망고100 보드로 놀아보자 -10

Kernel build 분석

http://cafe.naver.com/embeddedcrazyboys
http://www.mangoboard.com

커널 build 환경 분석

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 29
EXTRAVERSION =
NAME = Temporary Tasmanian Devil
```

>(Top Dir)/Makefile 첫번째 라인에 위치
>커널 2.6.29 버전 사용

```
ARCH        ?= arm
CROSS_COMPILE   := $(shell if [ -f .cross_compile ]; then \
                                                    cat .cross_compile; \
                           fi)
```

>ARCH?=arm 의미는 ARCH 의 값으로 arm있느냐 묻고, 없으면, arm 문자를 대입
>.cross_compile 이 있으면, .cross_compile 내용을 CROSS_COMPILE 로 사용

커널 build 환경 분석(Conf)

- #make mango100_android_defconfig 실행 시

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

Scripts/kconfig/Makefile에서 아래 코드 수행
%_defconfig: $(obj)/conf
    (Q)< -D arch/$(SRCARCH)/configs/$@ $(Kconfig)
➔make –D arch/arm/configs/mango100_android_defconfig

→ .config 파일생성

```
[icanjji@localhost mango100_kernel_2010_06_30]$ ls -al arch/arm/configs/mango100*
-rw-rw-r-- 1 icanjji icanjji 45298 2010-06-30 02:51 arch/arm/configs/mango100_android_defconfig
-rw-rw-r-- 1 icanjji icanjji 46308 2010-06-30 02:51 arch/arm/configs/mango100_defconfig
-rw-rw-r-- 1 icanjji icanjji 50889 2010-06-30 02:51 arch/arm/configs/mango100_wifi_android_defconfig
