A Dive Into Kbuild

Aug, 2018
Cao Jin <caoj.fnst@cn.fujitsu.com>
Fujitsu Limited.
Outline

- Simple instroduction of Kbuild
- Instroduction of Kconfig
- Dive into Kbuild
- Current status & update
Simple introduction of Kbuild

- A build framework based on GNU make and a standard set of cross platform tools, designed for linux kernel.
  - include a configuration framework called Kconfig

- Powerful build system
  - Highly modular and customizable, friendly to linux hacker
  - The same code base is used for a different range of computing systems, from supercomputers to very tiny embedded devices.

- Not just linux kernel who use kbuild/kconfig
  - U-boot
  - seabios
  - Xen
  - ...
Simple introduction of Kbuild

The benefits of understanding Kbuild

- Acquire the *perspective of God*.
- Deep understanding how does makefile manage big project
- Won't be scared when encountering compilation error
- See the relation and difference between vmlinux & bzImage
- Help to understand the boot process of kernel
- ...

Copyright 2018 FUJITSU LIMITED
# Introduction of Kconfig

**MANY targets for Kconfig**

<table>
<thead>
<tr>
<th>Configuration targets:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Update current config utilising a line-oriented program</td>
</tr>
<tr>
<td>nconfig</td>
<td>Update current config utilising a ncurses menu based program</td>
</tr>
<tr>
<td>menuconfig</td>
<td>Update current config utilising a menu based program</td>
</tr>
<tr>
<td>xconfig</td>
<td>Update current config utilising a Qt based front-end</td>
</tr>
<tr>
<td>gconfig</td>
<td>Update current config utilising a GTK+ based front-end</td>
</tr>
<tr>
<td>oldconfig</td>
<td>Update current config utilising a provided .config as base</td>
</tr>
<tr>
<td>localmodconfig</td>
<td>Update current config disabling modules not loaded</td>
</tr>
<tr>
<td>localyesconfig</td>
<td>Update current config converting local mods to core</td>
</tr>
<tr>
<td>defconfig</td>
<td>New config with default from ARCH supplied defconfig</td>
</tr>
<tr>
<td>savedefconfig</td>
<td>Save current config as ./defconfig (minimal config)</td>
</tr>
<tr>
<td>allnoconfig</td>
<td>New config where all options are answered with no</td>
</tr>
<tr>
<td>alllyesconfig</td>
<td>New config where all options are accepted with yes</td>
</tr>
<tr>
<td>allmodconfig</td>
<td>New config selecting modules when possible</td>
</tr>
<tr>
<td>allddefconfig</td>
<td>New config with all symbols set to default</td>
</tr>
<tr>
<td>randconfig</td>
<td>New config with random answer to all options</td>
</tr>
<tr>
<td>listnewconfig</td>
<td>List new options</td>
</tr>
<tr>
<td>olddefconfig</td>
<td>Same as oldconfig but sets new symbols to their default value without prompting</td>
</tr>
<tr>
<td>kvmconfig</td>
<td>Enable additional options for kvm guest kernel support</td>
</tr>
<tr>
<td>xenconfig</td>
<td>Enable additional options for xen dom0 and guest kernel support</td>
</tr>
<tr>
<td>tinyconfig</td>
<td>Configure the tiniest possible kernel</td>
</tr>
</tbody>
</table>

Copyright 2018 FUJITSU LIMITED
Introduction of Kconfig

- How .config is produced
Introduction of Kconfig

- **config targets usage**
  - Save current config as a default config?
    - make savedefconfig;
    - `cp defconfig arch/$(ARCH)/my_cool_defconfig;`
    - `# file name must end with "_defconfig"
    - make my_cool_defconfig
  - Customize configuration automatically
    - make localmodconfig
    - have your specific configuration in "*.config" file under `arch/$(ARCH)/configs` or `kernel/configs`
    - make "*.config"

  **TIP:** You must know well about the dependency of your specific configuration
Introduction of Kconfig

- `syncconfig` (Was silentoldconfig)

- `auto.conf` & `tristate.conf`: used in Makefile text processing
  - example: `obj-$(CONFIG_GENERIC_CALIBRATE_DELAY) += calibrate.o`

- `include/config/* .h`: used to track configuration update
  - details in scripts/basic/fixdep.c & `.<target>.cmd`
Dive into Kbuild

The most important thing before diving?
- GNU Makefile of course
- The best way to learn? `info make`

The basics of GNU Makefile
- Phony target
- Force target
- Empty Recipes
- Two "flavors" of variables
- Multiple Rules for One Target
- Generating Prerequisites Automatically
- Functions
- Target-specific Variable Values
- ...

