1. vmlinux (bare kernel), ELF file

2. 由vmlinux (bare kernel) 生成binary file --- Image

```
arm-linux-gnueabi-objcopy -O binary -R .comment -S vmlinux Image
```

in arch/arm/Makefile

```
OBJCOPYFLAGS    :=-O binary -R .comment -S
```

OBJCOPYFLAGS为objcopy由vmlinux产生Image所用的options。

3. generate compressed vmlinux (假设CONFIG_KERNEL_GZIP=y)

in arch/arm/boot/compressed directory

3.1 get piggy.gzip from Image (like gzip -9 Image)

in arch/arm/boot/compressed/makefile

```
$(obj)/piggy.$(suffix_y): $(obj)/../Image FORCE
        $(call if_changed,$(suffix_y))
```

3.2 generate piggy.gzip.o from piggy.gzip

in arch/arm/boot/compressed/piggy.gzip.S

```
        .section .piggydata,#alloc
        .globl  input_data
input_data:
        .incbin "arch/arm/boot/compressed/piggy.gzip"
        .globl  input_data_end
input_data_end:
```

in arch/arm/boot/compressed/Makefile

```
$(obj)/piggy.$(suffix_y).o:  $(obj)/piggy.$(suffix_y) FORCE
```

3.3 generate vmlinux (compressed)

in arch/arm/boot/compressed/Makefile

```
$(obj)/vmlinux: $(obj)/vmlinux.lds $(obj)/$(HEAD) $(obj)/piggy.$(suffix_y).o \
                $(addprefix $(obj)/, $(OBJS)) $(lib1funcs) $(ashldi3) \
                $(bswapsdi2) FORCE
        @$(check_for_multiple_zreladdr)
        $(call if_changed,ld)
        @$(check_for_bad_syms)
```

这里的piggy.$(suffix_y).o即为piggy.gzip.o。

vmlinux是compressed vmlinux。

Notes:

---

compressed vmlinux的code是PIC(Position Independent Code),也就是它可以运行在任何地址，而不需要象

bare kernel一样运行在固定地址。

```
$ arm-linux-gnueabi-objdump -d vmlinux

vmlinux：     文件格式 elf32-littlearm



Disassembly of section .text:


00000000 <start>:
       0:       e1a00000        nop                     ; (mov r0, r0)
       4:       e1a00000        nop                     ; (mov r0, r0)
       8:       e1a00000        nop                     ; (mov r0, r0)
       c:       e1a00000        nop                     ; (mov r0, r0)
      10:       e1a00000        nop                     ; (mov r0, r0)
      14:       e1a00000        nop                     ; (mov r0, r0)
      18:       e1a00000        nop                     ; (mov r0, r0)
      1c:       e1a00000        nop                     ; (mov r0, r0)
      20:       ea000003        b       34 <start+0x34>
      24:       016f2818        .word   0x016f2818
      28:       00000000        .word   0x00000000
      2c:       002f5030        .word   0x002f5030
      30:       04030201        .word   0x04030201
      34:       e10f9000        mrs     r9, CPSR
      38:       eb000ff8        bl      4020 <__hyp_stub_install>
      3c:       e1a07001        mov     r7, r1
      40:       e1a08002        mov     r8, r2
```

```
    44:    e10f2000    mrs    r2, CPSR
    48:    e3120003    tst    r2, #3
    4c:    1a000001    bne    58 <not_angel>
    50:    e3a00017    mov    r0, #23
    54:    ef123456    svc    0x00123456
.....
```

compressed vmlinux的entry(start symbol)在0地址，这当然是不应该的。

in arch/arm/boot/compressed/Makefile

```
ccflags-y := -fpic -mno-single-pic-base -fno-builtin -I$(obj)
```

而bare kernel的entry就不是这样了。

```
$ arm-linux-gnueabi-objdump -d vmlinux
vmlinux：     文件格式 elf32-littlearm


Disassembly of section .head.text:

c0008000 <stext>:
c0008000:    eb003aee    bl    c0016bc0 <__hyp_stub_install>
c0008004:    e10f9000    mrs    r9, CPSR
c0008008:    e229901a    eor    r9, r9, #26
```

```
c000800c:       e319001f        tst     r9, #31
c0008010:       e3c9901f        bic     r9, r9, #31
c0008014:       e38990d3        orr     r9, r9, #211    ; 0xd3
c0008018:       1a000004        bne     c0008030 <stext+0x30>
c000801c:       e3899c01        orr     r9, r9, #256    ; 0x100
c0008020:       e28fe00c        add     lr, pc, #12
c0008024:       e16ff009        msr     SPSR_fsxc, r9
c0008028:       e12ef30e        .word   0xe12ef30e
c000802c:       e160006e        .word   0xe160006e
c0008030:       e121f009        msr     CPSR_c, r9
c0008034:       ee109f10        mrc     15, 0, r9, cr0, cr0, {0}
c0008038:       eb0001fe        bl      c0008838 <__lookup_processor_type>
c000803c:       e1b0a005        movs    sl, r5
c0008040:       0a000221        beq     c00088cc <__error_p>
......
```

