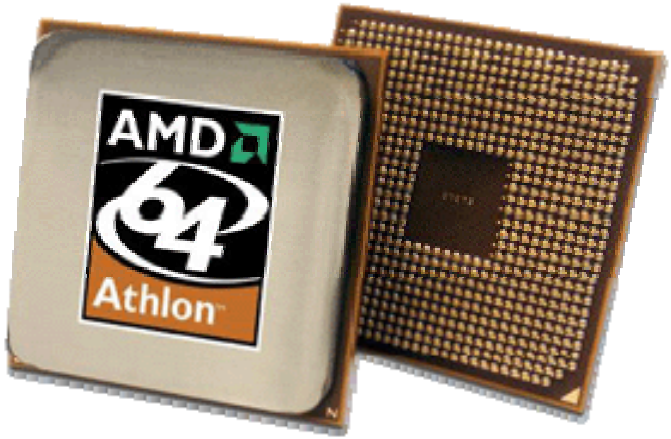
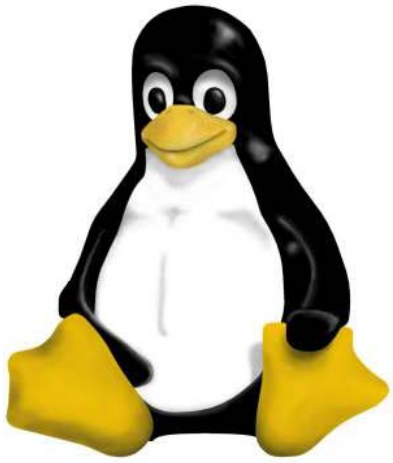



