Dive into kernel booting(lightning)

The self-salvation of a kernel engineer

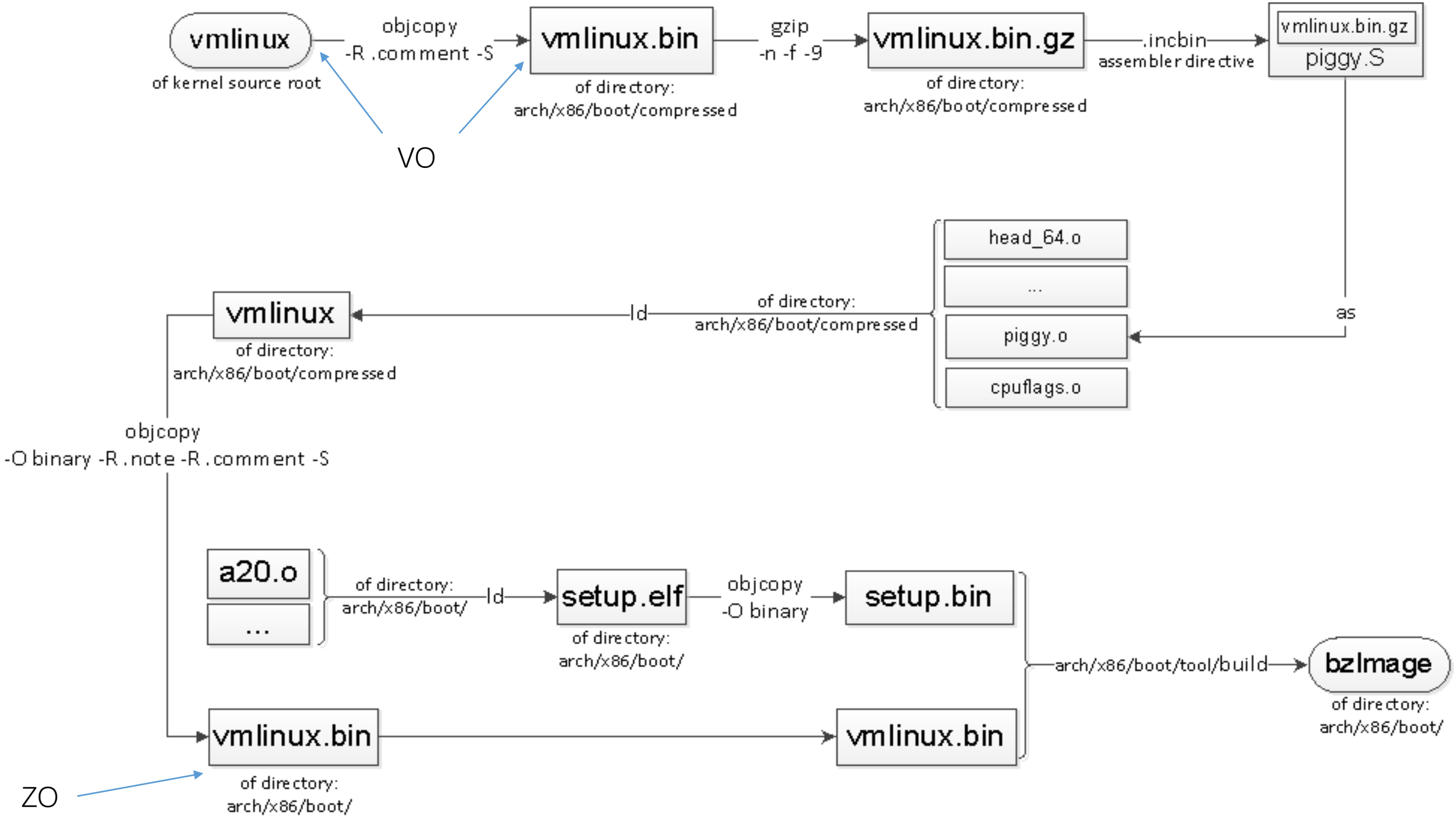
一个内核工程师的自我修养

Oct, 2019@CLK 2019

caoj.fnst@cn.fujitsu.com

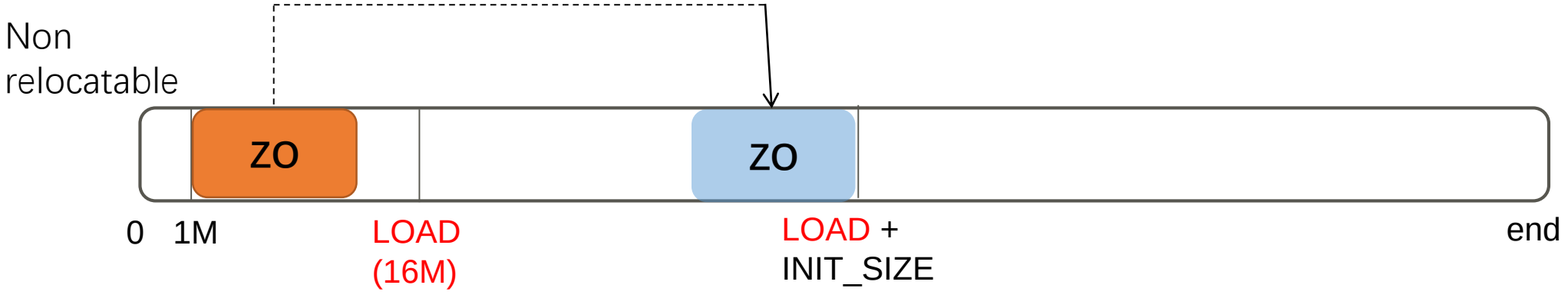
Ideas worth spreading

3 confirmed questions
+
1 unconfirmed one


vmlinux
of kernel source root
objcopy
-R .comment -S
vmlinux.bin
of directory:
arch/x86/boot/compressed
gzip
-n -f -9
vmlinux.bin.gz
of directory:
arch/x86/boot/compressed
.incbin
assembler directive
vmlinux.bin.gz
piggy.S
VO
head_64.o
...
piggy.o
cpuflags.o
as
ld
of directory:
arch/x86/boot/compressed
vmlinux
of directory:
arch/x86/boot/compressed
objcopy
-O binary -R .note -R .comment -S
a20.o
...
of directory:
arch/x86/boot/
ld
setup.elf
of directory:
arch/x86/boot/
objcopy
-O binary
setup.bin
arch/x86/boot/tool/build
bzImage
of directory:
arch/x86/boot/
vmlinux.bin
vmlinux.bin
ZO
of directory:
arch/x86/boot/

3 + 1: relocatable & randomize

- CONFIG_RELOCATABLE vs CONFIG_RANDOMIZE_BASE
  - RELOCATABLE: ZO can be loaded to arbitrary physical address than default 1M