bare kernel必须从0xc0008000开始运行！即bare kernel的code不是PIC的。

---

4. zImage

in arch/arm/boot/Makefile

```
$(obj)/zImage:  $(obj)/compressed/vmlinux FORCE
        $(call if_changed,objcopy)
        @$(kecho) '  Kernel: $@ is ready'
```

zImage是vmlinux(compressed)的binary

in temp/run.do_uboot_mkimage

```
arm-poky-linux-gnueabi-objcopy -O binary -R .note -R .comment -S arch/arm/boot/compressed/vmlinux linux.bin
```

5. uImage = 64 bytes header + zImage

in temp/run.do_uboot_mkimage

```
uboot-mkimage -A arm -O linux -T kernel -C none -a 0x00008000 -e $ENTRYPOINT -n "Poky (Yocto Project Reference Distro)/3.18.7+gitAUTOINC+e2438e08f1/granite2" -d linux.bin arch/arm/boot/uImage
```

uImage在zImage的文件头上加了64 bytes的header,其他完全一样。添加header由mkimage utility完成。

$ ls -l | egrep "[uz]Image"

-rw-r--r-- 1 walterzh walterzh 3100784 12月 25 21:47 uImage

-rwxrwxr-x 1 walterzh walterzh 3100720 12月 25 21:47 zImage

header的信息可以用mkimage utility查看

$ mkimage -l uImage

Image Name: Linux-3.18.7-yocto-standard

Created: Fri Dec 25 21:47:17 2015

Image Type: ARM Linux Kernel Image (uncompressed)

Data Size: 3100720 Bytes = 3028.05 kB = 2.96 MB

Load Address: 00008000

Entry Point: 00008000

Question: 这添加的64 bytes到底是什么呢?

in u-boot/include/image.h

```
/*
 * Legacy format image header,
 * all data in network byte order (aka natural aka bigendian).
 */
typedef struct image_header {
        __be32          ih_magic;       /* Image Header Magic Number    */
        __be32          ih_hcrc;        /* Image Header CRC Checksum    */
        __be32          ih_time;        /* Image Creation Timestamp     */
        __be32          ih_size;        /* Image Data Size              */
        __be32          ih_load;        /* Data  Load  Address          */
        __be32          ih_ep;          /* Entry Point Address          */
        __be32          ih_dcrc;        /* Image Data CRC Checksum      */
        uint8_t         ih_os;          /* Operating System             */
        uint8_t         ih_arch;        /* CPU architecture             */
        uint8_t         ih_type;        /* Image Type                   */
        uint8_t         ih_comp;        /* Compression Type             */
        uint8_t         ih_name[IH_NMLEN];      /* Image Name           */
} image_header_t;
```

sizeof(image_header_t) == 64

arch/arm/boot$ hexdump -C -n 64 uImage

00000000 27 05 19 56 39 71 9b 1f 56 7d 48 e5 00 2f 50 30 |'..V9q..V}H../P0|

00000010 00 00 80 00 00 00 80 00 2b f8 30 35 05 02 02 00 |........+.05....|

00000020 4c 69 6e 75 78 2d 33 2e 31 38 2e 37 2d 79 6f 63 |Linux-3.18.7-yoc|

00000030 74 6f 2d 73 74 61 6e 64 61 72 64 00 00 00 00 00 |to-standard.....|

00000040

这里数据以big endian存储。

.ih_magic = IH_MAGIC

in u-boot/include/image.h

#define IH_MAGIC   0x27051956   /* Image Magic Number   */

#define IH_NMLEN   32   /* Image Name Length   */

.ih_hcrc = 39 71 9b 1f,是image_header_t本身64 bytes的crc32校验值。

.ih_load = 00 00 80 00

.ih_ep   = 00 00 80 00

zImage(the binary of compressed vmlinux)将被载入到0x8000,并且入口同样为0x8000。

.ih_arch = 02 = IH_ARCH_ARM

#define IH_ARCH_ARM   2   /* ARM   */

.ih_name = Linux-3.18.7-yocto-standard

.ih_size = 00 2f 50 30,即zImage为0x2f5030 = 3100720 bytes

arch/arm/boot$ ls -l zImage

-rwxrwxr-x 1 walterzh walterzh 3100720 12月 25 21:47 zImage

与mkimage -l uImage看到的一样！