# kernel 2.6 makefile 분석

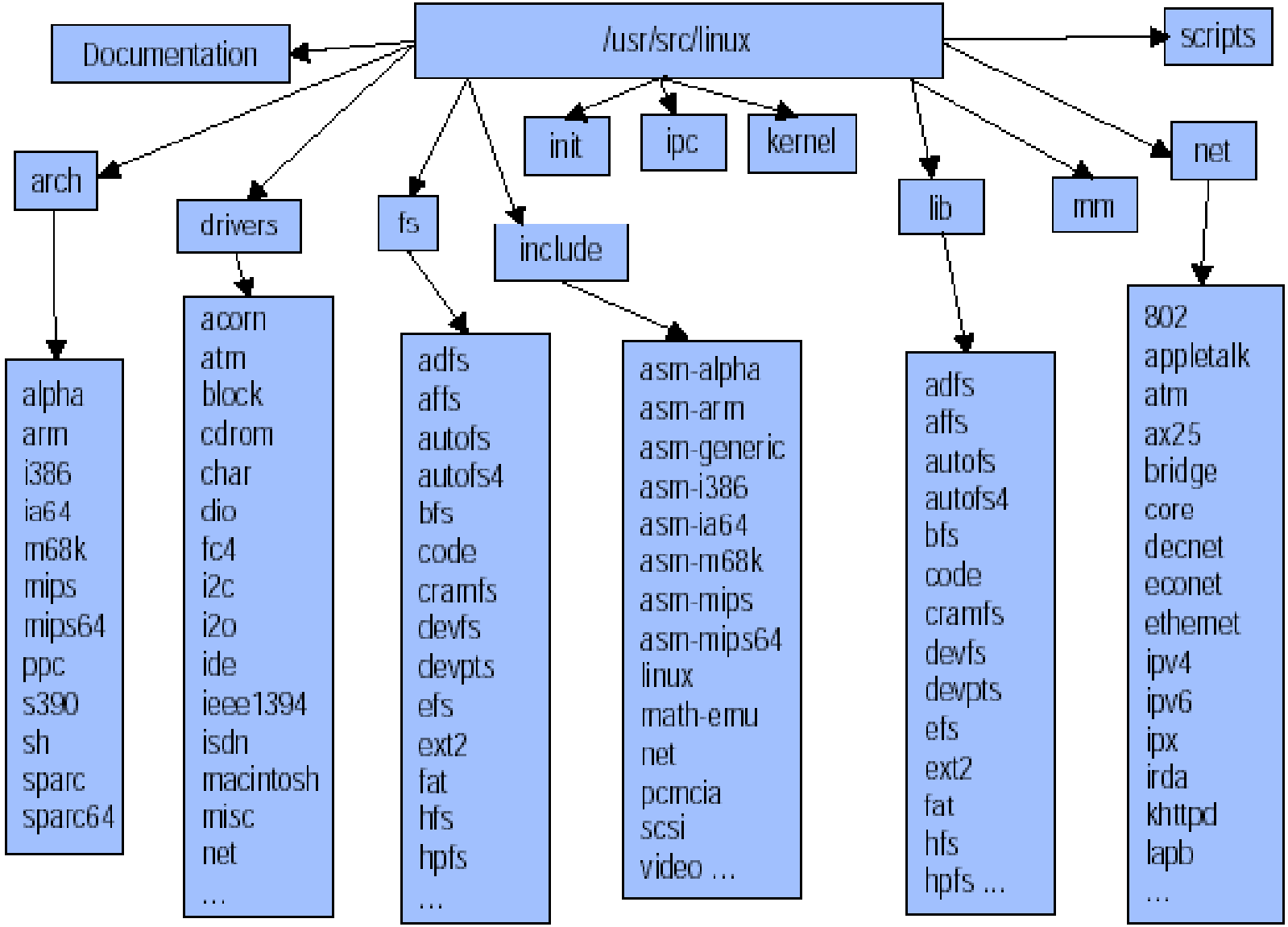
LKSAS 3기
송형주



# Kernel Makefile 제대로 분석하려면?

- GNU make, gcc, ld manual
- Kbuild System
  - (Documents/kubild/makefiles.txt)
- 쉘 스크립트
- Linux Binary Utilities (binutils)
- ELF file format
- 임베디드 시스템 엔지니어를 위한 리눅스 커널 분석 (http://kldp.org/KoreanDoc/html/EmbeddedKernel-KLDP/)
- 백창우, '유닉스,리눅스 프로그래밍 필수 유틸리티'
  - make, binutil 등 설명이 잘되어 있음. ^^

# Linux Kernel Source Tree



# kbuild overview

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

# Kbuild의 실행 절차

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(bzImage)를 만들기 위한 아키텍처에 따른 부분이 실행됨

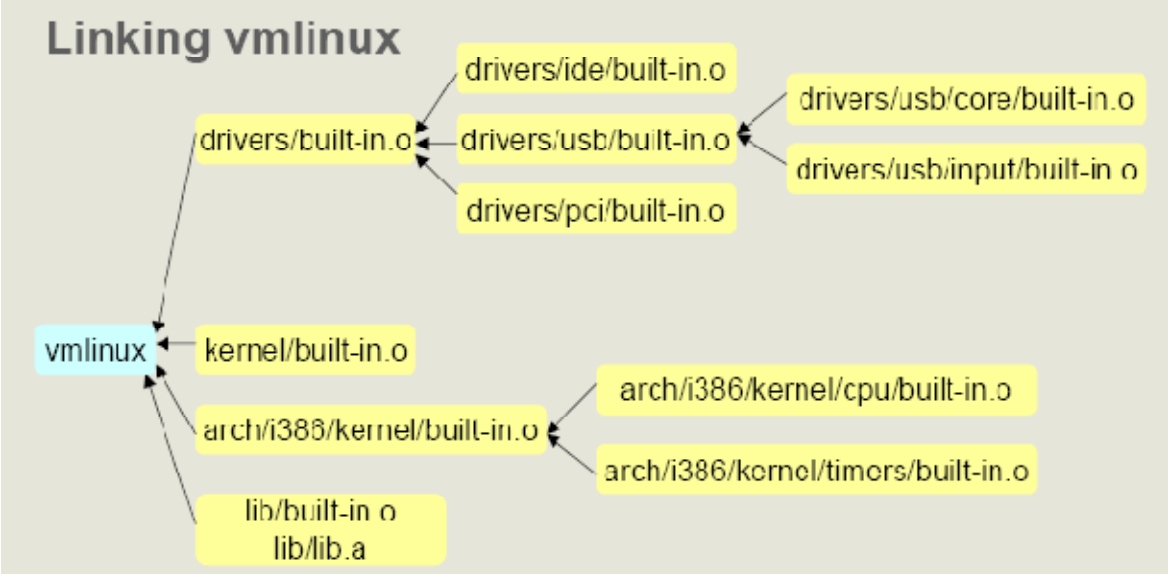
(Documentation/kbuild/makefile.txt 참조)

# 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



# Loadable module goals – (obj-m)

- object files which are built as loadable kernel modules.

```
24 obj-$(CONFIG_RT_MUTEXES) += rtmutex.o
25 obj-$(CONFIG_DEBUG_RT_MUTEXES) += rtmutex-debug.o
26 obj-$(CONFIG_RT_MUTEX_TESTER) += rtmutex-tester.o
27 obj-$(CONFIG_GENERIC_ISA_DMA) += dma.o
28 obj-$(CONFIG_SMP) += cpu.o spinlock.o
29 obj-$(CONFIG_DEBUG_SPINLOCK) += spinlock.o
30 obj-$(CONFIG_PROVE_LOCKING) += spinlock.o
31 obj-$(CONFIG_UID16) += uid16.o
32 obj-$(CONFIG_MODULES) += module.o
33 obj-$(CONFIG_KALLSYMS) += kallsyms.o
34 obj-$(CONFIG_PM) += power/
35 obj-$(CONFIG_BSD_PROCESS_ACCT) += acct.o
```

ex. $(TOP)/kernel/Makefile

```
123 CONFIG_MICROCODE=m
124 CONFIG_MICROCODE_OLD_INTERFACE=y
125 CONFIG_X86_MSR=m
126 CONFIG_X86_CPUID=m
127 CONFIG_X86_HT=y
128 CONFIG_X86_IO_APIC=y
129 CONFIG_X86_LOCAL_APIC=y
130 CONFIG_MTRR=y
131 CONFIG_SMP=m
132 CONFIG_SCHED_SMT=y
133 CONFIG_SCHED_MC=y
134 CONFIG_PREEMPT_NONE=y
```

ex. $(TOP)/.config

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

# Building non-kbuild targets (extra-y)

- obj-y에 지정된 타겟이 아닌, 현재 디렉토리에서 만들어지는 추가 타겟을 지정
- 즉, obj-y로 지정되지 않았기 때문에, built-in.o 로 같이 linking되지 않음.
- (내 생각) : 커널 빌드 시, 아키텍처 종속적인 부분의 처리를 위해서 사용하는 듯??

```
61 arch/x86_64/kernel/Makefile:5:extra-y   := head.o head64.o init_task.o vmlinux.lds
```