```

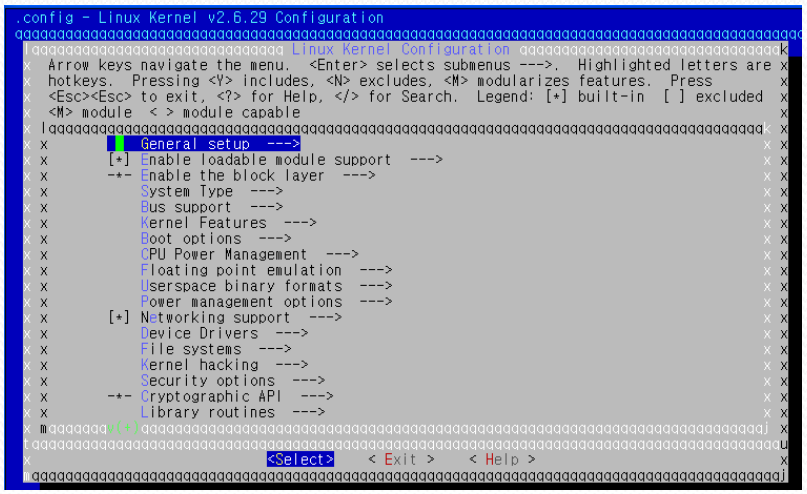
커널 build 환경 분석(Conf)

- #make menu_config 명령 실행 시

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

#make scripts/kconfig menuconfig 이 실행 됨

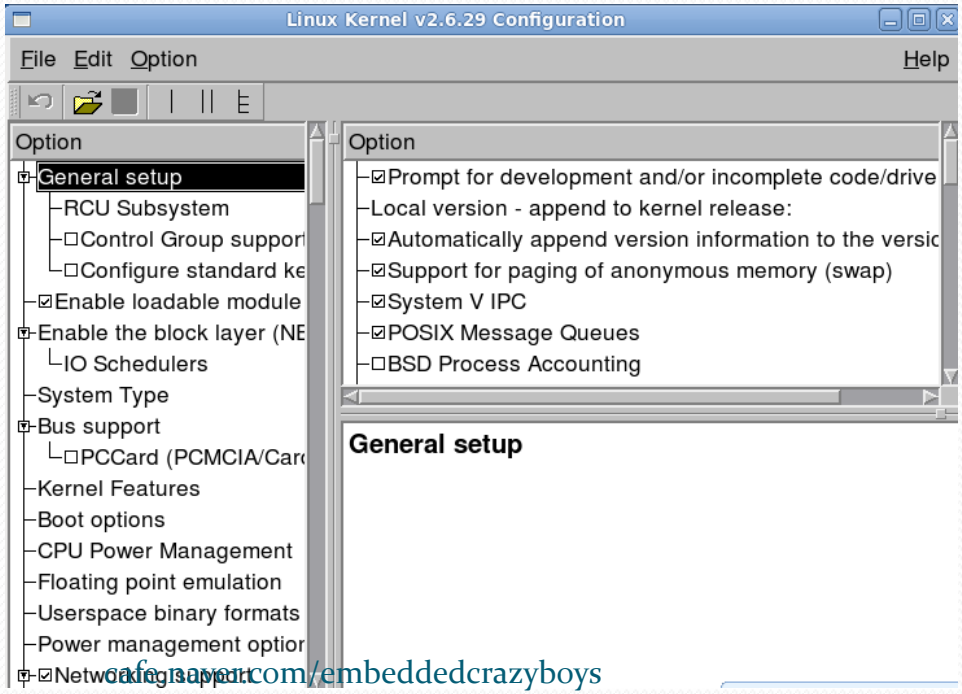
menuconfig: $(obj)/mconf
$< $(Kconfig)



.config 저장

커널 build 환경 분석(Conf)

- #make xconfig (QT3 Package 필요)
- #yum install qt* 명령으로 설치
- “cannot find –lXi 에러 발생 시 #yum install libXi 실행



커널 build 실행 분석

- $(TOP)/Makefile – 최상위 Makefile
  - vmlinux와 modules 생성
- .config – 커널 설정 파일
  - make [config | menuconfig | xconfig] 를 통해 생성.
- arch/$(ARCH)/Makefile - 아키텍처별 makefile
- scripts/Makefile.* - 모든 kbuild Makefile에 사용되는 규칙이 들어있는 파일
- kbuild Makefiles – 약 500개 정도가 있다.

커널 build 실행 분석

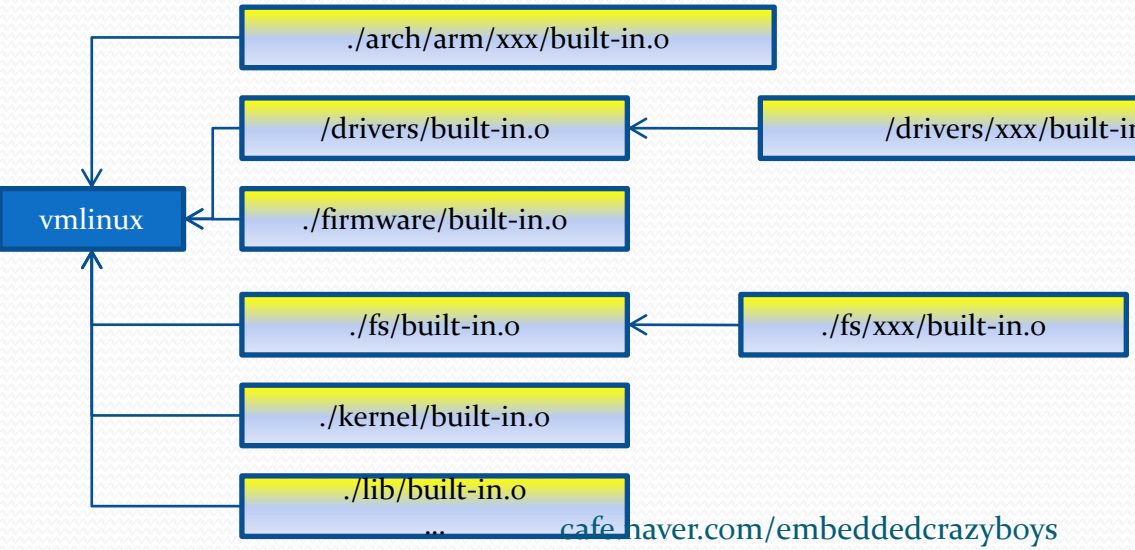
1. 커널 설정 (make config|menuconfig|xconfig)
   .config를 만듬
2. 커널 버전을 include/linux/version.h 에 저장
3. include/asm-$(ARCH)에 대한 심볼릭 링크 만듬
4. arch/$(ARCH)/Makefile 에서 정의된, 그외의 타겟 빌딩을 위한 모든 종속 리스트를 준비
5. init-*, core-*, driver-*, net-* 등의 타겟 등을 만듬
6. 모든 오브젝트들이 링크되고, 소스 트리의 루트 디렉토리에 vmlinux를 만듬.
7. 최종 부트이미지(zImage)를 만들기 위한 아키텍처에 따른 부분이 실행됨

Built-in object goals (obj-y)

- specifying object files for vmlinux
- "$(LD) –r" : to merge $(obj-y) files into one built-in.o file

