The Linux Kernel Internal

Ramin Farajpour Cami

Github : @raminfp

Twitter : @MF4rr3ll

Agenda

- Linux Kernel Booting Process
- How To Make Your Own Kernel OS
- Anatomy of Linux Kernel Development
- System Call
- Kernel Security (Bugs/Vulnerability) And Kernel Fuzzing

Who Am I?

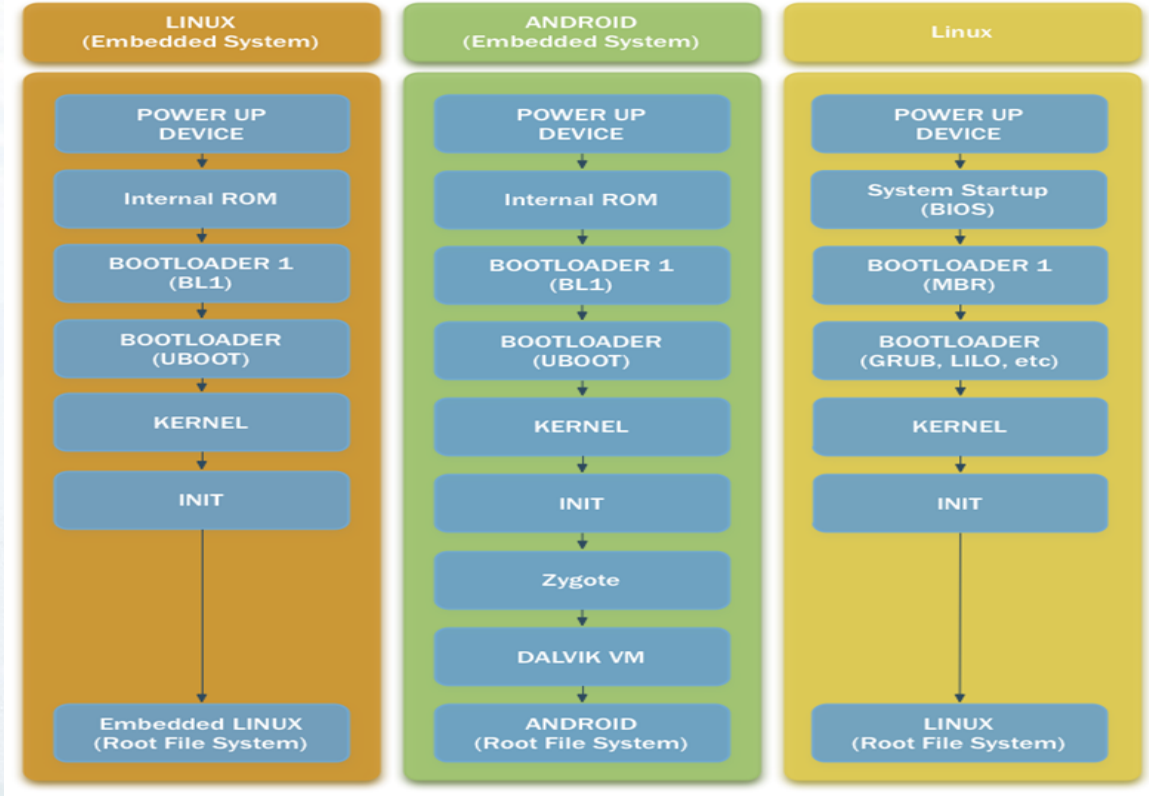
- Cyber Security Researcher
- Bug Bounty Program

  (Google, Twitter, Yahoo, Apple, Ebay, Blackberry, etc)
- Linux Exploit Developer
- Malware Analysis
- Linux System Programmer
- Linux Kernel/Device Developer
- Windows System Programmer
- Django Contributor

Type of OS

- Personal OS : Linux, Unix, MaCOS , etc
- Mobile OS : Android, iOS, WinPhone, UIQ, etc
- Real-Time OS : VxWork (NASA), QNX, RTLinux, etc
- Network OS : Router OS, Switch OS, etc
- Distributed OS : Internet, Telephone networks

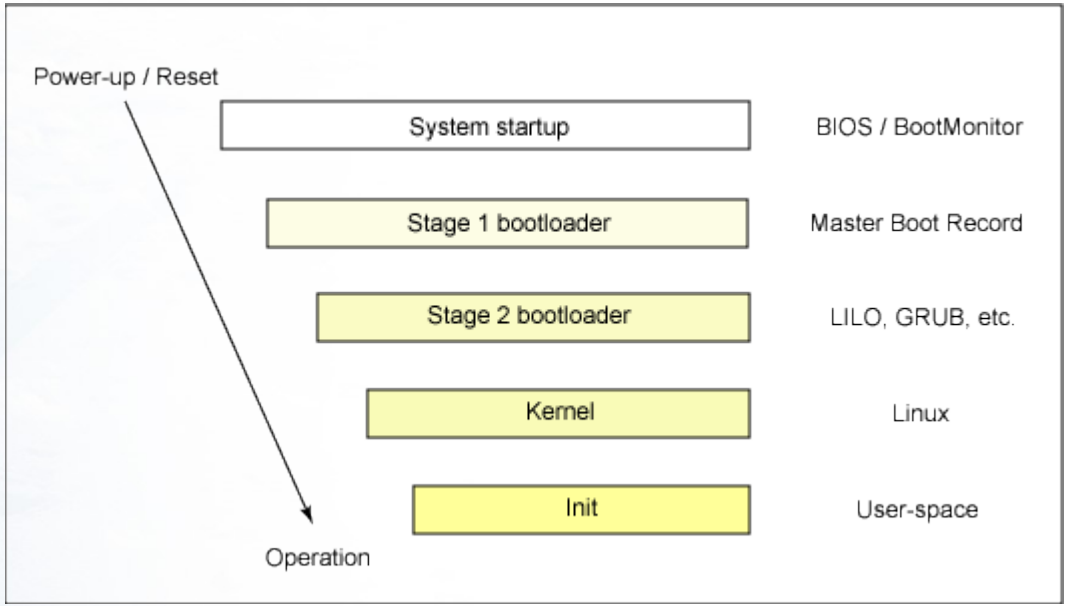
Linux Kernel



Android Device Linux Kernels Source

- Google: https://android.googlesource.com/kernel/msm/
- HTC: https://www.htcdev.com/devcenter/downloads
- OnePluseOS : https://github.com/OnePlusOSS/android_kernel_oneplus_msm8996
- Moto X : https://github.com/MotorolaMobilityLLC/kernel-msm
- Sony : https://github.com/sonyxperiadev/kernel

The Linux Boot Process



The Linux Boot Process

- BIOS (Basic Input/Output System)

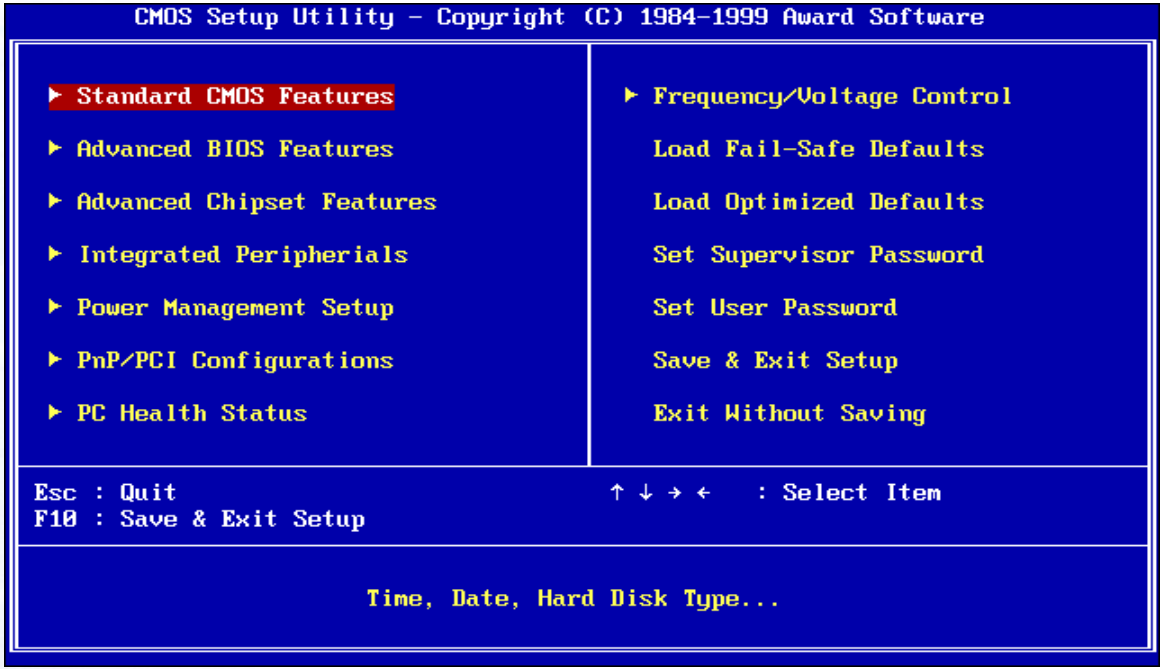
  The processor executes code at a well-known location In a personal computer (PC), which is stored in flash memory on the motherboard.

  booting Linux begins in the BIOS at address 0xFFFF0.

  When your computer boots—and after the POST finishes—the BIOS looks for a Master Boot Record, or MBR, stored on the boot device and uses it to launch the bootloader (GRUB).

The BIOS will soon be dead

- UEFI plans to completely replace it with UEFI on all their chipsets by 2020.



BIOS Tools

- apt-cache search bios | grep –i bios
- apt-get source phnxdeco (phonix tech)

- Security advisory (Intel):

  https://edk2-docs.gitbooks.io/security-advisory/content/

- Security Tool:

  Platform Security Assessment Framework

  https://github.com/chipsec/chipsec

- Attacking and Defending BIOS in 2015
- http://c7zero.info/stuff/AttackingAndDefendingBIOS-RECon2015.pdf

Bootkit Malware

What is bootkit?

