /
                $(if $(CONFIG_KALLSYMS_ALL),--all-symbols) > $@

.tmp_kallsyms1.o .tmp_kallsyms2.o .tmp_kallsyms3.o: %.o: %.S scripts FORCE
 $(call if_changed_dep,as_o_S)

.tmp_kallsyms%.S: .tmp_vmlinux% $(KALLSYMS)
 $(call cmd,kallsyms)

# .tmp_vmlinux1 must be complete except kallsyms, so update vmlinux version
.tmp_vmlinux1: $(vmlinux-lds) $(vmlinux-all) FORCE
 $(call if_changed_rule,ksym_ld)

.tmp_vmlinux2: $(vmlinux-lds) $(vmlinux-all) .tmp_kallsyms1.o FORCE
 $(call if_changed,vmlinux__)

.tmp_vmlinux3: $(vmlinux-lds) $(vmlinux-all) .tmp_kallsyms2.o FORCE
 $(call if_changed,vmlinux__)

# Needs to visit scripts/ before $(KALLSYMS) can be used.
$(KALLSYMS): scripts ;

# Generate some data for debugging strange kallsyms problems
debug_kallsyms: .tmp_map$(last_kallsyms)

.tmp_map%: .tmp_vmlinux% FORCE
 ($(OBJDUMP) -h $< | $(AWK) '/^ +[0-9]/{print $$4 " 0 " $$2}'; $(NM) $<) | sort > $@

.tmp_map3: .tmp_map2

.tmp_map2: .tmp_map1

endif # ifdef CONFIG_KALLSYMS

# Do modpost on a prelinked vmlinux. The finally linked vmlinux has
# relevant sections renamed as per the linker script.
quiet_cmd_vmlinux-modpost = LD      $@
     cmd_vmlinux-modpost = $(LD) $(LDFLAGS) -r -o $@                    /
  $(vmlinux-init) --start-group $(vmlinux-main) --end-group          /
  $(filter-out $(vmlinux-init) $(vmlinux-main) FORCE ,$^)
define rule_vmlinux-modpost: #定义命令包
 +$(call cmd,vmlinux-modpost)
 $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost $@
 $(Q)echo 'cmd_$@ := $(cmd_vmlinux-modpost)' > $(dot-target).cmd
endef

#===================================================
```

```
#==================================================
#vmlinux的依赖目标集,开始真正的做主目录下的映像文件
# vmlinux image - including updated kernel symbols
vmlinux: $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) vmlinux.o $(kallsyms.o) FORCE
ifdef CONFIG_HEADERS_CHECK  #这里主要是头检验
 $(Q)$(MAKE) -f $(srctree)/Makefile headers_check
#PHONY += headers_check 当make -f $(srctree)/Makefile headers_check执行以下命令
#headers_check: headers_install
# $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.headersinst ARCH=$(SRCARCH) obj=include
HDRCHECK=1
endif
ifdef CONFIG_SAMPLES
#配置样本即调用Makefile.build 到Samples目录下make生成样本
 $(Q)$(MAKE) $(build)=samples
endif
 $(call vmlinux-modpost)#vmlinux-modpost没有被定义,是一条多余的没有被定义的变量
 $(call if_changed_rule,vmlinux__)
#在我们所以依赖目标都生成后我们开始链接!!!!!!!!!!==========
#if_changed_rule = $(if $(strip $(any-prereq) $(arg-check) ),                /
# @set -e;                                                    /
# $(rule_$(1)))即执行rule_vmlinux__ 924line


 $(Q)rm -f .old_version

# build vmlinux.o first to catch section mismatch errors early
ifdef CONFIG_KALLSYMS
.tmp_vmlinux1: vmlinux.o
endif
#modpost-init表示除了init/下的built-in.o
modpost-init := $(filter-out init/built-in.o, $(vmlinux-init))
vmlinux.o: $(modpost-init) $(vmlinux-main) FORCE
 $(call if_changed_rule,vmlinux-modpost)#这里是没有存在的,无效

# The actual objects are generated when descending,
# make sure no implicit rule kicks in
$(sort $(vmlinux-init) $(vmlinux-main)) $(vmlinux-lds): $(vmlinux-dirs) ;

# Handle descending into subdirectories listed in $(vmlinux-dirs)
# Preset locale variables to speed up the build process. Limit locale
# tweaks to this spot to avoid wrong language settings when running
# make menuconfig etc.
# Error messages still appears in the original language

################真正执行动作!!!!!!####################
PHONY += $(vmlinux-dirs)  #伪目标
$(vmlinux-dirs): prepare scripts
 $(Q)$(MAKE) $(build)=$@
#(1)build=-f scripts/Makefile.build obj=

进行实际的编
#等价于make -f scripts/Makefile.build obj=$@
#在Makefile.build中第一个真正的依赖规则是__build 大概在93行进入这里我们
#__build又依赖于各个目录下的build-in.o或者lib.a等
#$@表示vmlinux-dirs变量中的各个目标用于传给obj
```