```
 5 obj-y     = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
 6           exit.o itimer.o time.o softirq.o resource.o \
 7           sysctl.o capability.o ptrace.o timer.o user.o \
 8           signal.o sys.o kmod.o workqueue.o pid.o \
 9           rcupdate.o extable.o params.o posix-timers.o \
10           kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
11           hrtimer.o rwsem.o latency.o nsproxy.o srcu.o
```

ex. $(TOP)/kernel/Makefile



컴파일 후
#find . –name built-in.o
명령으로 확인

Built-in object goals (obj-y)

```
echo "(patsubst pattern,replacement,text)"
vmlinux-dirs    := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
                   $(core-y) $(core-m) $(drivers-y) $(drivers-m) \
                   $(net-y) $(net-m) $(libs-y) $(libs-m)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/, \
                   $(init-n) $(init-) \
                   $(core-n) $(core-) $(drivers-n) $(drivers-) \
                   $(net-n)  $(net-)  $(libs-n)    $(libs-))))

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

Loadable module goals – (obj-m)

- object files which are built as loadable kernel modules.

```
# This goes last, so that "real" scsi devices probe e
obj-$(CONFIG_SCSI_DEBUG)        += scsi_debug.o

obj-$(CONFIG_SCSI_WAIT_SCAN)    += scsi_wait_scan.o

scsi_mod-y                      += scsi.o hosts.o scs
                                   scsicam.o scsi_err
scsi_mod-$(CONFIG_SCSI_DMA)     += scsi_lib_dma.o
scsi_mod-y                      += scsi_scan.o scsi_s
scsi_mod-$(CONFIG_SCSI_NETLINK) += scsi_netlink.o
scsi_mod-$(CONFIG_SYSCTL)       += scsi_sysctl.o
scsi_mod-$(CONFIG_SCSI_PROC_FS) += scsi_proc.o
```

ex. $(TOP)/driver/scsi/Makefile

```
# CONFIG_CHR_DEV_ST is not set
# CONFIG_CHR_DEV_OSST is not set
# CONFIG_BLK_DEV_SR is not set
# CONFIG_CHR_DEV_SG is not set
# CONFIG_CHR_DEV_SCH is not set

#
# Some SCSI devices (e.g. CD jukebox) sup
#
# CONFIG_SCSI_MULTI_LUN is not set
# CONFIG_SCSI_CONSTANTS is not set
# CONFIG_SCSI_LOGGING is not set
# CONFIG_SCSI_SCAN_ASYNC is not set
CONFIG_SCSI_WAIT_SCAN=m

#
# SCSI Transports
#
# CONFIG_SCSI_SPI_ATTRS is not set
```

ex. $(TOP)/.config

```
@$(kecho) '  Building modules, stage 2.';
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.fwinst obj=firmware __fw_modbuild
```

- 참고!! 커널에 포함되지 않은 external module을 컴파일 하기위해서는 Documentation/kbuild/modules.txt를 참조
  - 디바이스 드라이버 개발

Environment Variables

variable	value	Description
V	0	빌드시에, 현재 컴파일되는 파일명만을 보여줌. (default)
V	1	빌드시에 실행되는 모든 명령 및 메시지를 보여줌.
O	dir	컴파일 되는 모든 output file들을 dir에 저장되게 지정
C	1	빌드과정에서 sparse tool이 컴파일된 파일을 체크하게끔 한다. sparse은 커널 소스 파일의 프로그래밍 에러를 찾는 툴이다.
C	2	sparse tool은 컴파일에 관계없이 모든 파일을 체크하게끔 한다.

