FUJITSU

shaping tomorrow with you

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 instroduction of Kbuild

FUJITSU

- A build framework based on GNU make and a standard set of cross platform tools, designed for linux kernel.
  - include a configration 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 instroduction of Kbuild

FUJITSU

- 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
  - ...

Introduction of Kconfig

FUJITSU

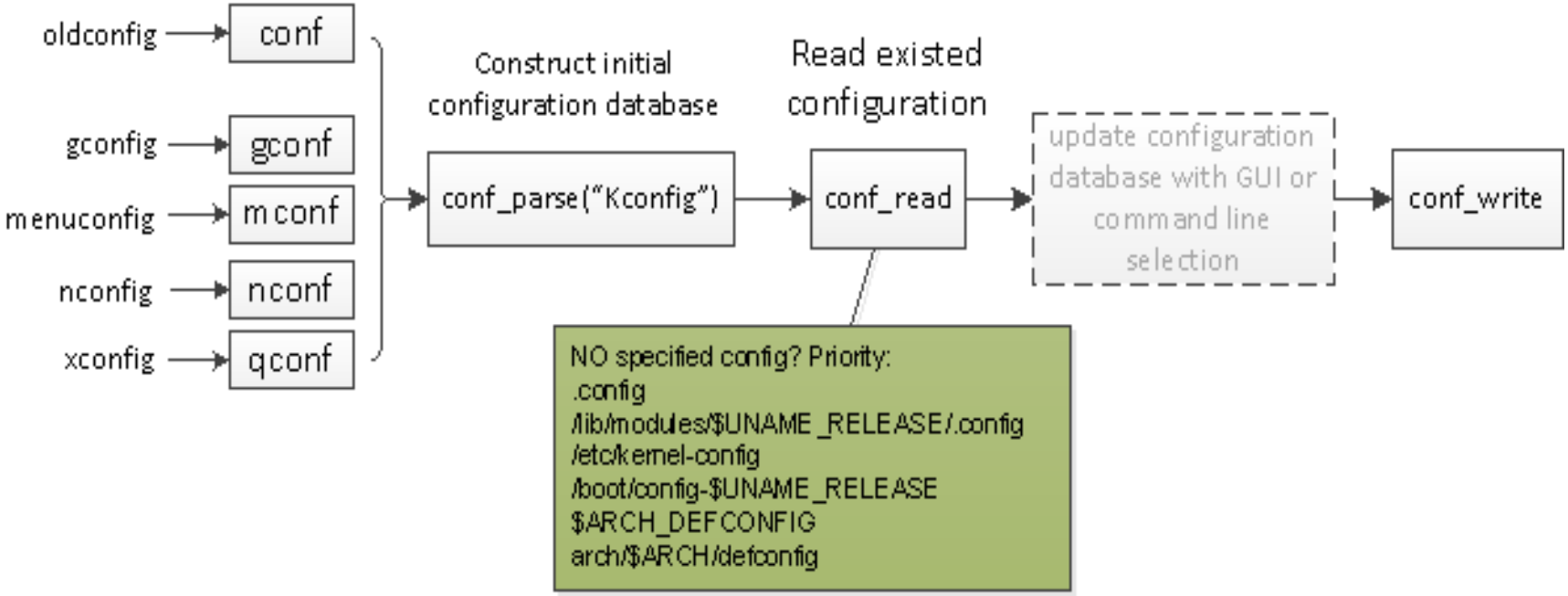
- MANY targets for Kconfig

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

Introduction of Kconfig

FUJITSU

How .config is produced



Introduction of Kconfig

FUJITSU

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
```

```
cp .config  arch/<$ARCH>/config/my_cool_defconfig
```

Customize configuration automatically

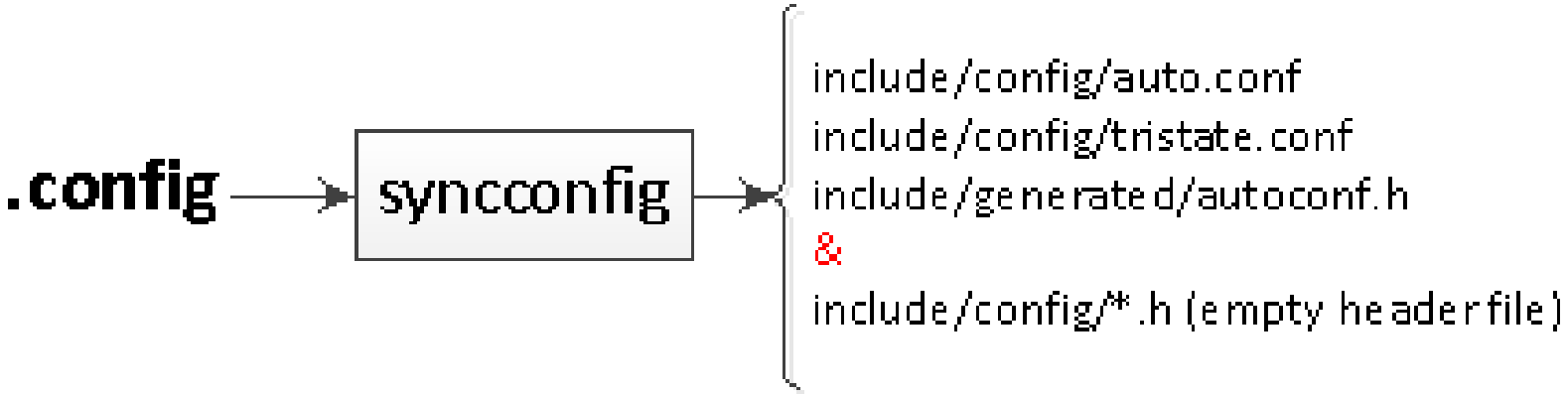
- make localmodconfig
- have your specific configuration in *.config file under arch/$(ARCH)/configs or kernel/configs/
- make *.config

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

Introduction of Kconfig

FUJITSU

- 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

FUJITSU

- 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

FUJITSU

- 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
  - ...

How linux kenrel is compiled?

FUJITSU

Recursive make