ex. $(TOP)/arch/x86_64/kernel/Makefile

# 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=x86_64

•커널 빌드시, 명령 및 메시지 출력 옵션

```
49 ifdef V
50   ifeq ("$(origin V)", "command line")
51     KBUILD_VERBOSE = $(V)
52   endif
53 endif
54 ifndef KBUILD_VERBOSE
55   KBUILD_VERBOSE = 0
56 endif
```

```
280 # If KBUILD_VERBOSE equals 0 then the above command will be hidden.
281 # If KBUILD_VERBOSE equals 1 then the above command is displayed.
282
283 ifeq ($(KBUILD_VERBOSE),1)
284   quiet =
285   Q =
286 else
287   quiet=quiet_
288   Q = @
289 endif
```

•커널 빌드시, 소스 코드 체크 옵션

```
67 ifdef C
68   ifeq ("$(origin C)", "command line")
69     KBUILD_CHECKSRC = $(C)
70   endif
71 endif
72 ifndef KBUILD_CHECKSRC
73   KBUILD_CHECKSRC = 0
74 endif
```

•빌드된 파일의 출력 디렉토리 지정

```
110 ifdef O
111   ifeq ("$(origin O)", "command line")
112     KBUILD_OUTPUT := $(O)
113   endif
114 endif
```

$(top)/Makefile

# How to build vmlinux ??

-Makefile : 아키텍처 독립적인 부분

```
793 # vmlinux image - including updated kernel symbols
794 vmlinux: $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) $(kallsyms.o) FORCE
795 ifdef CONFIG_HEADERS_CHECK
796     $(Q)$(MAKE) -f $(srctree)/Makefile headers_check
797 endif
798     $(call if_changed_rule,vmlinux__)
799     $(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost $@
800     $(Q)rm -f .old_version
801
```

```
656 vmlinux-init := $(head-y) $(init-y)
657 vmlinux-main := $(core-y) $(libs-y) $(drivers-y) $(net-y)
658 vmlinux-all  := $(vmlinux-init) $(vmlinux-main)
659 vmlinux-lds  := arch/$(ARCH)/kernel/vmlinux.lds
```

```
481 # Objects we will link into vmlinux / subdirs we need to visit
482 init-y      := init/
483 drivers-y   := drivers/ sound/
484 net-y       := net/
485 libs-y      := lib/
486 core-y      := usr/
```

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

- arch/x86_64/Makefile : 아키텍처 종속적인 부분

```
79 head-y := arch/x86_64/kernel/head.o arch/x86_64/kernel/head64.o arch/x86_64/kernel/init_task.o
80
81 libs-y                 += arch/x86_64/lib/
82 core-y                 += arch/x86_64/kernel/ \
83                         arch/x86_64/mm/ \
84                         arch/x86_64/crypto/
```

# Building vmlinux

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

링커스크립트 지정

```
ld -m elf_x86_64  -o vmlinux -T arch/x86_64/kernel/vmlinux.lds arch/x86_64/kernel/head.o
arch/x86_64/kernel/head64.o arch/x86_64/kernel/init_task.o  init/built-in.o --start-group  usr/built-in.o
arch/x86_64/kernel/built-in.o  arch/x86_64/mm/built-in.o  arch/x86_64/crypto/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
lib/lib.a  arch/x86_64/lib/lib.a  lib/built-in.o  arch/x86_64/lib/built-in.o  drivers/built-in.o  sound/built-
in.o  arch/x86_64/pci/built-in.o  net/built-in.o --end-group .tmp_kallsyms2.o
```

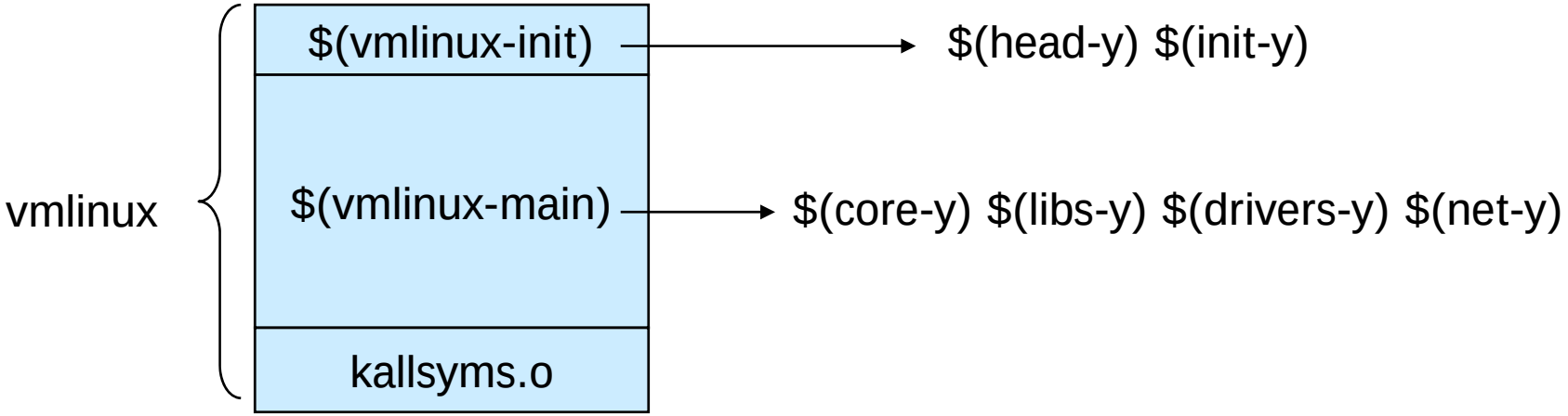
ld [옵션] 오브젝트파일 ..