Malware that executes before the operating system boots.

Malwares:

FIN1：Network protocols and communication channels for command and control (C2C).

Including: file transfer (http DLL web shell), screen capture, keystroke logging, process injection

BOOTRASH：

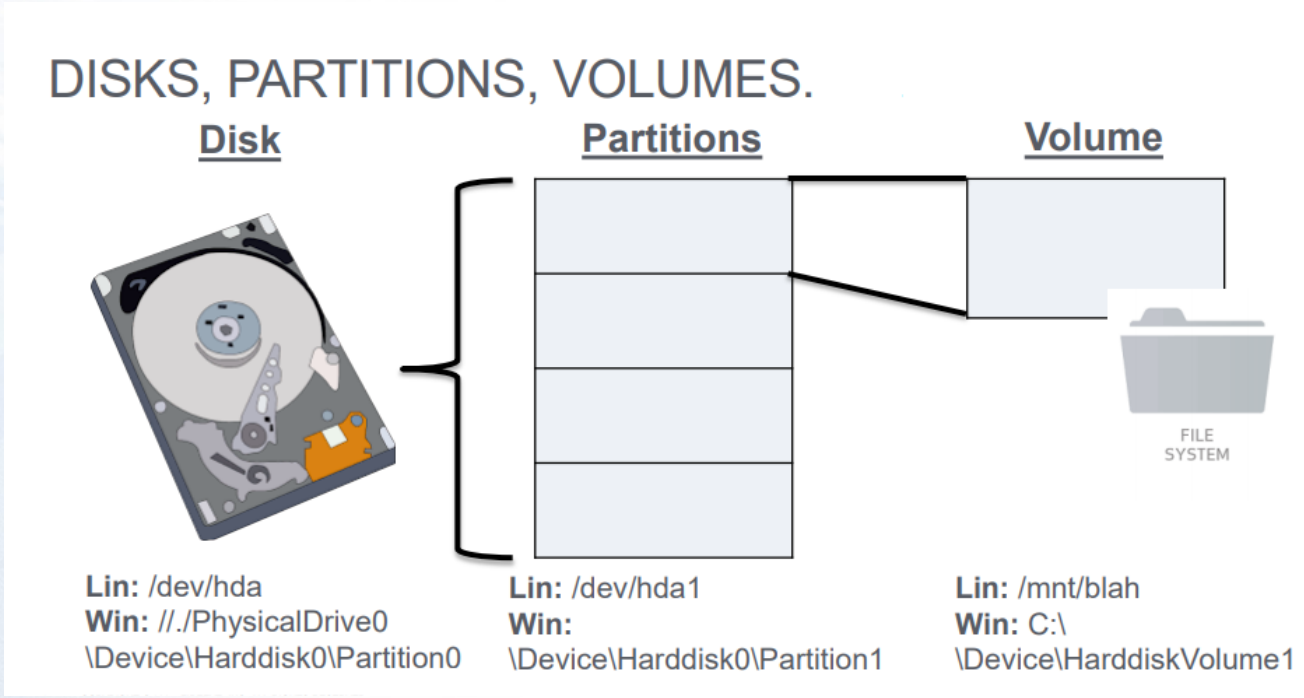
File Content : Core.sys, vfs.sys and etc

Including ：Services, Run keys, Scheduled tasks, Startup folders

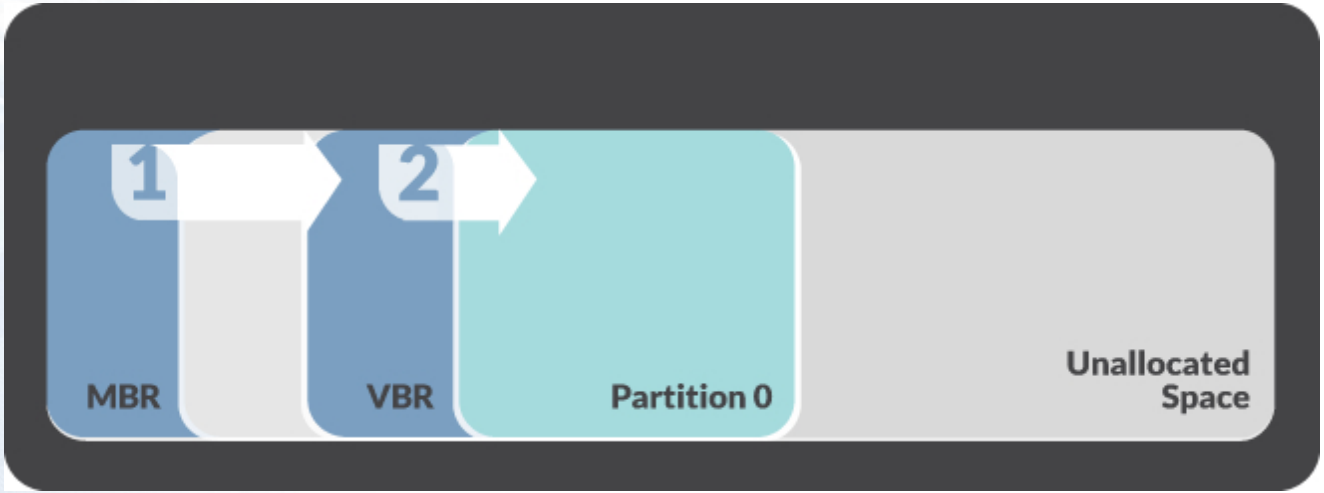
Article：

https://www.fireeye.com/blog/threat-research/2015/12/fin1-targets-boot-record.html

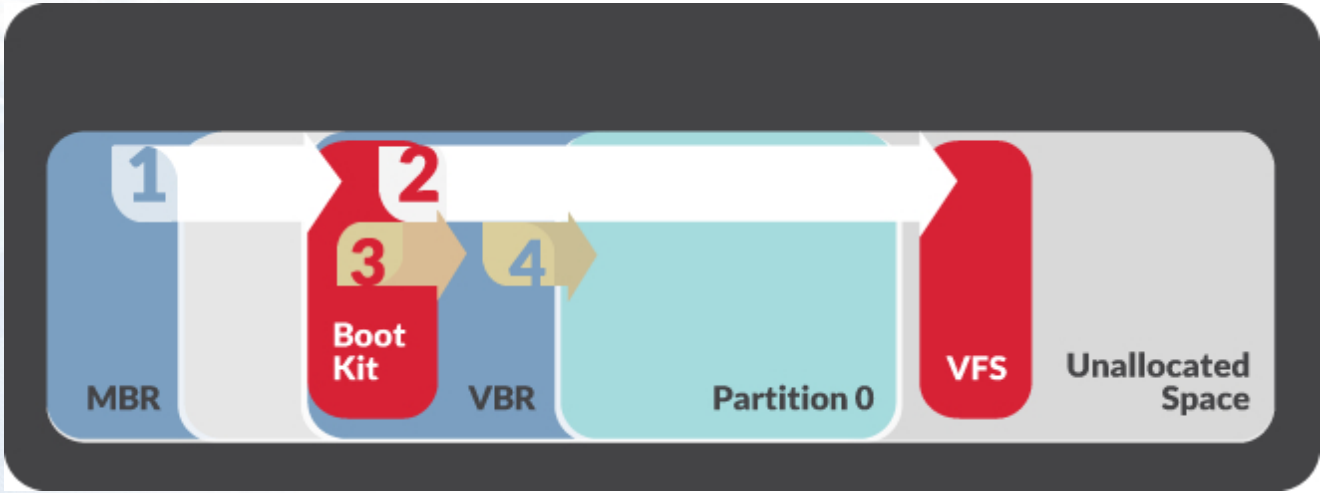
DISKS, PARTITIONS, VOLUMES



Normal boot process windows



Hijacked boot process



What is UEFI?

- You need to buy new hardware that supports and includes UEFI.
- UEFI firmware can boot from drives of 2.2 TB or larger.
- UEFI can run in 32-bit or 64-bit mode.
- Your boot process is faster.
- UEFI screens can be slicker than BIOS settings screens, including graphics and mouse cursor support.
- UEFI supports Secure Boot, which means the operating system can be checked for validity to ensure no malware has tampered with the boot process.
- UEFI support networking features.
- UEFI is modular.