```
#vmlinux-dirs := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) /
#       $(core-y) $(core-m) $(drivers-y) $(drivers-m) /
#       $(net-y) $(net-m) $(libs-y) $(libs-m)))

#(2)这里的依赖目标有两个prepare:和scripts
# L1180  prepare: prepare0
# L646   PHONY += scripts
# scripts: scripts_basic include/config/auto.conf

# Build the kernel release string
#
# The KERNELRELEASE value built here is stored in the file
# include/config/kernel.release, and is used when executing several
# make targets, such as "make install" or "make modules_install."
#
# The eventual kernel release string consists of the following fields,
# shown in a hierarchical format to show how smaller parts are concatenated
# to form the larger and final value, with values coming from places like
# the Makefile, kernel config options, make command line options and/or
# SCM tag information.
#
# $(KERNELVERSION)
#   $(VERSION)   eg, 2
#   $(PATCHLEVEL)   eg, 6
#   $(SUBLEVEL)   eg, 18
#   $(EXTRAVERSION)  eg, -rc6
# $(localver-full)
#   $(localver)
#     localversion*  (files without backups, containing '~')
#     $(CONFIG_LOCALVERSION) (from kernel config setting)
#   $(localver-auto)  (only if CONFIG_LOCALVERSION_AUTO is set)
#     ./scripts/setlocalversion (SCM tag, if one exists)
#     $(LOCALVERSION)  (from make command line if provided)
#
#  Note how the final $(localver-auto) string is included *only* if the
# kernel config option CONFIG_LOCALVERSION_AUTO is selected.  Also, at the
# moment, only git is supported but other SCMs can edit the script
# scripts/setlocalversion and add the appropriate checks as needed.

pattern = ".*/localversion[^~]*"
string  = $(shell cat /dev/null /
   `find $(objtree) $(srctree) -maxdepth 1 -regex $(pattern) | sort -u`)

localver = $(subst $(space),, $(string) /
       $(patsubst "%",%,$(CONFIG_LOCALVERSION)))

# If CONFIG_LOCALVERSION_AUTO is set scripts/setlocalversion is called
# and if the SCM is know a tag from the SCM is appended.
# The appended tag is determined by the SCM used.
#
# Currently, only git is supported.
# Other SCMs can edit scripts/setlocalversion and add the appropriate
# checks as needed.
ifdef CONFIG_LOCALVERSION_AUTO
 _localver-auto = $(shell $(CONFIG_SHELL) /
                $(srctree)/scripts/setlocalversion $(srctree))
 localver-auto  = $(LOCALVERSION)$(_localver-auto)
endif
```

```
localver-full = $(localver)$(localver-auto)

# Store (new) KERNELRELASE string in include/config/kernel.release
kernelrelease = $(KERNELVERSION)$(localver-full)
include/config/kernel.release: include/config/auto.conf FORCE
 $(Q)rm -f $@
 $(Q)echo $(kernelrelease) > $@


# Things we need to do before we recursively start building the kernel
# or the modules are listed in "prepare".
# A multi level approach is used. prepareN is processed before prepareN-1.
# archprepare is used in arch Makefiles and when processed asm symlink,
# version.h and scripts_basic is processed / created.

# Listed in dependency order
PHONY += prepare archprepare prepare0 prepare1 prepare2 prepare3
#定义多个伪目标
# prepare3 is used to check if we are building in a separate output directory,
#第三步主要检查我们是否为自定义了输出目录
# and if so do:
# 1) Check that make has not been executed in the kernel src $(srctree)
# 2) Create the include2 directory, used for the second asm symlink
prepare3: include/config/kernel.release
ifneq ($(KBUILD_SRC),)
 @echo '  Using $(srctree) as source for kernel'
 $(Q)if [ -f $(srctree)/.config -o -d $(srctree)/include/config ]; then /
  echo "  $(srctree) is not clean, please run 'make mrproper"';/
  echo "  in the '$(srctree)' directory.";/
  /bin/false; /
 fi;
 $(Q)if [ ! -d include2 ]; then mkdir -p include2; fi;
 $(Q)ln -fsn $(srctree)/include/asm-$(SRCARCH) include2/asm
endif

# prepare2 creates a makefile if using a separate output directory
prepare2: prepare3 outputmakefile

prepare1: prepare2 include/linux/version.h include/linux/utsrelease.h /
                include/asm include/config/auto.conf
 $(cmd_crmodverdir)

archprepare: prepare1 scripts_basic

prepare0: archprepare FORCE
 $(Q)$(MAKE) $(build)=.
 $(Q)$(MAKE) $(build)=. missing-syscalls

# All the preparing..
prepare: prepare0


# Leave this as default for preprocessing vmlinux.lds.S, which is now
# done in arch/$(ARCH)/kernel/Makefile

export CPPFLAGS_vmlinux.lds += -P -C -U$(ARCH)

