Linux: the first second

Alison Chaiken
alison@she-devel.com
March 3, 2017
How Linux starts

- What precisely does “off” mean?
- Why does my bootloader have two binaries?
- ACPI vs DTB
- When does the kernel first spawn a second thread?
- When does the kernel bring up a second core?
- How does PID 1 start?
- Out-of-bounds: systemd; SecureBoot; fastboot
THE FOLLOWING CONTENT MAY BE TOO DISTURBING FOR SOME VIEWERS

-- you will witness a boot failure;

-- a NULL pointer will be dereferenced
   … successfully!
Applying power
# Warm vs. power-on reset

<table>
<thead>
<tr>
<th></th>
<th>Clears memory?</th>
<th>Restarts clocks?</th>
<th>Pros</th>
<th>Cons</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power-on Reset</td>
<td>Yes, then reads</td>
<td></td>
<td>Won't fail.</td>
<td>Slightly slower.</td>
<td>Plug-in device</td>
</tr>
<tr>
<td></td>
<td>boot-mode pins.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warm Reset</td>
<td>DDR set to 'self-</td>
<td>reset clocks and</td>
<td>Faster; retains</td>
<td>Can fail.</td>
<td>'reboot'; watchdog;</td>
</tr>
<tr>
<td></td>
<td>refresh', then</td>
<td>jump to stored</td>
<td>'reset reason' and</td>
<td></td>
<td>JTAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>address.</td>
<td>RAM data.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
x86_64: Never genuinely off

Source: Intel

Figure 1. Example using Intel® Active Management Technology in a retail operation to monitor a network of embedded systems even while the enabled systems are powered off.

Similar to Integrated Platform Management Interface, but for 'personal' systems.
ME: High-level overview

Credit: Intel 2009

Source: https://recon.cx/2014/slides/Recon%202014%20Skochinsky.pdf
Bootloaders:
x86 and u-boot
Bootloaders according to Intel

Start ➔ Early Init ➔ Advanced Init ➔ Runtime

- Memory configuration
- Configure stack, jump to advanced initialization

- Configure GPIOs
- Advanced CPU initialization
- Advanced Cache initialization

- Start system timers
- Initialize SMRAM
- Configure PCI resources

- Initialize USB
- Initialize memory map
- Initialize legacy services -optional

- Initialize SATA -optional
- Find and initialize video OPRROM -optional
- Find and initialize expansion ROMS

- Configure serial console -optional
- Shadow ROM to RAM
- Initialize DMA and PIT
- Initialize kbd/ mouse -optional
- Initialize ACPI tables
- Boot to OS or RTOS
Advanced Configuration and Power Interface

Source: Intel

Entertainment: 'sudo acpidump | grep Windows'
Getting more detailed u-boot messages

U-boot config:
# Boot timing
CONFIG_BOOTSTAGE=y
CONFIG_BOOTSTAGE_REPORT=y
CONFIG_BOOTSTAGE_FDT=y

/* Exiting U-Boot, entering OS */
Interpretation of **bootstages**

```c
enum bootstage_id {
    BOOTSTAGE_ID_START = 0,
    BOOTSTAGE_ID_CHECK_MAGIC, /* Checking image magic */
    BOOTSTAGE_ID_CHECK_HEADER,  /* Checking image header */
    BOOTSTAGE_ID_CHECK_CHECKSUM,  /* Checking image checksum */
    BOOTSTAGE_ID_CHECK_ARCH, /* Checking architecture */
    BOOTSTAGE_ID_CHECK_IMAGETYPE = 5, /* Checking image type */
    BOOTSTAGE_ID_DECOMP_IMAGE,  /* Decompressing image */
    BOOTSTAGE_ID_KERNEL_LOADED,  /* Kernel has been loaded */
    BOOTSTAGE_ID_DECOMP_UNIMPL = 7,  /* Odd decompression algorithm */
    BOOTSTAGE_ID_RUN_OS   = 15,  /* Exiting U-Boot, entering OS */

    /* Boot stages related to loading a kernel from an network device */
    BOOTSTAGE_ID_NET_CHECKSUM = 60,
    BOOTSTAGE_ID_NET_ETH_START = 64,
    BOOTSTAGE_ID_NET_ETH_INIT,

    BOOTSTAGE_ID_NET_START = 80,
    BOOTSTAGE_ID_NET_NETLOOP_OK,
};
```
But wait! What are 'shim' bootloaders for?