- Analyzing UEFI BIOSes from attacking [BH 2014] (https://youtu.be/CGBpil0S5NI)

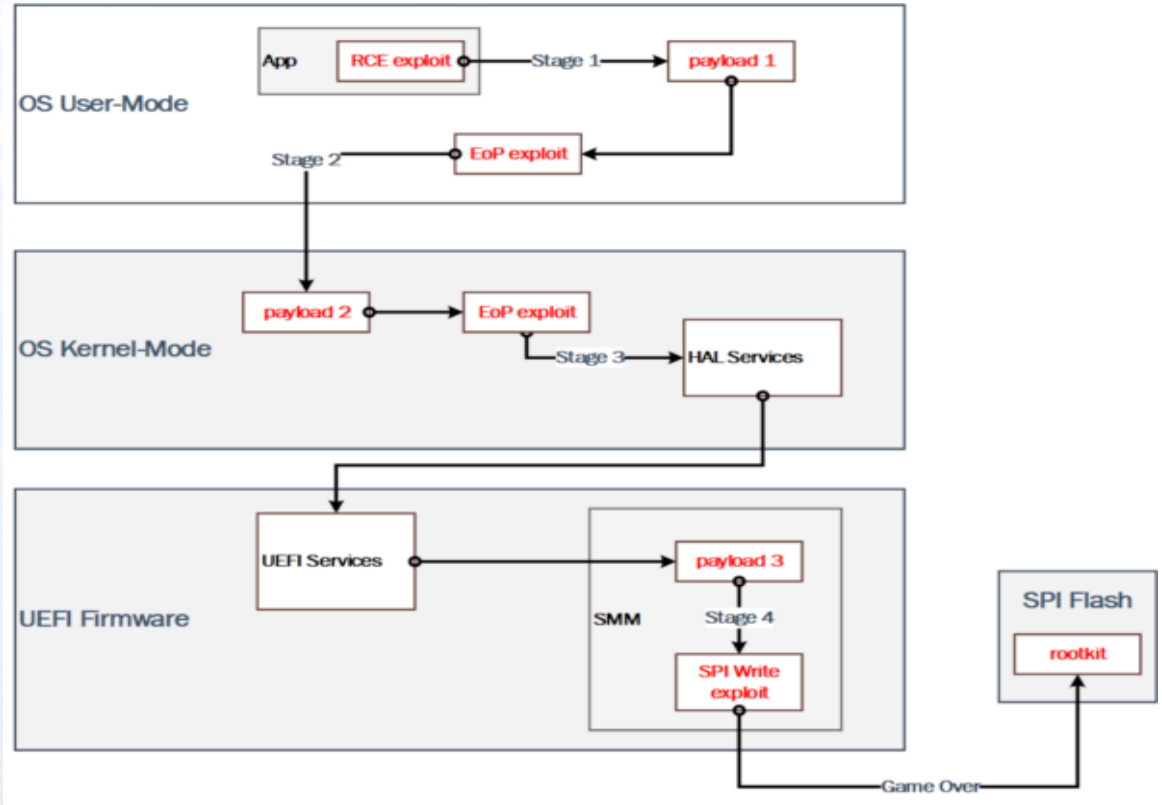
UEFI


EFI BIOS Utility - EZ Mode
Exit/Advanced Mode
06:06
Tuesday[11/16/2010]
P8P67 DELUXE
BIOS Version : 0304
CPU Type : Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
Total Memory : 2048 MB (DDR3 1333MHz)
English
Build Date : 10/21/2010
Speed : 3100 MHz
Temperature
CPU +116.6°F/+47.0°C
MB +96.8°F/+36.0°C
Voltage
CPU 1.200V
5V 5.120V
3.3V 3.408V
12V 12.288V
Fan Speed
CPU_FAN 1834RPM
PWR_FAN1 N/A
CHA_FAN1 N/A
CHA_FAN2 N/A
System Performance
Quiet
Performance
Energy Saving
Normal
Boot Priority
UEFI
UEFI
Use the mouse to drag or keyboard to navigate to decide the boot priority.
Boot Menu(F8)
Default(F5)

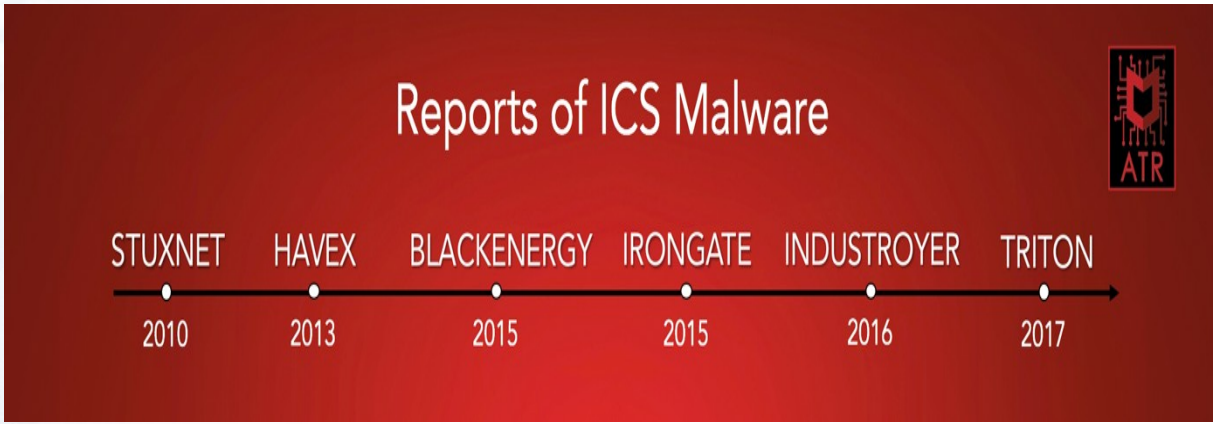
UEFI Rootkit Malware (**LoJax**)

- How malware works:
  - Information dumping tool:
  - *RwDrv.sys driver*, attackers reads the information on your UEFI BIOS. This information is then saved to a text file. This step helps the malware understand the victim system.
  - System Firmware Image Creation :
  - Firmware Image of SPI Flash Memory where the UEFI/BIOS is located. This image is then again saved to a file.
  - Rootkit Installation:
  - The firmware image is infected. This infected Firmware image is then installed onto the SPI Flash Memory

OS drivers dangerous for BIOS



Create malware is important for country



ICS (Industrial Control Systems) malware

- 2010 Stuxnet : This cyber weapon was created to target Iranian centrifuges.
- 2013 Havex : Targeted energy grids, electricity firms, and many others.
- 2015 BlackEnergy : It targeted critical infrastructure and destroyed files stored on workstations and servers. In Ukraine.
- 2015 IronGate : It targeted Siemens control systems and had functionalities similar to Stuxnet's.
- 2016 Industroyer : The attack caused a second shutdown of Ukraine's power grid.
- 2017 Triton : The attack did not succeed

What Is BootLoader?

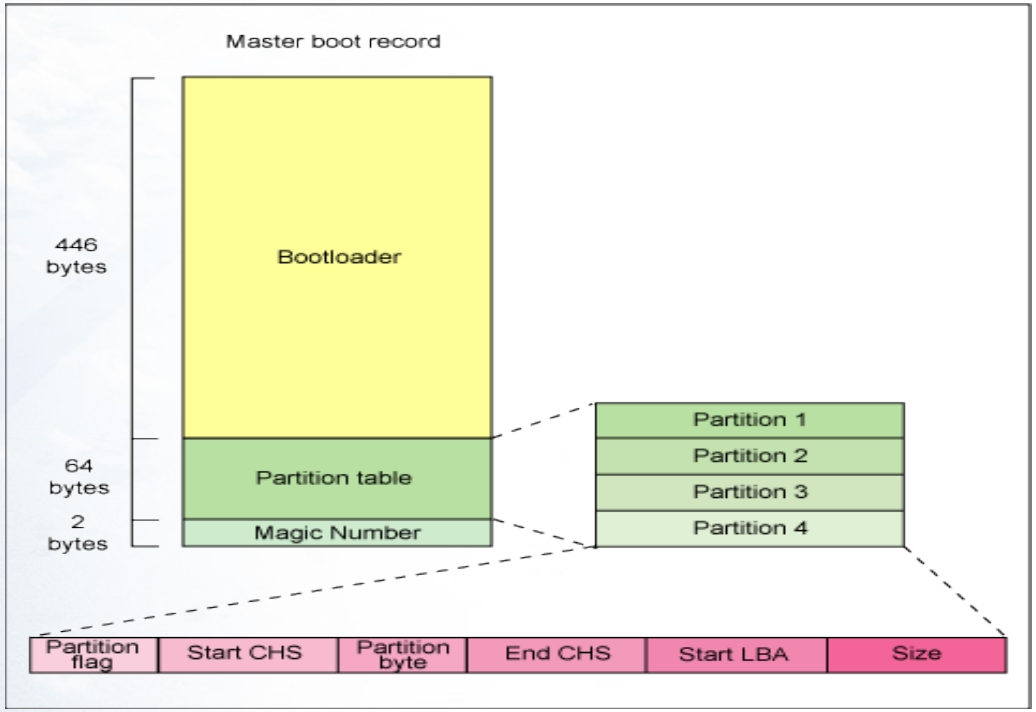
- It is the part that starts the system up and loads the operating system kernel

- Bootloader has two main jobs:
- [1] Initialize the system to a basic level (MBR) and to [2] Load the kernel.

Type Bootloder

Name	Main architectures supported
Das U-Boot	ARC, ARM, Blackfin, Microblaze, MIPS, Nios2, OpenRiec, PowerPC, SH
Barebox	ARM, Blackfin, MIPS, Nios2, PowerPC
GRUB 2	X86, X86_64
Little Kernel	ARM
RedBoot	ARM, MIPS, PowerPC, SH
CFE	Broadcom MIPS
YAMON	MIPS

First stage bootloader = MBR



MBR (Master Boot Record)

- MBR :

  The first 446 bytes are the primary boot loader, which contains both executable code and error message text

  The next sixty-four bytes are the partition table, which contains a record for each of four partitions The MBR ends with two bytes that are defined as the magic number (0xAA55)

Extracting the MBR

 # dd if=/dev/hda of=mbr.bin bs=512 count=1

 # od -xa mbr.bin

```
[redacted]:~$ sudo dd if=/dev/sda of=mbr.bin bs=512 count=1
[sudo] password for [redacted]:
1+0 records in
1+0 records out
512 bytes copied, 0.0355089 s, 14.4 kB/s
[redacted]:~$
[redacted]:~$
[redacted]:~$ od -xa mbr.bin
0000000    0000    0000    0000    0000    0000    0000    0000    0000
         nul nul nul nul nul nul nul nul nul nul nul nul nul nul nul nul
*
0000700    0001    feee    ffff    0001    0000    6daf    7470    0000
         soh nul   n   ~ del del soh nul nul nul   /   m   p   t nul nul
0000720    0000    0000    0000    0000    0000    0000    0000    0000
         nul nul nul nul nul nul nul nul nul nul nul nul nul nul nul nul
*
0000760    0000    0000    0000    0000    0000    0000    0000    aa55
         nul nul nul nul nul nul nul nul nul nul nul nul nul nul   U   *
0001000
[redacted]:~$
```

Partition table information of MBR

- # file mbr.bin

mbr.bin: x86 boot sector; partition 1: ID=0x83, active, starthead 32, startsector 2048, 19451904 sectors; partition 2: ID=0x5, starthead 254, startsector 19455998, 2093058 sectors, code offset 0x63

Second stage bootloader

Splash screen is commonly displayed, and Linux and an optional initial RAM disk (temporary root file system) are loaded into memory.

second-stage, boot loader called the kernel loader. The task at this stage is to load the Linux kernel and optional initial RAM disk.