Example :# make V=1 ARCH=arm

•커널 빌드시, 명령 및 메시지 출력 옵션

```
38 ifdef V
39   ifeq ("$(origin V)", "command line")
40     KBUILD_VERBOSE = $(V)
41   endif
42 endif
43 ifndef KBUILD_VERBOSE
44   KBUILD_VERBOSE = 0
45 endif
```

```
288 ifeq ($(KBUILD_VERBOSE),1)
289   quiet =
290   Q =
291 else
292   quiet=quiet_
293   Q = @
294 endif
```

$(top)/Makefile

•커널 빌드시, 소스 코드 체크 옵션

```
57 ifdef C
58   ifeq ("$(origin C)", "command line")
59     KBUILD_CHECKSRC = $(C)
60   endif
61 endif
62 ifndef KBUILD_CHECKSRC
63   KBUILD_CHECKSRC = 0
64 endif
```

•빌드된 파일의 출력 디렉토리 지정

```
101 ifdef O
102   ifeq ("$(origin O)", "command line")
103     KBUILD_OUTPUT := $(O)
104   endif
105 endif
```

How to build vmlinux ??

-Makefile : 아키텍처 독립적인 부분

```
837 # vmlinux image - including updated kernel symbols
838 vmlinux: $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) vmlinux.o $(kallsyms.o) FORCE
839 ifdef CONFIG_HEADERS_CHECK
840         $(Q)$(MAKE) -f $(srctree)/Makefile headers_check
841 endif
842 ifdef CONFIG_SAMPLES
843         $(Q)$(MAKE) $(build)=samples
844 endif
845 ifdef CONFIG_BUILD_DOCSRC
846         $(Q)$(MAKE) $(build)=Documentation
847 endif
848         $(call vmlinux-modpost)
849         $(call if_changed_rule,vmlinux__)
850         $(Q)rm -f .old_version
```

```
688 vmlinux-init := $(head-y) $(init-y)
689 vmlinux-main := $(core-y) $(libs-y) $(drivers-y) $(net-y)
690 vmlinux-all  := $(vmlinux-init) $(vmlinux-main)
691 vmlinux-lds  := arch/$(SRCARCH)/kernel/vmlinux.lds
```

```
# Objects we will link into vmlinux / subdirs
init-y          := init/
drivers-y       := drivers/ sound/ firmware/
net-y           := net/
libs-y          := lib/
core-y          := usr/
```

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

- arch/arm/Makefile : 아키텍처 종속적인 부분

```
190 core-y                              += arch/arm/kernel/ arch/arm/mm/ arch/arm/common/
191 core-y                              += $(machdirs) $(platdirs)
192 core-$(CONFIG_FPE_NWFPE)            += arch/arm/nwfpe/
193 core-$(CONFIG_FPE_FASTFPE)          += $(FASTFPE_OBJ)
194 core-$(CONFIG_VFP)                   += arch/arm/vfp/
195
196 drivers-$(CONFIG_OPROFILE)          += arch/arm/oprofile/
197
198 libs-y                              := arch/arm/lib/ $(libs-y)
```

Building vmlinux

- vmlinux는 $(vmlinux-init)와 $(vmlinux-main)에서 정의된 오브젝트로 만들어진다.
- 각 오브젝트의 linking 순서가 중요.