TARGET ... : PREREQUISITES ...
RECIPE
...
...
Dive into Kbuild

- Kbuild Makefiles have 5 parts
  - Makefile the top Makefile.
  - .config the kernel configuration file.
  - arch/$(ARCH)/Makefile the arch Makefile.
  - scripts/Makefile.* common rules etc. for all kbuild Makefiles.
  - kbuild Makefiles there are about 500 of these.

- All kinds of targets need to build
  - vmlinux, bzImage
  - modules
  - host program
  - library
  - ...

Copyright 2018 FUJITSU LIMITED
How Linux kernel is compiled?

- Recursive make

```
head_64.o
head64.o
ebda.o
platform-quirks.o init/built-in.a
  |
  $(head-y) $(init-y)
  |
  |
  |
  KBUILD_VMLINUX_INIT

/usr/built-in.a  lib/built-in.a
/kernel/built-in.a
certs/built-in.a  drivers/built-in.a
/mm/built-in.a    sound/built-in.a
/fs/built-in.a    firmware/built-in.a
/ipc/built-in.a   security/built-in.a
/security/built-in.a
/crypto/built-in.a
/block/built-in.a virt/built-in.a
lib/built-in.a

$(core-y) $(libs-y2)
$(drivers-y) $(net-y)
$(virt-y)
$(libs-y1)
```
How kbuild implement recursive make

Show you the code

```make
$(sort $(vmlinux-deps)): $(vmlinux-dirs) ;

vmlinux-deps := $(KBUILD_LDS) $(KBUILD_VMLINUX_INIT) $(KBUILD_VMLINUX_MAIN) $(KBUILD_VMLINUX_LIBS)

export KBUILD_VMLINUX_INIT := $(head-y) $(init-y)
export KBUILD_VMLINUX_MAIN := $(core-y) $(libs-y2) $(drivers-y) $(net-y) $(virt-y)
export KBUILD_VMLINUX_LIBS := $(libs-y1)
export KBUILD_LDS := arch/$(SRCARCH)/kernel/vmlinux.lds

# In arch/x86/Makefile
head-y := arch/x86/kernel/head_$(BITS).o
head-y += arch/x86/kernel/head$(BITS).o
head-y += arch/x86/kernel/ebda.o
head-y += arch/x86/kernel/platform-quirks.o
```

Copyright 2018 FUJITSU LIMITED
init-y := init/
drivers-y := drivers/ sound/ firmware/
net-y := net/
libs-y := lib/
core-y := usr/
virt-y := virt/

init-y := $(patsubst %/, %/built-in.a, $(init-y))
core-y := $(patsubst %/, %/built-in.a, $(core-y))
drivers-y := $(patsubst %/, %/built-in.a, $(drivers-y))
net-y := $(patsubst %/, %/built-in.a, $(net-y))
libs-y1 := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2 := $(patsubst %/, %/built-in.a, $(filter-out %.a, $(libs-y)))
virt-y := $(patsubst %/, %/built-in.a, $(virt-y))

$(vmlinux-dirs): prepare scripts
(Q)$(MAKE) $(build)@$ need-builtin=1

$(sort $(vmlinux-deps)): $(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) $(virt-y)))

make -f $(srctree)/scripts/Makefile.build obj=<subdir_name> need-builtin=1
Example: init/

```
# init/Makefile
obj-y := main.o version.o mounts.o
ifneq ($(CONFIG_BLK_DEV_INITRD),y)
  obj-y += noinitramfs.o
else
  obj-$(CONFIG_BLK_DEV_INITRD) += initramfs.o
endif
obj-$(CONFIG_GENERIC_CALIBRATE_DELAY) += calibrate.o
obj-y += init_task.o

mounts-y := do_mounts.o
mounts-$(CONFIG_BLK_DEV_RAM) += do_mounts_rd.o
mounts-$(CONFIG_BLK_DEV_INITRD) += do_mounts_initrd.o
mounts-$(CONFIG_BLK_DEV_MD) += do_mounts_md.o

# scripts/Makefile.build
PHONY := __build
__build:

-include include/config/auto.conf
include scripts/Kbuild.include

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

ifeq ($(hostprogs-y)$((hostprogs-m)$((hostlibs-y)$((hostlibs-m)$((hostcxxlibs-y)$((hostcxxlibs-m),))))
include scripts/Makefile.host
```

Copyright 2018 FUJITSU LIMITED
Example: init/

scripts/Makefile.build

__build: $(if $(KBUILD_BUILTIN),$((builtin-target) $(lib-target) $(extra-y)) \$(if $(KBUILD_MODULES),$(obj-m) $(modorder-target)) \$(subdir-ym) $(always)
@:

ifeq ($(strip $(real-obj-y) $(need-built-in))),
builtin-target := $(obj)/built-in.a
endif