• AKA “SPL”, “XLoader” or “MLO”.
• “Software program loader” separates CPU-specific code.
• **Problem**: DRAM not ready: controller must be initialized.
• **Solution**: load into SRAM ('OCRAM' in i.MX6, 'l2ram' for TI).
  – Why this works: SRAM (and pNOR) are mapped memory.
• **Problem**: SRAM is little! (256K on i.MX6, 2 MB on DRA7x).
• **Solution**: start with a tiny SPL.
Passing info from Kernel to Bootloader

- U-boot can pass info in registers or in kernel cmdline.
- Most arches pass “Reset cause” to bootloader.
- `mtdoops` and `ramoops` save backtraces in persistent stores specified by `device-tree`.
mtdoops in ARMv7 Device-tree

flash: m25p80@0 {
    compatible = "sst,sst25vf016b", "jedec,spi-nor";
    spi-max-frequency = <0x1312d00>;
    reg = <0x0>;
    address-cells = <0x1>;
    size-cells = <0x1>;

    mtd@00000000 {
        label = "u-boot.img";
        reg = <0x0 0xc0000>;
    }

    mtd@000c0000 {
        label = "u-boot.env";
        reg = <0xc0000 0x2000>;
    }

    mtd@000c2000 {
        label = "splash";
        reg = <0xc2000 0x4000>;
    }

    mtd@000c6000 {
        label = "mtdoops";
        reg = <0xc6000 0x13a000>;
    }
};
The coming revolution in non-volatile storage

Source: Micron

Specs: ArsTechnica

AKA, 'Optane' by Intel

- 1 Gb non-volatile memory → suspend even for brief inactivity.
- POR will become a rare event.
- Profound security implications.
- Linux drivers: Matthew Wilcox, XIP → DAX
- Intel Optane Performance: Non-volatile storage medium in the PCIe M.2 format for significant improvements in endurance, performance, and power consumption.
Starting up the kernel
The kernel is an ELF binary

- vmlinux is a regular ELF binary.
  - `readelf -e vmlinux`
- Extract vmlinux from vmlinuz:
  - `<path-to-kernel-source>/scripts/extract-vmlinux /boot/vmlinuz-$\{uname -r\} > vmlinux`
Quiz:
How do ELF binaries start?
Examining ELF binary start with GDB
(results depend on toolchain)

• Compile your C program with '-ggdb'.
• `gdb <some-binary-executable>`
• Type 'info files'
• Look for 'Entry point'.

• x86_64:
  - Type 'b *(hex address)'
  - Type 'run'
  - Type 'info functions'

• ARM:
  - Type 'l *(hex address)'
  - Type 'l 1,80'
  - Type 'info functions' or 'info sources'
Symbols from "/home/alison/gitsrc/drm-tests-0.1/drm_info_arm".  
Local exec file: 
`/home/alison/gitsrc/drm-tests-0.1/drm_info_arm', file type elf32-little
Entry point: 0x893c
0x00008134 - 0x0000814d is .interp
0x00008150 - 0x00008170 is .note.ABI-tag
0x00008170 - 0x00008194 is .note.gnu.build-id
0x00008194 - 0x00008284 is .gnu.hash
0x00008284 - 0x000084c4 is .dynsym
0x000084c4 - 0x000086ea is .dynstr
0x000086ea - 0x00008732 is .gnu.version
0x00008734 - 0x00008754 is .gnu.version_r
0x00008754 - 0x00008764 is .rel.dyn
0x00008764 - 0x00008814 is .rel.plt
0x00008814 - 0x00008820 is .init
0x00008820 - 0x0000893c is .plt
0x0000893c - 0x00009660 is .text
0x00009660 - 0x00009668 is .fini
0x00009668 - 0x00009b48 is .rodata
0x00009b48 - 0x00009b50 is .ARM.exidx
0x00009b50 - 0x00009b54 is .eh_frame
0x00011b54 - 0x00011b58 is .init_array
0x00011b58 - 0x00011b5c is .fini_array
0x00011b5c - 0x00011b60 is .jcr
0x00011b60 - 0x00011c60 is .dynamic
0x00011c60 - 0x00011cc8 is .got
0x00011cc8 - 0x00011f58 is .data
0x00011f58 - 0x00011f60 is .bss