링커스크립트 지정

```
  arm-linux-ld -EL  -p --no-undefined -X -o vmlinux -T arch/arm/kernel/vmlinux.lds arch/arm/kernel/head.o arch/arm/kernel/init_task.o  init/built-in.o --start-group  usr/built-in.o  arch/arm/kernel/built-in.o  arch/arm/mm/built-in.o  arch/arm/common/built-in.o  arch/arm/mach-s5pc100/built-in.o  arch/arm/plat-s5pc1xx/built-in.o  arch/arm/plat-s3c/built-in.o  arch/arm/nwfpe/built-in.o  arch/arm/vfp/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
```

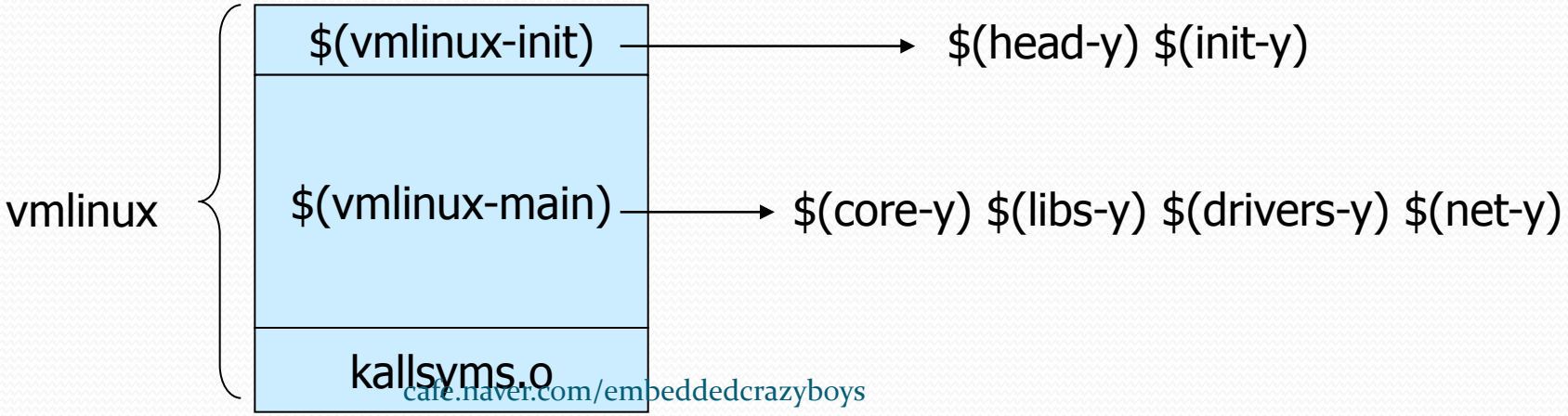
ld [옵션] 오브젝트파일 ..

-m : 어떤 포맷으로 출력물을 만들 것인가??

-T: 링커 스크립트 지정

--start-group ~ --end-group : ~에 지정된 오브젝트들 서로간에 변수나 함수 참조를 가능하게 함.

-o : 출력 파일명 지정



Building zImage (1/3)

1. arch/arm/boot/compressed/Image

$(topdir)/vmlinux에서 .note 섹션, .comment 섹션 및 모든 심볼들과 재배치 정보들을 제거한 후, 인스트럭션 데이터만을 뽑아, arch/arm/boot/compressed/Image 라는 바이너리 파일을 만든다.

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

objcopy [옵션] 입력파일 [출력파일]

-O *오브젝트형식*: 어떤 오브젝트 형식으로 출력 파일을 만들 것인지 지정 (예: elf32-i386, binary)

-R *섹션* : 출력 파일에서 해당 섹션을 지운다.

-S : 입력 파일의 재배치 정보와 심볼 정보를 출력 파일에 복사하지 않는다.