  - RANDOMIZE_BASE: KASLR randomize phy & virt addresses of VO
  - RANDOMIZE_BASE depends on RELOCATABLE?



LOAD_PHYSICAL_ADDR = ALIGN(CONFIG_PHYSICAL_START, CONFIG_PHYSICAL_ALIGN)
Abbr as LOAD

3 + 1: decompressing kernel

- **Background**
  - In-place decompression

```
                                |-----compressed kernel image------|
                                V                                  V
0                           extract_offset                    +INIT_SIZE
|-----------|---------------|-------------------------|--------|
            |               |                         |        |
          VO__text      startup_32 of ZO          VO__end    ZO__end
            ^                                         ^

            |-------uncompressed kernel image---------|
```

  - Buffer size = VO__end – VO__text + extra space = INIT_SIZE
  - ZO image is moved against end of buffer
  - Data example: VO ≈ 26.5M; ZO ≈ 7.8M; INIT_SIZE ≈ 30.4M;
  - OVERLAP: (extract_offset, VO__end)

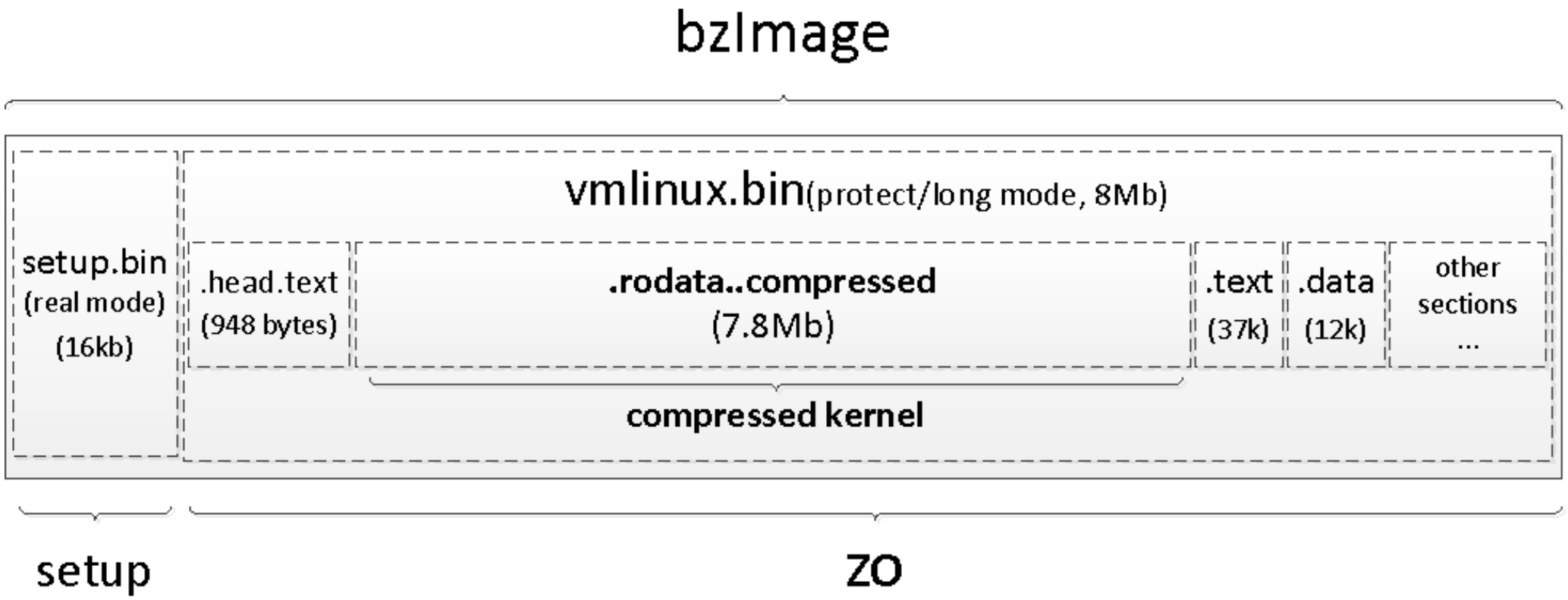
- **Q: Will the uncompressed kernel override the “jmp *%rax” in head_64.S?**