-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 bzImage (1/3)

## 1. arch/x86_64/boot/compressed/vmlinux.bin

vmlinux에서 .note 섹션, .comment 섹션 및 모든 심볼들과 재배치 정보들을 제거한 후, 인스트럭션 데이터만을 뽑아, arch/x86_64/boot/compressed/vmlinux.bin 라는 바이너리 파일을 만든다.

```
objcopy -O binary -R .note -R .comment -S  vmlinux
arch/x86_64/boot/compressed/vmlinux.bin
```

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/x86_64/boot/compressed/vmlinux.bin.gz

1단계에서 만든, vmlinux.bin을 가장 압축률이 좋은 방법으로 압축해서(-9), vmlinux.bin.gz을 만듬

```
gzip -f -9 < arch/x86_64/boot/compressed/vmlinux.bin
> arch/x86_64/boot/compressed/vmlinux.bin.gz
```

# Building bzImage (2/3)

## 3. arch/x86_64/boot/compressed/piggy.o

링커 스크립트(vmlinux.scr)를 이용해서 vmlinux.bin.gz을 ELF 오브젝트 파일인 piggy.o로 링킹

```
ld -m elf_i386  -r --format binary --oformat elf32-i386
-T arch/x86_64/boot/compressed/vmlinux.scr arch/x86_64/boot/compressed/vmlinux.bin.gz
-o arch/x86_64/boot/compressed/piggy.o
```