Second stage = GRUB and etc

Type of bootloader :

1- Grub

2- LILO

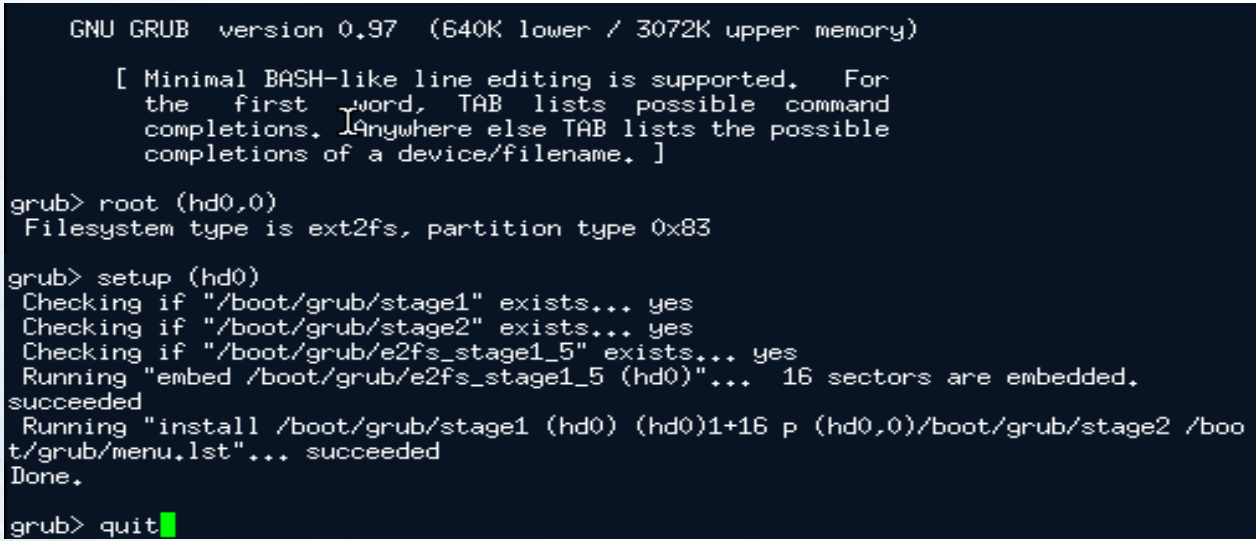
3- GRand

4- …

GRUB

Good knowledge of Linux file system. Instead of using raw sectors on the disk, as LILO.

GRUB can load a Linux kernel from an ext2 or ext3 file system



Stage 1 (MBR) + Stage 1.5 + Stage 2 (GRUB)

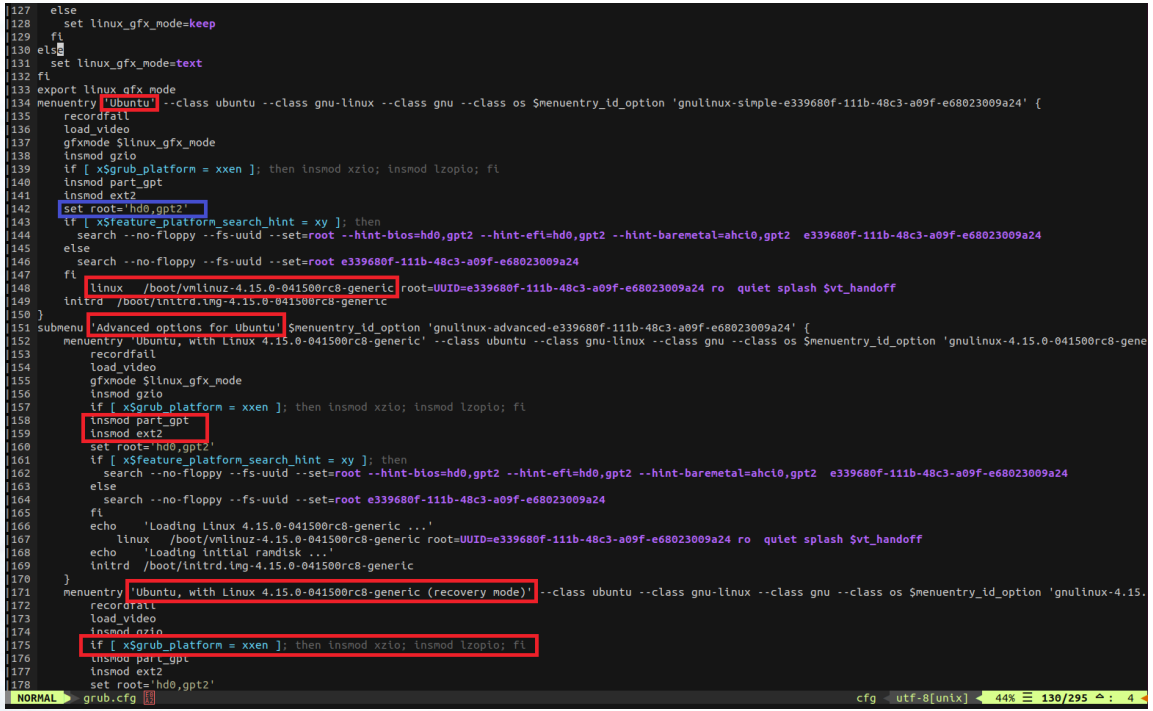
What is stage 1.5?

stage 1.5 boot loader that understands the particular file system containing the Linux kernel image.

Examples :

- CR-ROMs use the iso9660_stage_1_5
- Ext2 or ext3 file system use the e2fs_stage1_5

GRUB *.cfg (Stage 1.5 → Stage 2 loaded)



```
127   else
128     set linux_gfx_mode=keep
129   fi
130 else
131   set linux_gfx_mode=text
132 fi
133 export linux_gfx_mode
134 menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-simple-e339680f-111b-48c3-a09f-e68023009a24' {
135     recordfail
136     load_video
137     gfxmode $linux_gfx_mode
138     insmod gzio
139     if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
140     insmod part_gpt
141     insmod ext2
142     set root='hd0,gpt2'
143     if [ x$feature_platform_search_hint = xy ]; then
144       search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2  e339680f-111b-48c3-a09f-e68023009a24
145     else
146       search --no-floppy --fs-uuid --set=root e339680f-111b-48c3-a09f-e68023009a24
147     fi
148         linux   /boot/vmlinuz-4.15.0-041500rc8-generic root=UUID=e339680f-111b-48c3-a09f-e68023009a24 ro  quiet splash $vt_handoff
149     initrd  /boot/initrd.img-4.15.0-041500rc8-generic
150 }
151 submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-e339680f-111b-48c3-a09f-e68023009a24' {
152     menuentry 'Ubuntu, with Linux 4.15.0-041500rc8-generic' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-4.15.0-041500rc8-gene
153         recordfail
154         load_video
155         gfxmode $linux_gfx_mode
156         insmod gzio
157         if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
158         insmod part_gpt
159         insmod ext2
160         set root='hd0,gpt2'
161         if [ x$feature_platform_search_hint = xy ]; then
162           search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2  e339680f-111b-48c3-a09f-e68023009a24
163         else
164           search --no-floppy --fs-uuid --set=root e339680f-111b-48c3-a09f-e68023009a24
165         fi
166         echo    'Loading Linux 4.15.0-041500rc8-generic ...'
167             linux   /boot/vmlinuz-4.15.0-041500rc8-generic root=UUID=e339680f-111b-48c3-a09f-e68023009a24 ro  quiet splash $vt_handoff
168         echo    'Loading initial ramdisk ...'
169         initrd  /boot/initrd.img-4.15.0-041500rc8-generic
170     }
171     menuentry 'Ubuntu, with Linux 4.15.0-041500rc8-generic (recovery mode)' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-4.15.
172         recordfail
173         load_video
174         insmod gzio
175         if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
176         insmod part_gpt
177         insmod ext2
178         set root='hd0,gpt2'
NORMAL  grub.cfg                                                cfg  utf-8[unix]  44%  130/295  :  4
```

GRUB GUIDE

Refs:

https://thestarman.pcministry.com/asm/mbr/GRUB.htm

http://people.ds.cam.ac.uk/fanf2/hermes/src/grub-e1000/

https://www.gnu.org/software/grub/manual/grub/grub.html#General-boot-methods

https://github.com/coreos/grub/blob/93fb3dac4ae7a97c080d51d951d0e5a3109aaac7/grub-core/kern/main.c

Understanding the Various Grub Modules

$ ls /boot/grub/x86_64-efi/

/boot/grub/x86_64-efi/915resolution.mod

…

Grub module :

https://github.com/coreos/grub/tree/2.02-coreos/grub-core

Grub module error: file `/boot/grub/*/*.mod not found.

```
# GRUB loading.
# Welcome to GRUB!
#
# error: file `/boot/grub/i386-pc/normal.mod` not found.
# Entering rescue mode...
grub rescue> ls
hd(0) (hd0,msdos1)
grub rescue> set
cmdpath=(hd0)
prefix=(hd0,msdos1)/boot/grub
root=hd0,msdos1
grub rescue> set prefix=(hd0,msdos1)/usr/lib/grub
grub rescue> insmod normal
grub rescue> normal
grub> ls (hd0,msdos1) # Display UUID
grub> linux /boot/vmlinuz-linux root=UUID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx # Enter UUID
grub> initrd /boot/initramfs-linux.img
grub> boot
> pacman -Syyu
> grub-mkconfig -o /boot/grub/grub.cfg
> grub-install --recheck /dev/sda
> reboot
```