2. arch/arm/boot/compressed/piggy.gz

1단계에서 만든, Image을 가장 압축률이 좋은 방법으로 압축해서(-9), piggy.gz을 만듬

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

Building zImage (2/3)

3. arch/arm/boot/compressed/piggy.o

```
arm-linux-gcc -Wp,-MD,arch/arm/boot/compressed/.piggy.o.d  -nostdinc -isystem
/usr/local/arm/4.2.2-eabi/usr/bin-ccache/../lib/gcc/arm-unknown-linux-gnueabi/4.2.2/include
 -Iinclude  -I/home/icanjji/work/mango100/mango100_kernel_2010_06_30/arch/arm/include
 -include include/linux/autoconf.h -D__KERNEL__ -mlittle-endian
-Iarch/arm/mach-s5pc100/include -Iarch/arm/plat-s5pc1xx/include
-Iarch/arm/plat-s3c/include -D__ASSEMBLY__ -mabi=aapcs-linux -mno-thumb-interwork
-D__LINUX_ARM_ARCH__=7 -march=armv5t -Wa,-march=armv7-a  -msoft-float -gdwarf-2
-Wa,-march=all   -c -o arch/arm/boot/compressed/piggy.o arch/arm/boot/compressed/piggy.S
```

4. arch/arm/boot/compressed/vmlinux

head.o + misc.o + piggy.o 를 링킹해서, vmlinux를 만듬, 이 때 .text 섹션은 0x20008000 위치부터, 엔트리포인트는 startup_32로 지정한다.

```
  arm-linux-ld -EL    --defsym zreladdr=0x20008000 --defsym params_phys=0x20000100 -p --no-undefined -X /usr/loc
al/arm/4.2.2-eabi/usr/bin-ccache/../lib/gcc/arm-unknown-linux-gnueabi/4.2.2/libgcc.a -T arch/arm/boot/compressed
/vmlinux.lds arch/arm/boot/compressed/head.o arch/arm/boot/compressed/piggy.o arch/arm/boot/compressed/misc.o -o
 arch/arm/boot/compressed/vmlinux
```

ld [옵션] 오브젝트파일 ..

- -m *emulation* : 링커에게 해당 타겟 emulation에 맞는 정보를 제공 (예. 링커 스크립트 등)
- -r : 재할당 가능한 출력 파일을 생성. 즉, ld의 입력 오브젝트로 쓰일 수 있는 출력 파일을 생성. (실제로 piggy.o는 ld로 다시 링킹됨)
- --format input-format : 입력 오브젝트 파일의 형식 지정
- --oformat *output-format* : 출력 오브젝트 파일의 형식 지정.
- -o : 출력 파일명 지정
- -Ttext *org* : text 섹션의 시작주소를 org로 지정
- -e *entry* : 엔트리 포인트를 지정한다.