head_64.s: .head.text

```
/* Copy the compressed kernel to the end of our buffer
 * where decompression in place becomes safe. */
        pushq   %rsi
        leaq    (_bss-8)(%rip), %rsi
        leaq    (_bss-8)(%rbx), %rdi
        movq    $_bss /* - $startup_32 */, %rcx
        shrq    $3, %rcx
        std
        rep     movsq
        cld
        popq    %rsi

/* Jump to the relocated address. */
        leaq    relocated(%rbx), %rax
        jmp     *%rax

...
        .text
relocated:

...

/* Do the extraction, and jump to the new kernel.. */
        pushq   %rsi                    /* Save the real mode argument */
        movq    %rsi, %rdi              /* real mode address */
        leaq    boot_heap(%rip), %rsi   /* malloc area for uncompression */
        leaq    input_data(%rip), %rdx  /* input_data */
        movl    $z_input_len, %ecx      /* input_len */
        movq    %rbp, %r8               /* output target address */
        movq    $z_output_len, %r9      /* decompressed length, end of relocs */
        call    extract_kernel          /* returns kernel location in %rax */
        popq    %rsi

/* Jump to the decompressed kernel. */
        jmp     *%rax
```

```
SECTIONS
{
 . = 0;
 .head.text : {
  _head = . ;
  KEEP(*(.head.text))
  _ehead = . ;
 }
 .rodata..compressed : {
  *(.rodata..compressed)
 }
 .text : {
  _text = .;
  *(.text)
  *(.text.*)
  _etext = . ;
 }

/* SKIP */

 .data : {
  _data = . ;
  *(.data)
  *(.data.*)
  _edata = . ;
 }
 . = ALIGN((1 << (6)));
 .bss : {
  _bss = . ;
  *(.bss)
  *(.bss.*)
  *(COMMON)
  . = ALIGN(8);
  _ebss = .;
 }
```

Proportion example



3 + 1: Funny string operation in setup.bin

- Facts, in arch/x86/boot/
  - **memset** is defined as function in copy.S, and also defined as macro to __builtin_memset in string.h
  - All **memset** reference in setup will #include “string.h”
  - No **memset** entry in nm output of object file who reference **memset**,

- Q: why memset still need to be exist in copy.S as a function?

- Answer
  - setup.bin is compiled with –ffreestanding
  - Handling of GCC builtin functions depends on GCC version & flags
    - -mstringop-strategy=alg

3 + 1: Funny string operation in setup.bin

- Verify
  - Hack arch/x86/boot/compressed/Makefile with:

```
$(obj)/kaslr.o: KBUILD_CFLAGS += -mstringop-strategy=byte_loop
$(obj)/pgtable_64.o: KBUILD_CFLAGS += -mstringop-strategy=libcall
```

  - Check `nm *.o | grep mem` before and after hacking

String operation functions of boot/ are shared with boot/compressed

3 + 1

- Why CS/DS/ES/SS need to be cleared at startup_64?

```
# arch/x86/boot/compressed/head_64

ENTRY(startup_64)
        /* Setup data segments. */
        xorl    %eax, %eax
        movl    %eax, %ds
        movl    %eax, %es
        movl    %eax, %ss
        movl    %eax, %fs
        movl    %eax, %gs
```

- Answer from maintainer: PIC need them to be 0?

FUJITSU

shaping tomorrow with you

Bonus

- Facts, in arch/x86/kernel/ :
  - Input .head.text sections of VO scattered in 2 files: head_64.S & head64.C
  - Output .head.text section locates at the head of VO
  - **startup_64** of head_64.S is the entry of VO, means **.head.text section of head_64.S should be placed at the very head of VO**
- Question: how to assure?
- Answer: make sure head_64.o appear first to ld
- Verify: swap the order of these 2 files in arch/x86/Makefile

```
head-y := arch/x86/kernel/head_$(BITS).o
head-y += arch/x86/kernel/head$(BITS).o
```