Load kernel image with GRUB

grub> **kernel /bzImage-<version>**

[Linux-bzImage, setup=0x1400, size=0x29672e]

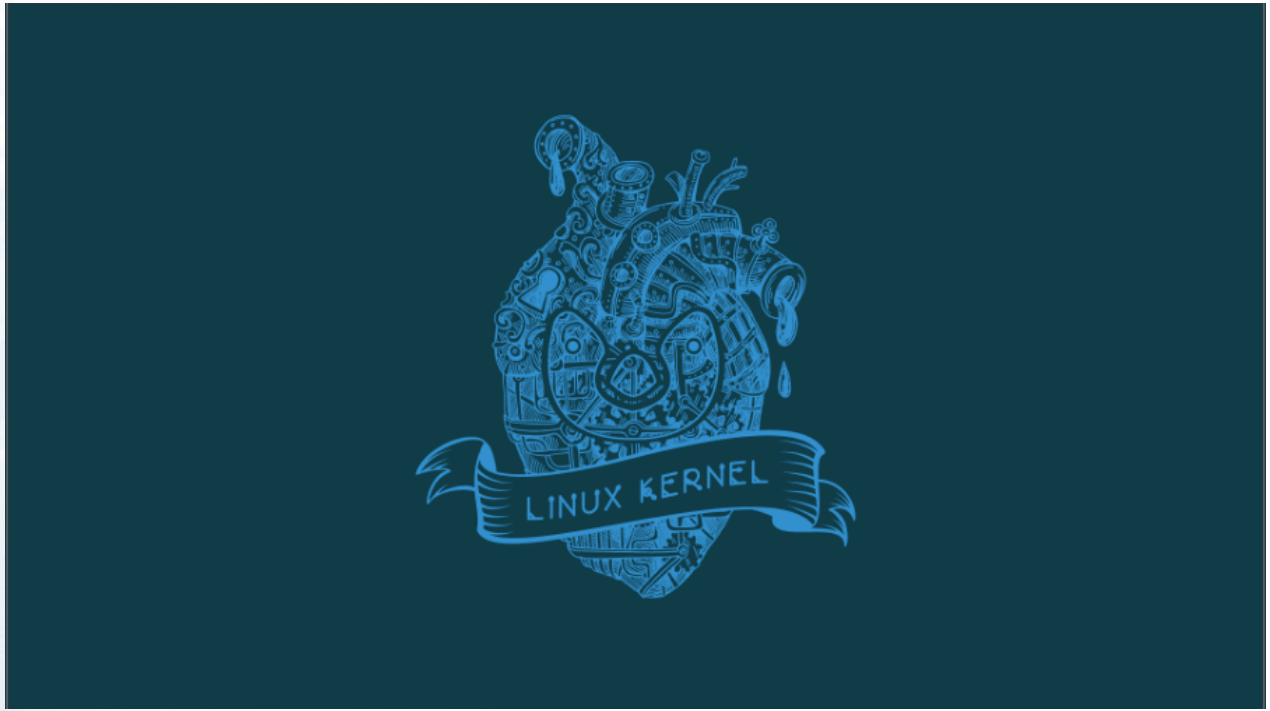
grub> **initrd /initrd-<version>.img**

[Linux-initrd @ 0x5f13000, 0xcc199 bytes]

grub> **boot**

Uncompressing Linux... Ok, booting the kernel.

Refs install manual : http://tinycorelinux.net/install_manual.html


LINUX KERNEL

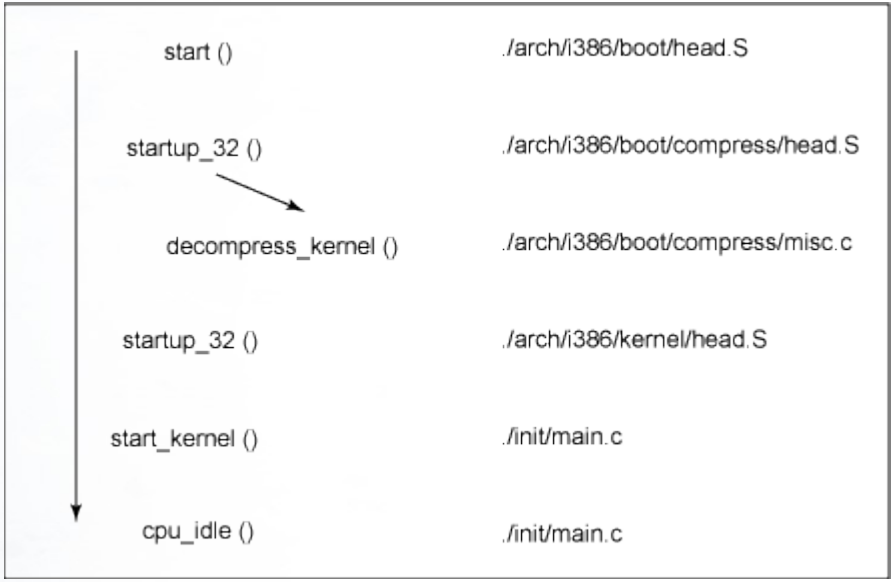
Count of Line Code Ubuntu Kernel

```
[redacted]:~/Desktop/kernel/linux-4.13.0$ cloc .
   62037 text files.
   61240 unique files.
Unescaped left brace in regex is deprecated here (and will be fatal in Perl 5.30), passed through in regex;
   11993 files ignored.

github.com/AlDanial/cloc v 1.70  T=369.78 s (135.4 files/s, 63323.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                            25558        2502431        2271988       12698359
C/C++ Header                 19873         495917         953724        3732465
Assembly                      1434          49270         114785         248172
JSON                           138              0              0          88336
Bourne Shell                   268          20145          11000          69947
make                          2424           8995           8517          39497
Perl                            66           5481           4104          29010
m4                             105           1572           1167          15862
Python                          80           2466           2852          13827
HTML                             3            565              0           4730
yacc                             9            682            357           4530
Bourne Again Shell              61            616            513           2940
lex                              8            302            300           1907
C++                              7            287             71           1838
awk                             13            186            179           1515
Markdown                         1            220              0           1077
TeX                              1            108              3            904
NAnt script                      2            158              0            602
Pascal                           3             49              0            231
Objective C++                    1             55              0            189
XSLT                             5             13             26             61
CSS                              1             14             23             35
YAML                             1              1              0             30
vim script                       1              3             12             27
Windows Module Definition        1              0              0              8
sed                              1              2              5              5
-------------------------------------------------------------------------------
SUM:                         50065        3089538        3369626       16956104
-------------------------------------------------------------------------------
```

Count of Line Code Main Kernel

```
[redacted]:~/Desktop/kernel$ cloc linux-4.19.tar.xz
   61693 text files.
   61264 unique files.
   12228 files ignored.

github.com/AlDanial/cloc v 1.70  T=401.07 s (123.4 files/s, 59120.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                            26082        2586769        2268941       13131964
C/C++ Header                 18856         491495         902716        3655384
Assembly                      1323          47331         106333         232938
JSON                           194              0              0         105175
make                          2389           8763           9449          38120
Bourne Shell                   377           6590           5904          28255
Perl                            55           5426           4004          27407
Python                         108           3093           3341          17645
HTML                             5            670              0           5497
yacc                             9            701            375           4648
lex                              8            326            314           2007
C++                              7            286             77           1844
Bourne Again Shell              51            352            318           1722
awk                             11            170            155           1386
Markdown                         1            220              0           1077
TeX                              1            108              3            915
NAnt script                      2            155              0            588
Windows Module Definition        2             14              0            102
m4                               1             15              1             95
XSLT                             5             13             26             61
CSS                              1             18             27             44
vim script                       1              3             12             27
Ruby                             1              4              0             25
INI                              1              1              0              6
sed                              1              2              5              5
-------------------------------------------------------------------------------
SUM:                         49492        3152525        3302001       17256937
-------------------------------------------------------------------------------
```

Kernel Boot Road Map



Underestand of Kernel Image

vmlinux: Plain linux ELF file just the way it was created by the linker, including symbols and everything.