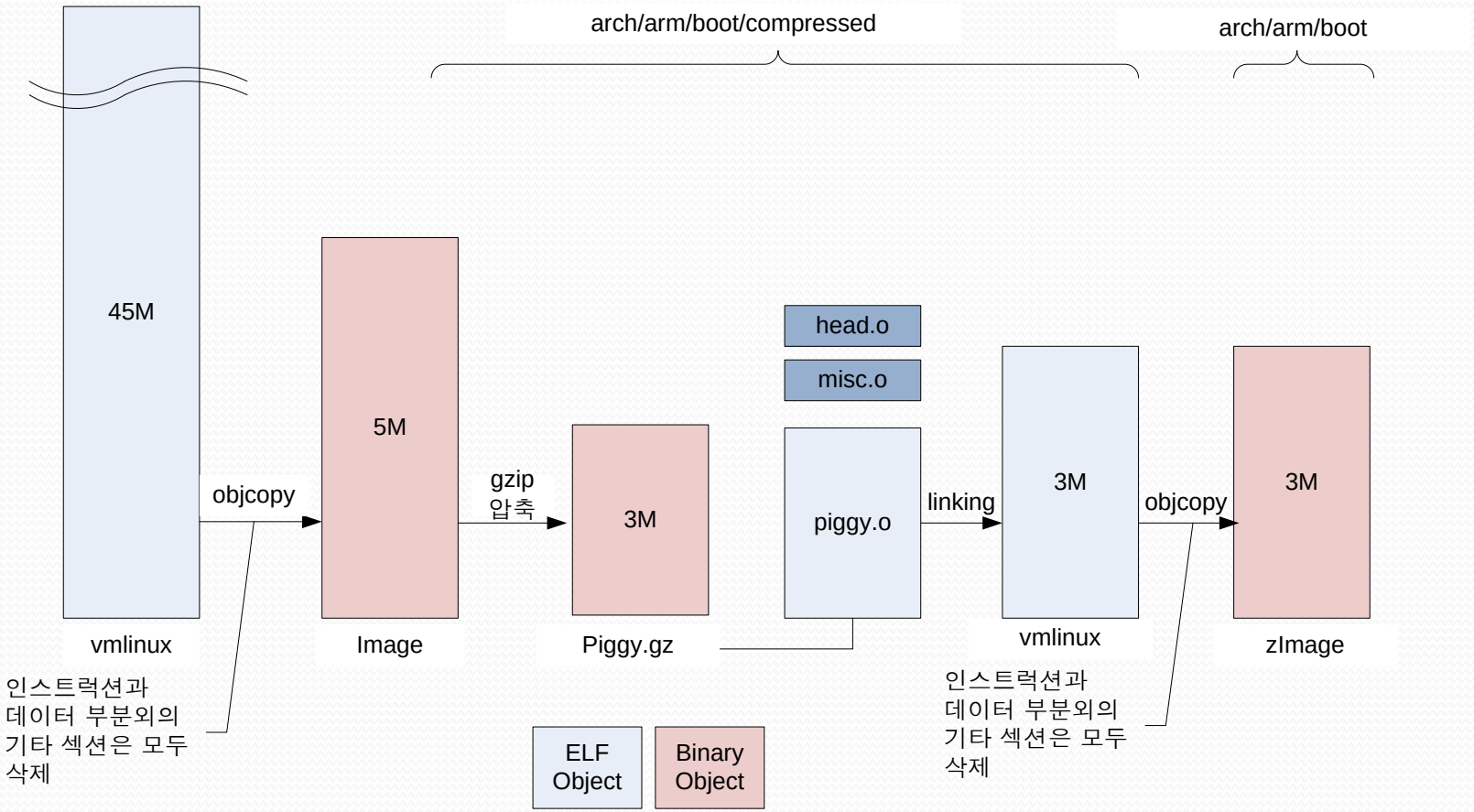
Building zImage (3/3)