## 4. arch/x86_64/boot/compressed/vmlinux

head.o + misc.o + piggy.o 를 링킹해서, vmlinux를 만듬, 이 때 .text 섹션은 0x100000 위치부터, 엔트리 포인트는 startup_32로 지정한다.

```
ld -m elf_i386  -Ttext 0x100000 -e startup_32 -m elf_i386
arch/x86_64/boot/compressed/head.o
arch/x86_64/boot/compressed/misc.o
arch/x86_64/boot/compressed/piggy.o
-o arch/x86_64/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 bzImage (3/3)

## 5. arch/x86_64/boot/vmlinux.bin

4단계에서 만든 vmlinux에서 .note 섹션, .comment 섹션 및 모든 심볼들과 재배치 정보들을 제거한 후, 인스트럭션 데이터만을 뽑아, arch/x86_64/boot/vmlinux.bin 라는 바이너리 파일을 만든다.

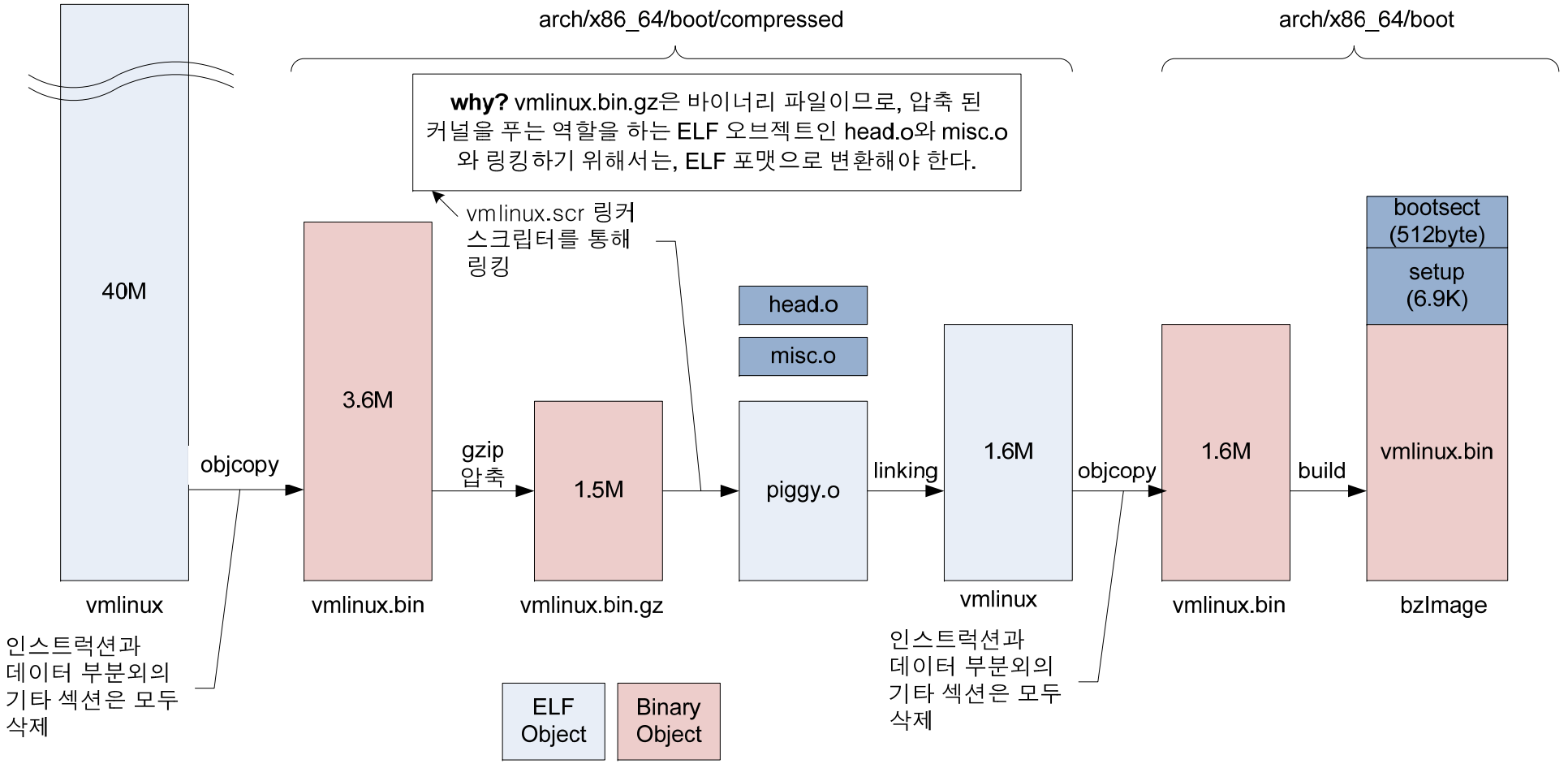
```
objcopy -O binary -R .note -R .comment -S  arch/x86_64/boot/compressed/vmlinux
arch/x86_64/boot/vmlinux.bin
```

## 6. arch/x86_64/boot/bzImage

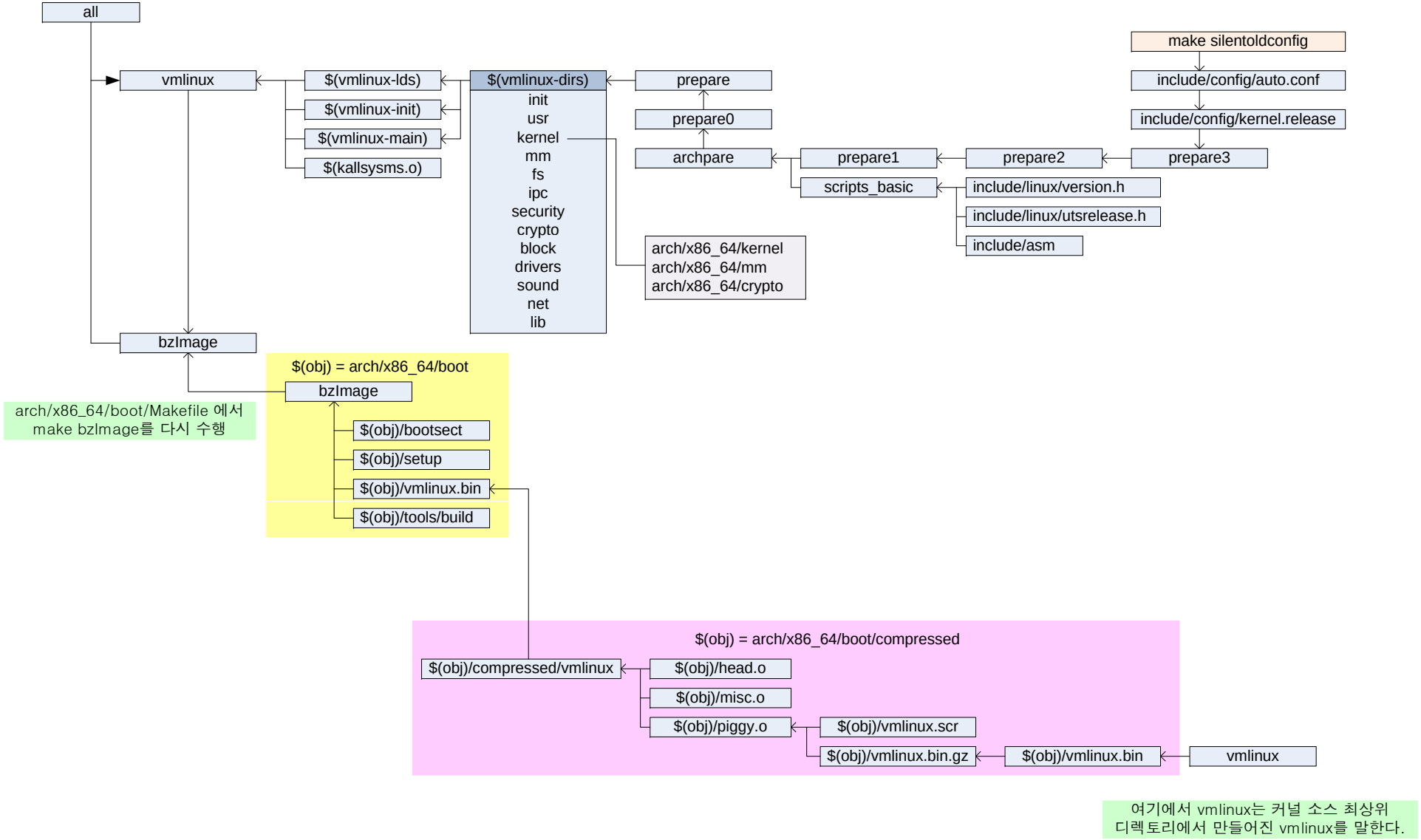
bootsect + setup + vmlinux.bin 를 합쳐서, bzImage 파일을 만든다.