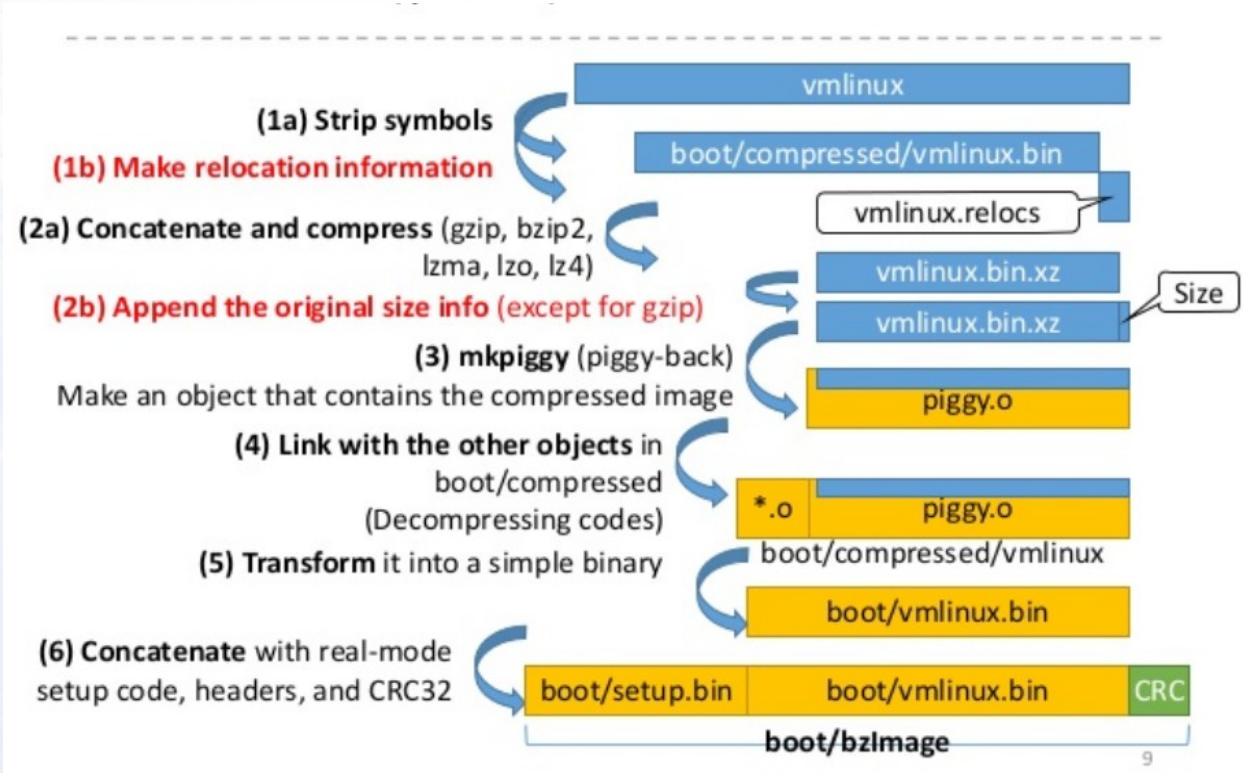
vmlinuz: Gzipped vmlinux file which got stripped of all its symbols

zImage: bootsect.o + setup.o + misc.o + piggy.o (piggy.o contains the piggy-backed vmlinuz).

zImage : is bootable because it can decompress and run the kernel it contains.

bzImage: Same as zImage except that it is built slightly differently which enables it to carry bigger kernels.

Vmlinux to Vmlinuz (make bzImage)



Vmlinux

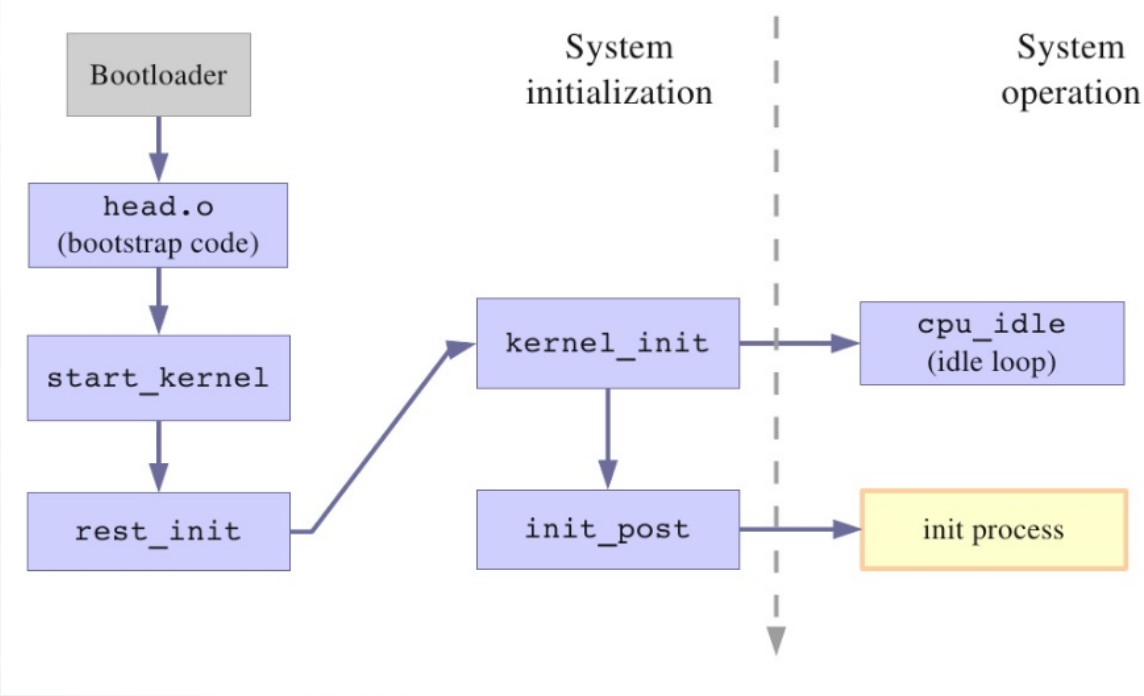
- Vmlinux is a ELF format,
- How to get ELF format?
- Download kernel source of https://kernel.com and compiled, or use $ apt-get source linux

Vmlinuz

- $ sudo file /boot/vmlinuz-4.15.0-041500rc8-generic

- /boot/vmlinuz-4.15.0-041500rc8-generic: Linux kernel x86 boot executable bzImage, version 4.15.0-041500rc8-generic (kernel@gloin) #201801142030 SMP Mon Jan 15 01:31:43 UTC 2018, RO-rootFS, swap_dev 0x7, Normal VGA

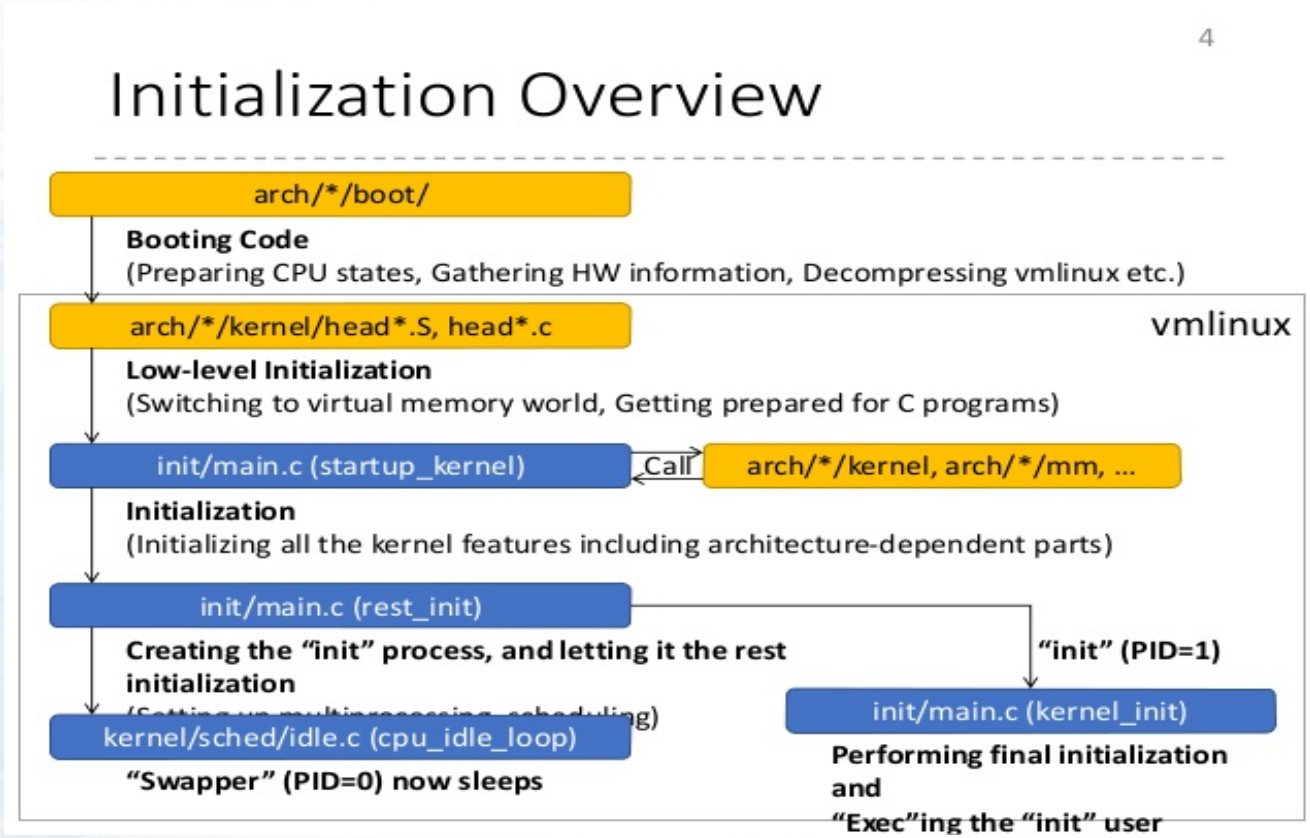
Kernel Start Up



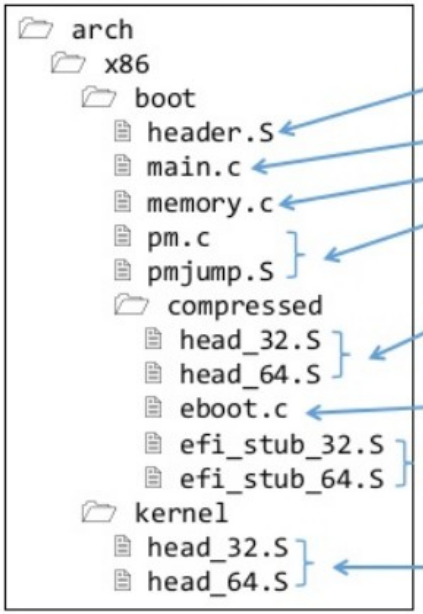
Kernel Image Process

- Kernel image (bzImage) load to Memory and kernel stage started …
- Typically zImage compressed image, less than 512KB a bzImage (big compressed image, greater than 512KB)
- When the bzImage (for an i386 image) is invoked, you begin at ./arch/i386/boot/head.S in the start assembly routine
- The kernel is then decompressed (./arch/i386/boot/compressed/misc.c) through a call to a C function called decompress_kernel function
- When the kernel is decompressed into memory, it is called. This is yet another startup_32 function, but this function is in ./arch/i386/kernel/head.S.
- More info : https://www.slideshare.net/itembedded/linux-kernel-image