```
                                        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
head_64.o                               ipc/built-in.a
head64.o                                security/built-in.a  net/built-in.a
ebda.o                                  crypto/built-in.a
platform-quirks.o    init/built-in.a    block/built-in.a     virt/built-in.a      lib/built-in.a
  (x86)
    |                       |               |                    |                    |
                                             $(core-y) $(libs-y2)
$(head-y)                $(init-y)           $(drivers-y) $(net-y)               $(libs-y1)
                                             $(virt-y)
    |                       |               |                    |                    |
    |                       |               |                    |                    |
     \                     /                 \                  /                    |
      \                   /                   \                /                     |

       KBUILD_VMLINUX_INIT                     KBUILD_VMLINUX_MAIN              KBUILD_VMLINUX_LIBS
               \                                     |                                 /
                \                                    |                                /
                 \__________________________________|________________________________/
                                                     |
                                                  vmlinux
```

How kbuild implement recursive make

FUJITSU

- Show you the code

```
_all<--all--|--vmlinux--|--scripts/link-vmlinux.sh
            |           |--autoksyms_recursive
            |           |
            |           |--$(vmlinux-deps)--|--<lots of work>
            |--bzImage
            |
            |--modules
```

```
$(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
```

Show you the code

```
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))
```

```
$(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)))
```

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

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

Example: init/

make -f scripts/Makefile.build obj=init need-builtin=1

```
# 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

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

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)
             @:

ifneq ($(strip $(real-obj-y) $(need-builtin)),)
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/

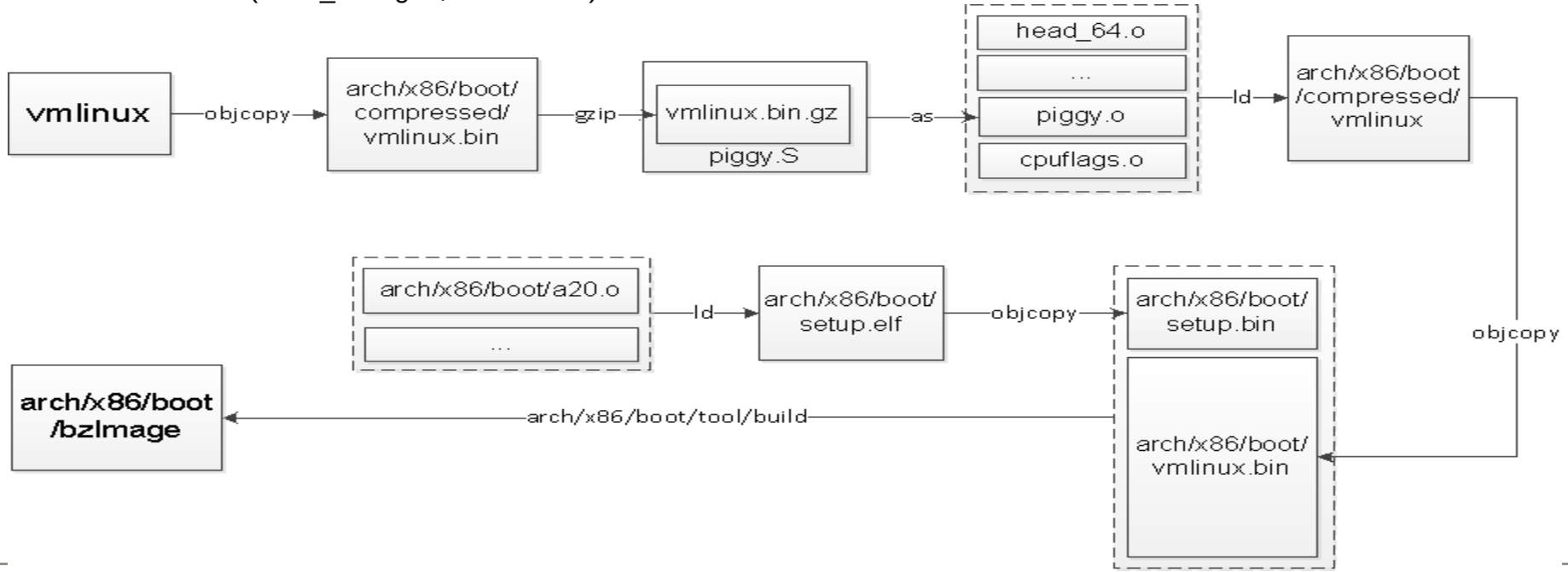
- scripts/Makefile.lib

```
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:.o=-objs)) $($(m:.o=-y)),$(m)))
```