5. arch/arm/boot/zImage

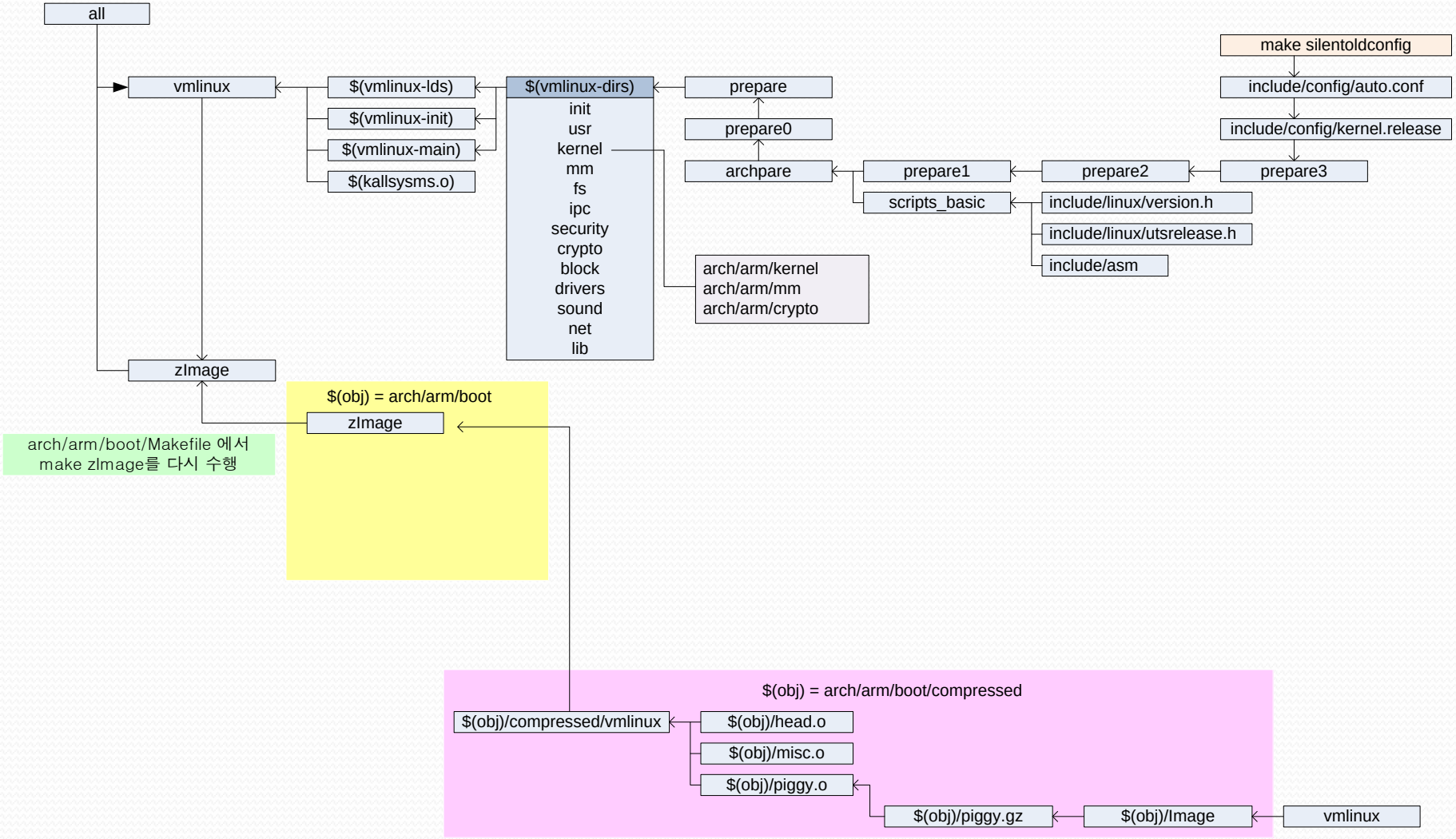
4단계에서 만든 vmlinux에서 .note 섹션, .comment 섹션 및 모든 심볼들과 재배치 정보들을 제거한 후, 인스트럭션 데이터만을 뽑아, arch/arm/boot/zImage 라는 바이너리 파일을 만든다.

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

결론 - Kernel Build Process



참고 : kernel Makefile 계층도



vmlinux.lds 첫 부분.

```
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
 . = 0;
 _text = .;

 .text : {
  _start = .;
  *(.start)
  *(.text)
  *(.text.*)
  *(.fixup)
  *(.gnu.warning)
  *(.rodata)
  *(.rodata.*)
  *(.glue_7)
  *(.glue_7t)
  *(.piggydata)
  . = ALIGN(4);
 }
```

Start entry 포인트는 _start

(예제) readelf -l arch/arm/boot/compressed/vmlinux

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          2408208 (bytes into file)
  Flags:                             0x4000000, Version4 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         1
  Size of section headers:           40 (bytes)
  Number of section headers:         20
  Section header string table index: 17

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 008000 23e254 00  AX  0   0 32
  [ 2] .got              PROGBITS        0023e254 246254 000070 00  WA  0   0  4
  [ 3] .got.plt          PROGBITS        0023e2c4 2462c4 00000c 04  WA  0   0  4
  [ 4] .bss              NOBITS          0023e2d0 2462d0 008448 00  WA  0   0  4
  [ 5] .stack            PROGBITS        00246718 2462d0 001000 00   W  0   0  1
  [ 6] .comment          PROGBITS        00000000 2472d0 000012 00      0   0  1
  [ 7] .ARM.attributes   ARM_ATTRIBUTES  00000000 2472e2 00001b 00      0   0  1
```

참고 : 커널 빌드시, Log 남기기

- make V=1 ARCH=arm 2>&1 | tee log-kernel.txt