```
arch/x86_64/boot/tools/build -b  arch/x86_64/boot/bootsect  arch/x86_64/boot/setup
arch/x86_64/boot/vmlinux.bin CURRENT > arch/x86_64/boot/bzImage
```

# 결론 - Kernel Build Process



# 참고 : kernel Makefile 계층도



# 참고 : kernel 2.6
# vmlinux.lds 링커 스크립터 분석

송형주

# vmlinux.lds 첫 부분.

```
12 OUTPUT_FORMAT("elf64-x86-64", "elf64-x86-64", "elf64-x86-64")
13 OUTPUT_ARCH(i386:x86-64)

/* 프로그램의 엔트리포인트를 지정 */
14 ENTRY(phys_startup_64)
15 jiffies_64 = jiffies;
16 _proxy_pda = 0;
```

- 실행 가능한 ELF 오브젝트 파일은 프로그램 헤더를 사용해서, 시스템 로더에서 읽히고, 메모리로 적재되는 방법을 설명
- 보통 디폴트 프로그램 헤더가 사용되지만, PHDRS 명령을 사용해 직접 정의 가능

*PHDRS 명령 문법*

name : 링커 스크립트의 SECTION 명령에서의 참조
type
PT_LOAD ( 이 프로그램 헤더가 파일로부터 로드되는 세그먼트를 기술함
PT_NOTE ( 참고 정보를 갖는 세그먼트)
FLAGS(flags) : 명시적으로 세그먼트 플래그를 지정 (flags 값은 정수여야함, R:4, W:2, E:1)

```
PHDRS
{
    name  type  [ FILEHDR ] [ PHDRS ] [ AT ( address ) ] [ FLAGS ( flags ) ] ;
}
```

```
17 PHDRS {
18        text       PT_LOAD   FLAGS(5);  /* R_E */
19        data       PT_LOAD   FLAGS(7);  /* RWE */
20        user       PT_LOAD   FLAGS(7);  /* RWE */
21        data.init  PT_LOAD   FLAGS(7);     /* RWE */
22        note       PT_NOTE   FLAGS(4);  /* R__ */
23 }
```

# Text Section 분석

• SECTIONS 는 출력 섹션들을 어떻게 배치해야 될지를 정하는 놈
  – 엔트리 포인트 정의
  – 심볼에 값 할당
  – 출력 섹션의 위치와 어떤 입력 섹션이 그 안으로 들어갈지 결정

여기서 '.' 은 특수한 링커 변수 *dot* 로 현재 출력 위치를 지정함.

__START_KERNEL = 0x100100 + 0xFFFFFFFF80000000 (LOAD_OFFSET)

```
24 SECTIONS
25 {
26   . = __START_KERNEL;
27   phys_startup_64 = startup_64 - LOAD_OFFSET;
28   _text = .;                  /* Text and read-only data */
29   .text :  AT(ADDR(.text) - LOAD_OFFSET) {
30       /* First the code that has to be first for bootstrapping */
31       *(.bootstrap.text)
32       /* Then all the functions that are "hot" in profiles, to group them
33          onto the same hugetlb entry */
34       #include "functionlist"
35       /* Then the rest */
36       *(.text)
37       SCHED_TEXT
38       LOCK_TEXT
39       KPROBES_TEXT
40       *(.fixup)
41       *(.gnu.warning)
42       } :text = 0x9090
43                           /* out-of-line lock text */
44   .text.lock : AT(ADDR(.text.lock) - LOAD_OFFSET) { *(.text.lock) }
45
46   _etext = .;                 /* End of text section */
```

링커에서 정의한 변수
외부 소스 파일에서 extern을 선언해 접근 가능

ADDR(section)
– 해당 secion의 VMA를 리턴

.text 섹션 정의

섹션을 프로그램 헤더에 의해 기술된 세그먼트에 할당하고,
정의되지 않은 영역은 0x9090의 패턴으로 채움.

# Output Section

```
section [address]  : [AT(lma)]
{
      output-section-command
      output-section-command
      …
} [:phdr][=fillexp]
```

- address
  - output section의 VMA(virtual memory address) 정의
- AT[lma]
  - output setction의 LMA(Load Memory address) 정의
- :phdr
  - Program Header에서 정의한 세그먼트에 해당 섹션을 위치 시킴
- =fillexp
  - 섹션 중에 정의되지 않은 영역을 해당값으로 채움

# Data Section, BSS Section

- .data Section

```
60                  /* Data */
61    .data : AT(ADDR(.data) - LOAD_OFFSET) {
62      *(.data)
63      CONSTRUCTORS
64      } :data
65
66    _edata = .;           /* End of data section */
```

- .bss Section

```
214    __bss_start = .;       /* BSS */
215    .bss : AT(ADDR(.bss) - LOAD_OFFSET) {
216      *(.bss.page_aligned)
217      *(.bss)
218      }
219    __bss_stop = .;
220
221    _end = . ;
```

BSS 영역은
compressed/head.S 에서
0으로 초기화 된다.

# 참고 - .init.text Section