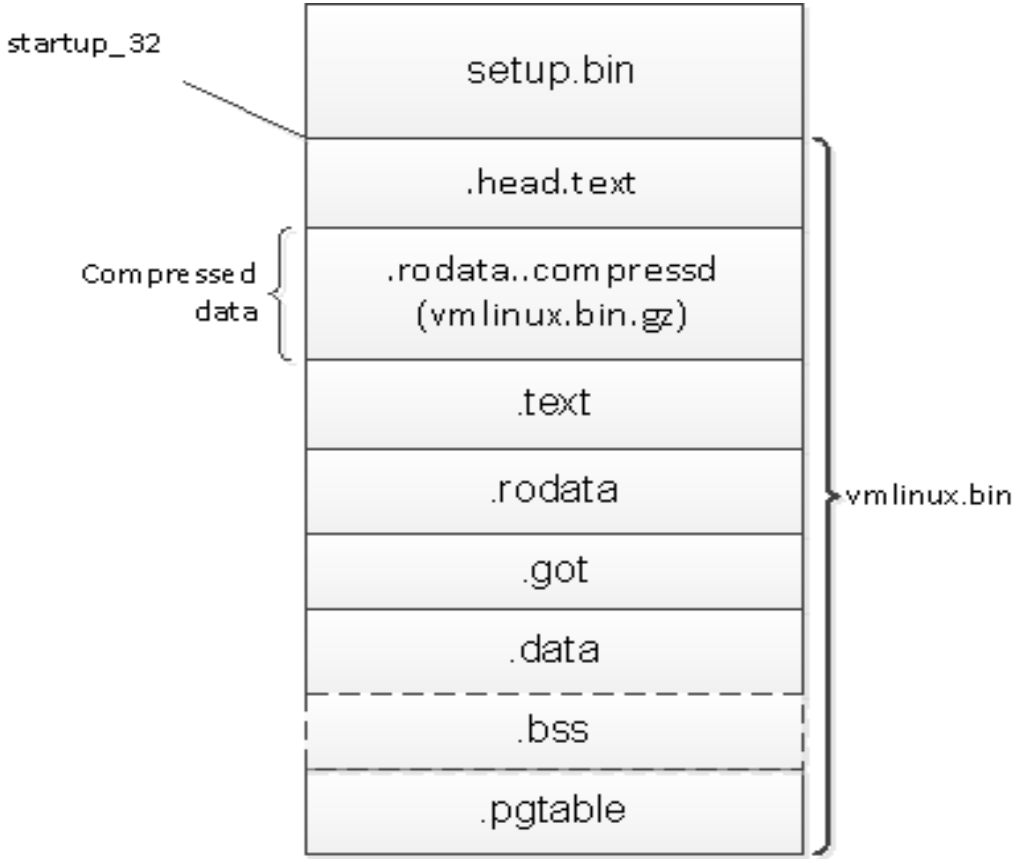
- A simple introduction to compilation flags
  - Global: KBUILD_CFLAGS
  - Apply for current directory: cc-flags
  - Apply for current & sub-directory: subdir-ccflags-y
  - Apply for certain files: CFLAGS_$@ & CFLAGS_REMOVE_@

vmlinux & bzImage

```
# Final link of vmlinux with optional arch pass after final link
cmd_link-vmlinux =                                          \
        $(CONFIG_SHELL) $< $(LD) $(LDFLAGS) $(LDFLAGS_vmlinux) ;    \
        $(if $(ARCH_POSTLINK), $(MAKE) -f $(ARCH_POSTLINK) $@, true)

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



bzImage memory



Modules

FUJITSU

- 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

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

```
main.o: defs.h
```

- 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
```

Trick

FUJITSU

How Kbuild does it - continued

```
# 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 $(f)).cmd))
ifneq ($(cmd_files),)
  include $(cmd_files)
endif
```

Current status

FUJITSU

- Kbuild Maintainer

  Masahiro Yamada,

  the latest maintainer since 2017-3.

  He made large amount of improvements

  and fixes to Kbuild. VERY productive!

Recent update

FUJITSU

- 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.
  - …

Q & A

FUJITSU

shaping tomorrow with you