$(builtin-target): $(real-obj-y) FORCE
 $(call if_changed,ar_builtin)

cmd_ar_builtin = rm -f $@; 
 $(AR) rcSTP$(KBUILD_ARFLAGS) $@ $(filter $(real-obj-y), $^)

$(subdir-ym):
 $(Q)$(MAKE) $(build)=$@ need-builtin=$(if $(findstring $@,$(subdir-obj-y)),1)

$(obj)/%.o: $(src)/%.c $(recordmcount_source) $(objtool_dep) FORCE
 $(call cmd,force_checksrc)
 $(call if_changed_rule,cc_o_c)
 cmd_cc_o_c = $(CC) $(c_flags) -c -o $@

Example: init/

- scripts/Makefile.lib

multi-used-y := $(sort $(foreach m,$(obj-y), $(if $(strip $(m:.o=-objs)) $(m:.o=-y))), $(m)))

real-obj-y := $(foreach m, $(obj-y), $(if $(strip $(m:.o=-objs)) $(m:.o=-y))),$(m))

A simple introduction to compilation flags

- Global: KBUILD_CFLAGS
- Apply for current directory: cc-flags
- Apply for current & sub-directory: subdir-ccflags-y
- Apply for certain files: CFLAGS_$@ & CFLAGS_REMOVE_$@
# Final link of vmlinux with optional arch pass after final link

```
cmd_link-vmlinux = $(CONFIG_SHELL) $< $(LD) $(LDFLAGS) $(LDFLAGS_vmlinux) ; $(MAKE) -f $(ARCH_POSTLINK) $@, true
```

vmlinux: scripts/link-vmlinux.sh autoksyms_recursive $(vmlinux-deps) FORCE +$(call if_changed,link-vmlinux)
bzImage memory

- `startup_32`
- `setup.bin`
- `.head.text`
- `.rodata..compressd (vmlinux.bin.gz)`
- `.text`
- `.rodata`
- `.got`
- `.data`
- `.bss`
- `.pgtable`

VMImage Memory Layout:
Modules

- scripts/Makefile.modpost

- stage 1 creates:
  - The individual .o files used for the module
  - A <module>.o file which is the .o files above linked together
  - A <module>.mod file in $(MODVERDIR)/, listing the name of the preliminary <module>.o file, plus all .o files

- stage 2 does:
  - Find all modules from the files listed in $(MODVERDIR)/
  - modpost is used to
    - create one <module>.mod.c file per module
    - create one Module.symvers file with CRC for all exported symbols
  - compile all <module>.mod.c files
  - final link of the module to a <module.ko> file
Trick

- Dependency tracking
  - All prerequisite files (both *.c and *.h)
  - CONFIG_ options used in all prerequisite files
  - Command-line used to compile target

- How Kbuild does it

  # In scripts/Makefile.build. Simplify for illustration
  $(obj)/%.o: $(src)/%.c
    $(call if_changed_rule,cc_o_c)

  # in scripts/Kbuild.include
  if_changed_rule = $(if $(strip $(any-prereq) $(arg-check) ),
    @set -e;
    $(rule_$(1)), @:)

  # check Kbuild.include for definition of any-prereq & arg-check

If 'main.c' uses 'defs.h' via an '#include',
you would write:

    main.o: defs.h
Trick

How Kbuild does it - continued

```bash
# In scripts/Makefile.lib
c_flags = -Wp,-MD,$(depfile) $(NOSTDINC_FLAGS) $(LINUXINCLUDE) \ 
  -include $(srctree)/include/linux/compiler_types.h \ 
  $(__c_flags) $(modkern_cflags) \ 
  $(basename_flags) $(modname_flags)

# In scripts/Kbuild.include
cmd_and_fixdep = \ 
  $(echo-cmd) $(cmd_$(_(1))); \ 
  scripts/basic/fixdep $(depfile) @'$(make-cmd)' > $(dot-target).tmp;\ 
  rm -f $(depfile); \ 
  mv -f $(dot-target).tmp $(dot-target).cmd;

# In scripts/Makefile.build
cmd_files := $(wildcard $(foreach f,$(sort $(targets)),$(dir $(f)).$(notdir $(f)).cmd))
ifeq ($(cmd_files),)
  include $(cmd_files)
endif
```
Current status

Kbuild Maintainer
Masahiro Yamada, the latest maintainer since 2017-3. He made large amount of improvements and fixes to Kbuild. VERY productive!
Recent update

- Kbuild is still under active development
  - Numerous cleanup
  - Fixes for compatibility to clang.
  - Thin archive: builtin.o --> builtin.a
- performance optimization for incremental build:
  - optimize compiler option test: move it from compilation to configuration
  - optimize output directory creation: speeding up the incremental build with O= option.
- …