# The asm symlink changes when $(ARCH) changes.
```

```
# Detect this and ask user to run make mrproper
#set -e 是设置一种shell状态即如果它执行的任何一条命令的退出码非零
#都立刻终止

include/asm: FORCE
 $(Q)set -e; asmlink=`readlink include/asm | cut -d '-' -f 2`;   /
 if [ -L include/asm ]; then                                    /
  if [ "$$asmlink" != "$(SRCARCH)" ]; then                 /
   echo "ERROR: the symlink $@ points to asm-$$asmlink but asm-$(SRCARCH) was
expected"; /
   echo "       set ARCH or save .config and run 'make mrproper' to fix it";          /
   exit 1;                                      /
  fi;                                           /
 else                                                   /
  echo '  SYMLINK $@ -> include/asm-$(SRCARCH)';         /
  if [ ! -d include ]; then                              /
   mkdir -p include;                              /
  fi;                                           /
  ln -fsn asm-$(SRCARCH) $@;                             /
 fi

# Generate some files
# ---------------------------------------------------------------------------

# KERNELRELEASE can change from a few different places, meaning version.h
# needs to be updated, so this check is forced on all builds

uts_len := 64
define filechk_utsrelease.h
 if [ `echo -n "$(KERNELRELEASE)" | wc -c ` -gt $(uts_len) ]; then /
   echo "'$(KERNELRELEASE)" exceeds $(uts_len) characters' >&2;    /
   exit 1;                                              /
 fi;                                                   /
 (echo /#define UTS_RELEASE /"$(KERNELRELEASE)/";)
endef

define filechk_version.h
 (echo /#define LINUX_VERSION_CODE $(shell                          /
 expr $(VERSION) Kbuild)))

PHONY += headers_install_all
headers_install_all: include/linux/version.h scripts_basic FORCE
 $(Q)$(MAKE) $(build)=scripts scripts/unifdef
 $(Q)for arch in $(HDRARCHES); do /
  $(MAKE) ARCH=$$arch -f $(srctree)/scripts/Makefile.headersinst obj=include BIASMDIR=-
bi-$$arch ;/
  done

PHONY += headers_install
headers_install: include/linux/version.h scripts_basic FORCE
 @if [ ! -r $(srctree)/include/asm-$(SRCARCH)/Kbuild ]; then /
   echo '*** Error: Headers not exportable for this architecture ($(SRCARCH))'; /
   exit 1 ; fi
 $(Q)$(MAKE) $(build)=scripts scripts/unifdef
 $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.headersinst ARCH=$(SRCARCH) obj=include

PHONY += headers_check_all
headers_check_all: headers_install_all
```

```
 $(Q)for arch in $(HDRARCHES); do /
  $(MAKE) ARCH=$$arch -f $(srctree)/scripts/Makefile.headersinst obj=include BIASMDIR=-
bi-$$arch HDRCHECK=1 ;/
  done

PHONY += headers_check
headers_check: headers_install
 $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.headersinst ARCH=$(SRCARCH) obj=include
HDRCHECK=1

# ---------------------------------------------------------------------------
# Modules

ifdef CONFIG_MODULES

# By default, build modules as well

all: modules

# Build modules
#
# A module can be listed more than once in obj-m resulting in
# duplicate lines in modules.order files.  Those are removed
# using awk while concatenating to the final file.

PHONY += modules
modules: $(vmlinux-dirs) $(if $(KBUILD_BUILTIN),vmlinux)
 $(Q)$(AWK) '!x[$$0]++' $(vmlinux-dirs:%=$(objtree)/%/modules.order) >
$(objtree)/modules.order
 @echo '  Building modules, stage 2.';
 $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost


# Target to prepare building external modules
PHONY += modules_prepare
modules_prepare: prepare scripts

# Target to install modules
PHONY += modules_install
modules_install: _modinst_ _modinst_post

PHONY += _modinst_
_modinst_:
 @if [ -z "`$(DEPMOD) -V 2>/dev/null | grep module-init-tools`" ]; then /
  echo "Warning: you may need to install module-init-tools"; /
  echo "See http://www.codemonkey.org.uk/docs/post-halloween-2.6.txt";/
  sleep 1; /
 fi
 @rm -rf $(MODLIB)/kernel
 @rm -f $(MODLIB)/source
 @mkdir -p $(MODLIB)/kernel
 @ln -s $(srctree) $(MODLIB)/source
 @if [ ! $(objtree) -ef  $(MODLIB)/build ]; then /
  rm -f $(MODLIB)/build ; /
  ln -s $(objtree) $(MODLIB)/build ; /
 fi
 @cp -f $(objtree)/modules.order $(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.
PHONY += _modinst_post
_modinst_post: _modinst_
 $(call cmd,depmod)

else # CONFIG_MODULES

# Modules not configured
# ---------------------------------------------------------------------

modules modules_install: FORCE
 @echo
 @echo "The present kernel configuration has modules disabled."
 @echo "Type 'make config' and enable loadable module support."
 @echo "Then build a kernel with module support enabled."
 @echo
 @exit 1

endif # CONFIG_MODULES

###
# Cleaning is done on three levels.
# make clean     Delete most generated files
#                Leave enough to build external modules
# make mrproper  Delete the current configuration, and all generated files
# make distclean Remove editor backup files, patch leftover files and the like

# Directories & files removed with 'make clean'
```