```
159    . = ALIGN(4096);        /* Init code and data */
160    __init_begin = .;
161    .init.text : AT(ADDR(.init.text) - LOAD_OFFSET) {
162      _sinittext = .;
163      *(.init.text)
164      _einittext = .;
165    }
166    __initdata_begin = .;
167    .init.data : AT(ADDR(.init.data) - LOAD_OFFSET) { *(.init.data) }
168    __initdata_end = .;
```

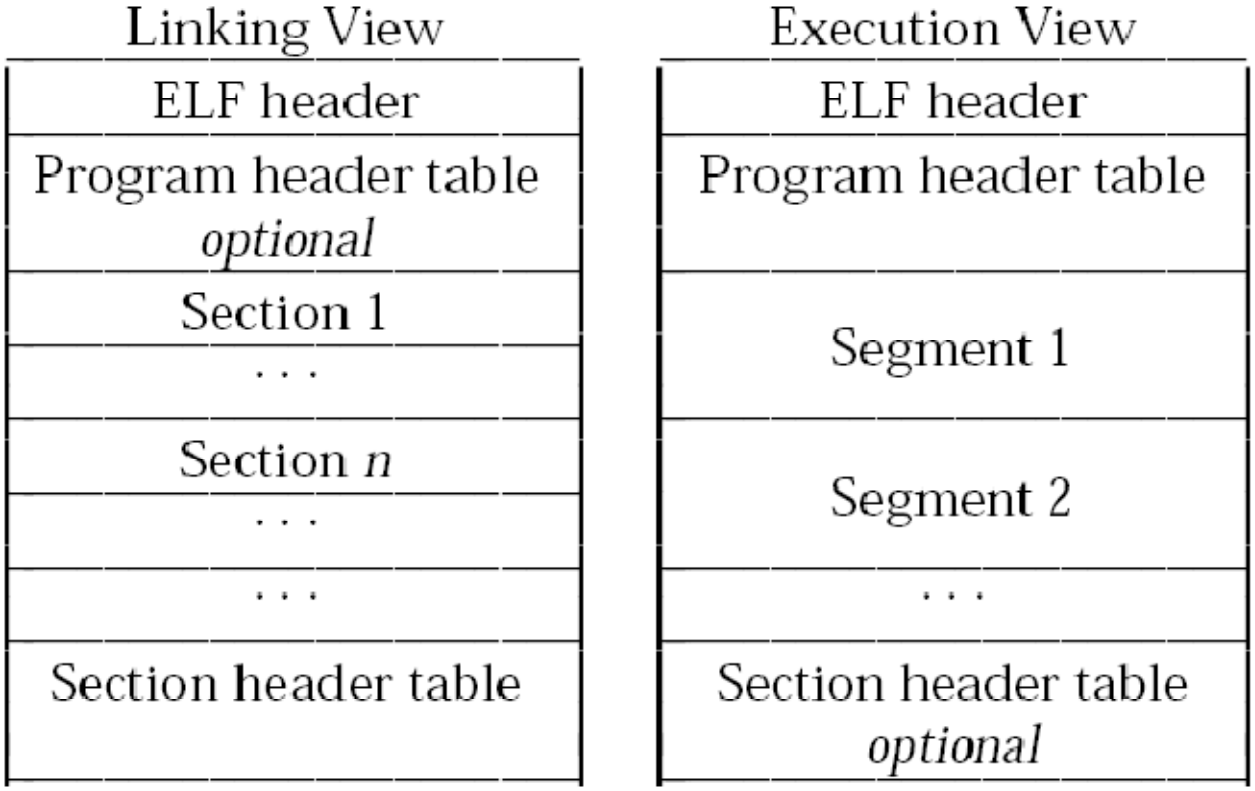
*Ex. arch/x86_64/kernel/head64.c 에서*

static void __init copy_bootdata(char *real_mode_data)
{ .. }

여기서 함수 앞에 쓰인 __init은 include/linux/init.h에 다음과 같이 정의되어 있다.
#define __init __attribute__ ((__section__ (".init.text")))

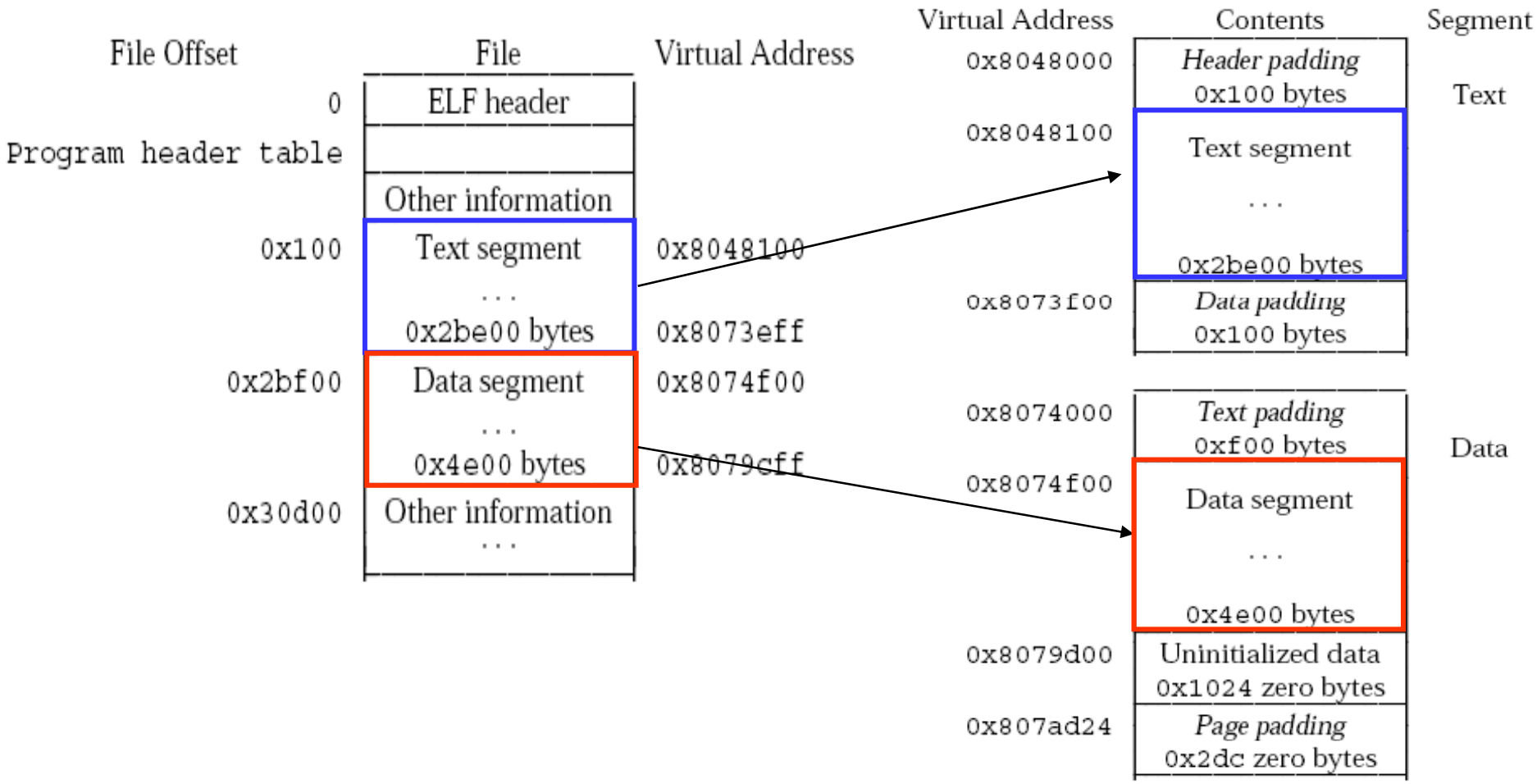
이 섹션들은 init 커널스레드의 free_initmem() 함수에서 삭제된다.

# ELF Format




Object files participate in program linking (building a program) and program execution (running a program).

# Program Loading



# (예제) readelf -l vmlinux

메모리에 올릴 수 있는 프로그램 세그먼트

보조정보 세그먼트