Kernel Overview



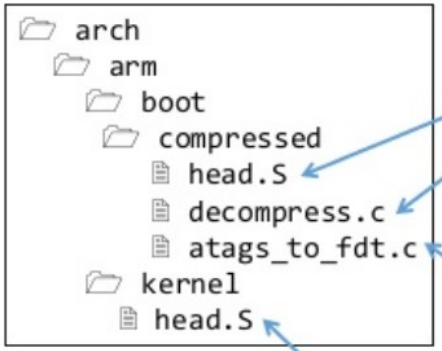
Kernel Source x86



- Real Mode (16-bit)
  - Boot sector, setup_header, and 16-bit entry point
  - C-Language main function
  - Retrieving memory information
  - Transition to the protected mode
- Protected Mode (32-bit)
  - 32-bit(/64-bit) entry point, preparing for decompression, calling decompression code
  - (EFI-Stub) efi_main (entry point from UEFI)
  - EFI call functions
- Protected Mode/Long Mode
  - The beginning of the main kernel

* The ..._32.S files are used in the 32-bit kernel, and ..._64.S files are not. Vice versa.

Kernel Source ARM



- Compressed
  - Entry point
  - Decompressing function
    - Actual decompressing algorithm is in lib/decompress_*.c
  - Building a FDT from ATAGS for compatibility (CONFIG_ARM_ATAG_DTB_COMPAT)
- Decompressed
  - The beginning of the main kernel

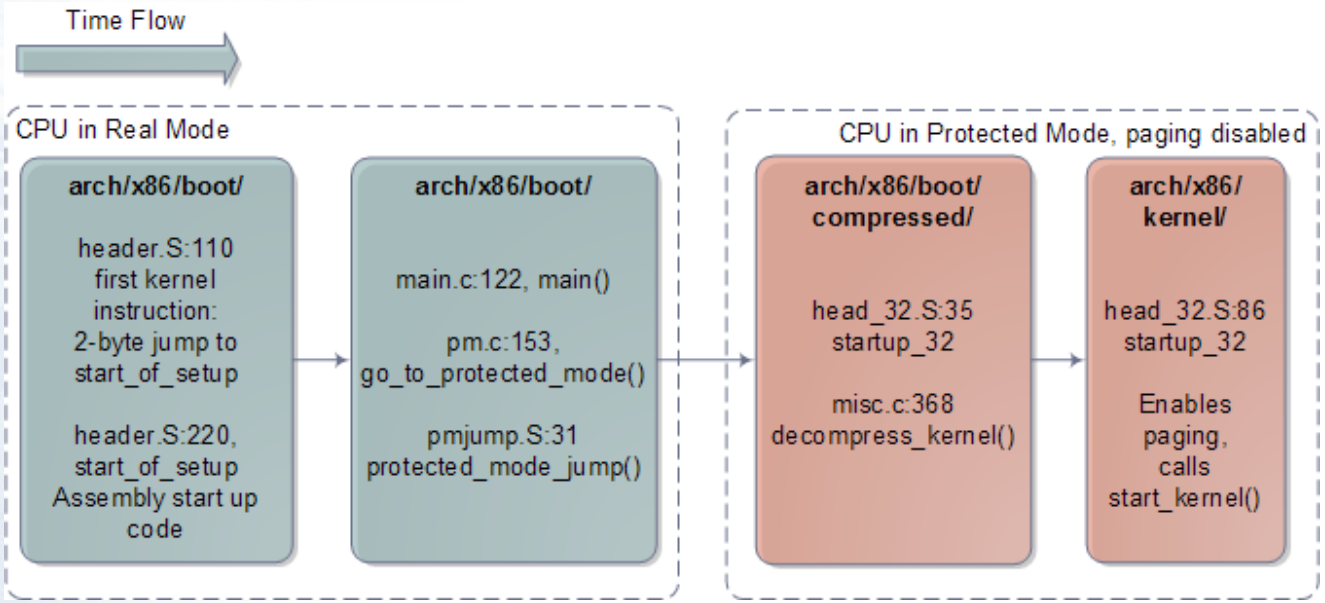
Kernel Source Tree

- Github linux kernel source tree :
  - https://github.com/torvalds/linux
- Bootlin kernel source tree :
  - https://elixir.bootlin.com/linux/latest/source

Kernel Type Mode

- Real Mode
- Protected Mode
- Long Mode

Architecture Linux Kernel Initialization



Kernel Boot Paging (Virtual Memory)

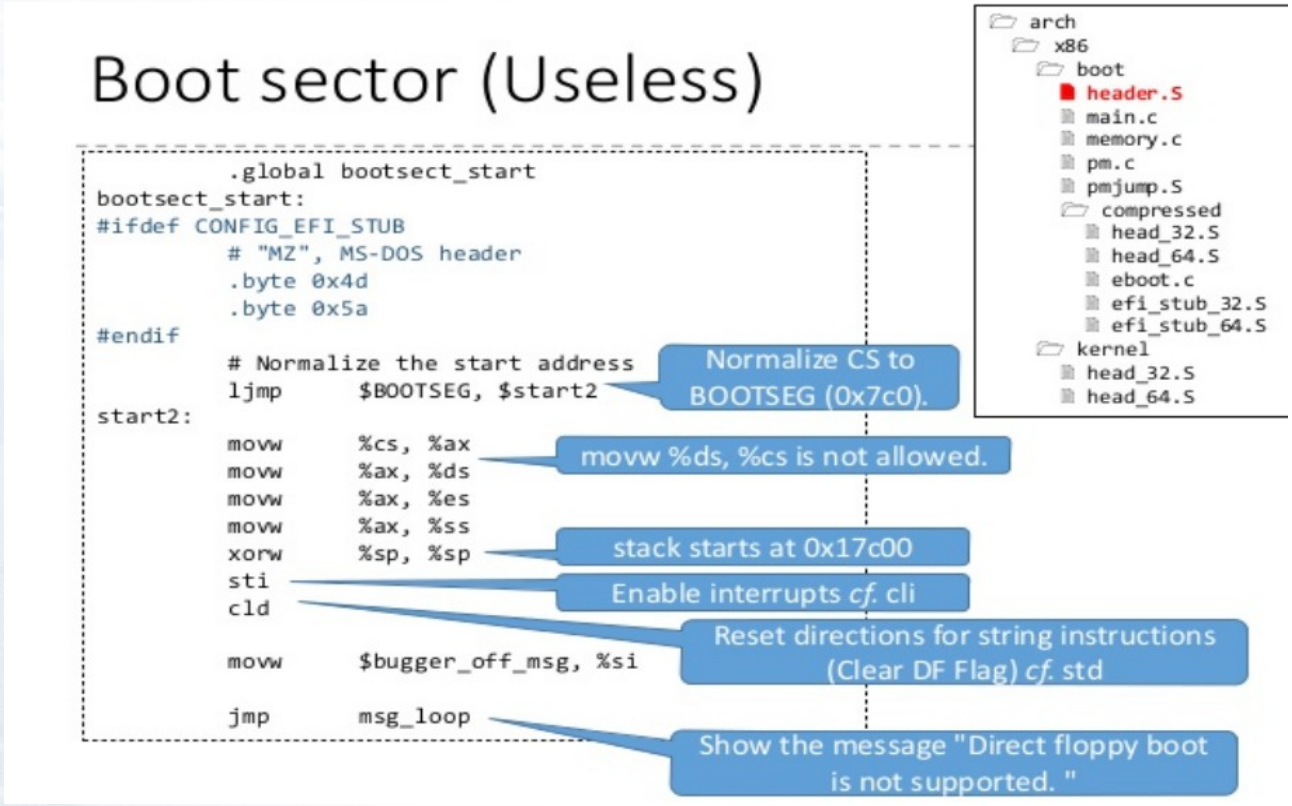
Before the kernel starts...

- x86 (32-bit)
  - Paging is disabled
  - kernel/head_32.S creates a page table and turns on paging
- x86 (64-bit)
  - compressed/head_64.S creates an identical (virtual = physical) page table for the first 4G
  - Long mode requires paging enabled.
  - kernel/head_64.S creates better page table
- ARM
  - kernel/head.S creates a page table and turns on paging

Real Mode Kernel

- header.S
  - Boot sector code which is no longer used
  - Contains setup_header
  - Prepares stack and BSS to run C programs
  - Jumps into the C program (main.c)
- main.c
  - Copies setup_header into "*zeropage*"
  - Setups early console
  - Initializes heap
  - Checks the CPUs (64-bit capable for 64-bit kernel?)
  - Collect HW information by querying to BIOS, and stores the results in "*zeropage*"
  - Finally transits to protected-mode, and jumps into the "protected-mode kernel"

Real Mode Boot sector (header.S)



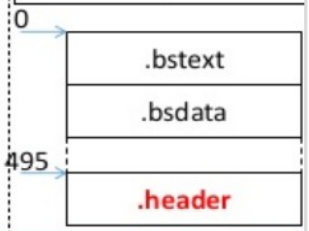
Kernel Real Mode (setup_header)

- header.S
  - Boot sector code which is no longer used
  - Contains setup_header
  - Prepares stack and BSS to run C programs
  - Jumps into the C program (main.c)
- main.c
  - Copies setup_header into "*zeropage*"
  - Setups early console
  - Initializes heap
  - Checks the CPUs (64-bit capable for 64-bit kernel?)
  - Collect HW information by querying to BIOS, and stores the results in "*zeropage*"
  - Finally transits to protected-mode, and jumps into the "protected-mode kernel"