```
Elf file type is EXEC (Executable file)
Entry point 0x100100
There are 5 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000100000 0xffffffff80100000 0x0000000000100000
                 0x000000000002ea660 0x000000000002ea660  R E    200000

  LOAD           0x000000000003eb000 0xffffffff803eb000 0x000000000003eb000
                 0x000000000004a284 0x000000000004a284  RWE    200000

  LOAD           0x0000000000600000 0xffffffffff600000 0x0000000000436000
                 0x0000000000000c08 0x0000000000000c08  RWE    200000

  LOAD           0x0000000000638000 0xffffffff80438000 0x0000000000438000
                 0x000000000003b004 0x000000000000aaf44  RWE    200000

  NOTE           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  R      8
```

Section to Segment mapping:
 Segment Sections...
  00 .text __ex_table .rodata .pci_fixup __ksymtab __ksymtab_gpl __kcrctab __kcrctab_gpl __ksymtab_strings __param __bug_table
  01 .data .data.cacheline_aligned .data.read_mostly
  02 .vsyscall_0 .xtime_lock .vxtime .vgetcpu_mode .sys_tz .sysctl_vsyscall .xtime .jiffies .vsyscall_1 .vsyscall_2 .vsyscall_3
  03 .data.init_task .data.page_aligned .smp_locks .init.text .init.data .init.setup .initcall.init .con_initcall.init .security_initcall.init .altinstructions .altinstr_replacement .exit.text .init.ramfs .data.percpu .data_nosave .bss
  04

# 참고 : vmlinux.scr

```
SECTIONS
{
  .data : {
    input_len = .;
    LONG(input_data_end - input_data) input_data = .;
    *(.data)
    input_data_end = .;
    }
}
```

-압축된 vmlinux.bin.gz는 .data 섹션에 포함되므로, *(.data)로 표시된 곳에 들어가게 된다. 그 전후에 LONG(input-data-end - input-data)로 압축된 커널의 크기를 저장한다.

-위의 input_data 변수는 커널 압축을 푸는, arch/x86_64/boot/compressed/misc.c에서 이용됨.

# 참고 : 커널 빌드시, Log 남기기

- make V=1 ARCH=x86_64 2>&1 | tee log-bzImage.txt