Kernel Real Mode (setup_header)

```
        .section ".header", "a"
        .globl  sentinel
sentinel:.byte 0xff, 0xff  /* Used to detect broken loaders */

        .globl  hdr
hdr:
setup_sects:    .byte 0             /* Filled in by build.c */
root_flags:     .word ROOT_RDONLY
syssize:        .long 0             /* Filled in by build.c */
ram_size:       .word 0             /* Obsolete */
vid_mode:       .word SVGA_MODE
root_dev:       .word 0             /* Filled in by build.c */
boot_flag:      .word 0xAA55

        # offset 512, entry point
        .globl  _start
_start:
        .byte   0xeb        # short (2-byte) jump
        .byte   start_of_setup-1f
1:
        .ascii  "HdrS"      # header signature
        .word   0x020d      # header version number (>= 0x0105)
```

```
arch
  x86
    boot
      header.S
      main.c
      memory.c
      pm.c
      pmjump.S
      compressed
        head_32.S
        head_64.S
        eboot.c
        efi_stub_32.S
        efi_stub_64.S
    kernel
      head_32.S
      head_64.S
```

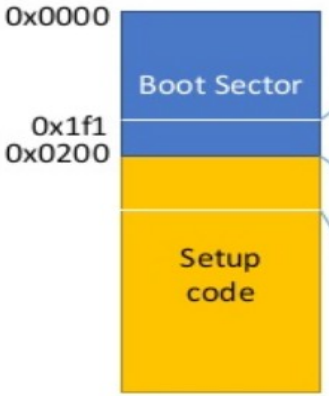


To prevent the compiler from accidentally producing a 3-byte jump

Struct set_header (header.S)

Setup_header

- ".header" section starts at 495
  - 2-byte sentinel is located at the beginning.
  - Struct setup_header begins at 497 (=0x1f1)




```
47:     struct setup_header {
48:             __u8    setup_sects;
49:             __u16   root_flags;
50:             __u32   syssize;
51:             __u16   ram_size;
52:             __u16   vid_mode;
53:             __u16   root_dev;
54:             __u16   boot_flag;
55:             __u16   jump;
56:             __u32   header;
57:             __u16   version;
58:             __u32   realmode_swtch;
... (arch/x86/include/uapi/asm/bootparam.h)
```

Kernel Real Mode Stack (header.S)

Get prepared to C (stack)

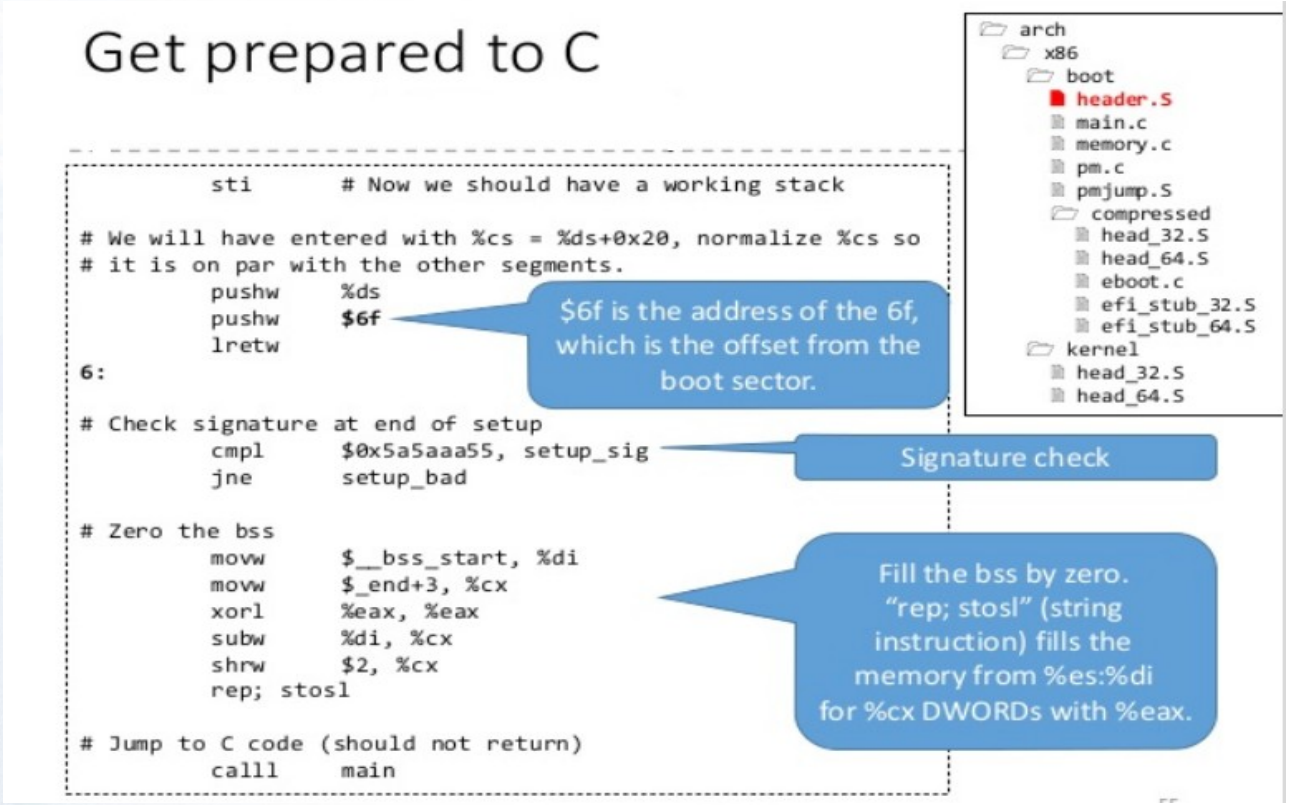
```
        .section ".entrytext", "ax"
start_of_setup:
# Force %es = %ds
        movw    %ds, %ax
        movw    %ax, %es
        cld
        movw    %ss, %dx
        cmpw    %ax, %dx # %ds == %ss?
        movw    %sp, %dx
        je      2f      # -> assume %sp is reasonably set

        # Invalid %ss, make up a new stack
        movw    $_end, %dx
        testb   $CAN_USE_HEAP, loadflags
        jz      1f
        movw    heap_end_ptr, %dx
1:      addw    $STACK_SIZE, %dx
        jnc     2f
        xorw    %dx, %dx # Prevent wraparound
2:      # Now %dx should point to the end of our stack
        andw    $~3, %dx # dword align (might as well
        jnz     3f
        movw    $0xfffc, %dx    # Make sure we're not zero
3:      movw    %ax, %ss
        movzwl  %dx, %esp# Clear upper half of %esp
```

```
arch
  x86
    boot
      header.S
      main.c
      memory.c
      pm.c
      pmjump.S
      compressed
        head_32.S
        head_64.S
        eboot.c
        efi_stub_32.S
        efi_stub_64.S
    kernel
      head_32.S
      head_64.S
```

If %ds == %ss, %sp is assumed to be properly set by the loader

If not, sets up a new stack. The address is _end + STACK_SIZE (512 byte) or heap_end_ptr + STACK_SIZE (if CAN_USE_HEAP is set)

Kernel Real Mode to C (header.S -> main.c)



Kernel Real Mode Main.c

```
void main(void)
{
        /* First, copy the boot header into the "zeropage" */
        copy_boot_params();

        /* Initialize the early-boot console */
        console_init();
        if (cmdline_find_option_bool("debug"))
                puts("early console in setup code\n");

        /* End of heap check */
        init_heap();

        /* Make sure we have all the proper CPU support */
        if (validate_cpu()) {
                puts("Unable to boot - please use a kernel appropriate "
                     "for your CPU.\n");
                die();
        }

        /* Tell the BIOS what CPU mode we intend to run in. */
        set_bios_mode();

        /* Detect memory layout */
        detect_memory();

        /* Set keyboard repeat rate (why?) and query the lock flags */
        keyboard_init();

        /* Query Intel SpeedStep (IST) information */
        query_ist();

        /* Query APM information */
#if defined(CONFIG_APM) || defined(CONFIG_APM_MODULE)
        query_apm_bios();
#endif
```

Copy header to zeropage

Secure | https://elixir.bootlin.com/linux/v4.19-rc2/source/arch/x86/boot/main.c#L30

/ arch / x86 / boot / main.c

```
24  /*
25   * Copy the header into the boot parameter block.  Since this
26   * screws up the old-style command line protocol, adjust by
27   * filling in the new-style command line pointer instead.
28   */
29
30  static void copy_boot_params(void)
31  {
32          struct old_cmdline {
33                  u16 cl_magic;
34                  u16 cl_offset;
35          };
36          const struct old_cmdline * const oldcmd =
37                  (const struct old_cmdline *)OLD_CL_ADDRESS;
38
39          BUILD_BUG_ON(sizeof boot_params != 4096);
40          memcpy(&boot_params.hdr, &hdr, sizeof hdr);
41
42          if (!boot_params.hdr.cmd_line_ptr &&
43              oldcmd->cl_magic == OLD_CL_MAGIC) {
44                  /* Old-style command line protocol. */
45                  u16 cmdline_seg;
46
47                  /* Figure out if the command line falls in the region
48                     of memory that an old kernel would have copied up
49                     to 0x90000... */
50                  if (oldcmd->cl_offset < boot_params.hdr.setup_move_size)
51                          cmdline_seg = ds();
52                  else
53                          cmdline_seg = 0x9000;
54
55                  boot_params.hdr.cmd_line_ptr =
56                          (cmdline_seg << 4) + oldcmd->cl_offset;
57          }
58  }
59
```

Start_kernel Initialization