(gdb) l *(0x893c)
0x893c is at ../ports/sysdeps/arm/start.S:79.
  .type _start,#function
  _start:
    /* Protect against unhandled exceptions. */
  .fnstart
    /* Clear the frame pointer and link register since this is the
     frame. */
The kernel as PID 0

- Userspace processes need a stack, a heap, STD* file descriptors and an environment to start.
- An ASM constructor “ctor” from crti.o, crtn.o and crt1.0 provided by libgcc allocates these resources.
- Source is start.S.
- Corresponding kernel resources are provided via inline ASM.
Examining kernel start with GDB

1. Type 'file vmlinux'. (If zImage, extract with linux/scripts/extract-vmlinux).

2. Type:
   
gdb vmlinux

3. Type:
   
info files

4. Find 'Entry point'.

5. Type:
   
   l *(hex address)

6. Type
   
   l 1,80
Kernel starts in head.S, not start.S.
What's in head.S?

- Type 'file vmlinux.o'
- Try 'arm-linux-gnueabihf-gdb vmlinux.o'
- Type 'info files'
- Type 'l *(0x0)'  <---- actually works!

```
(gdb) l *(0x0)
0x0 is at arch/arm/kernel/head.S:518.
warning: Source file is more recent than executable.
513 mov r4, #0x41000000
514 orr r4, r4, #0x00000b000
515 orr r4, r4, #0x00000020 @ val 0x4100b020
516 teq r3, r4 @ ARM 11MPCore?
517 reteq lr @ yes, assume SMP
518
519 mrc p15, 0, r0, c0, c0, 5 @ read MPIIDR
520 and r0, r0, #0xc0000000 @ multiprocessing extensions and
521 teq r0, #0x80000000 @ not part of a uniprocessor system?
522 bne __fixup_smp_on_up @ no, assume UP
```
The kernel's main() function: 
highlights of start_kernel()

start_kernel() {
− boot_cpu_init();
− page_address_init();
− setup_arch(&command_line);
− page_alloc_init();
− pr_notice("Kernel command line: ");
− parse_args("Booting kernel", command_line);
− mm_init();
− sched_init();
− init_IRQ();
− init_timers(); timekeeping_init();
− security_init();
}
About Initrd
What is an initrd anyway?

- 'init ramdisk' = filesystem that is loaded into memory by the kernel before the rootfs mounts.
- Why?
  - To provide a 'rescue shell' in case rootfs doesn't mount.
  - To provide modules that don't fit in zImage.
  - To provide a safe environment to run aggressive tests.
  - To facilitate software updates on devices with limited storage.
What's in an initrd and why?

• Boot into the rescue shell by providing a broken cmdline in /boot/grub/grub.cfg
  – Type 'ls'
• Or try 'lsinitramfs /boot/$(uname -r)'
• initrd is a gzipped cpio archive:
  
  cp /boot/initrd-$(uname -r) /tmp/initrd.gz
  gunzip /tmp/initrd.gz
  cpio -t < /tmp/initrd
Exploring initramfs

```
(initramfs) ls
bin  dev  init  lib64  root  sbin  sys  var
conf  etc  lib  proc  run  scripts  tmp

(initramfs) mount
rootfs on / type rootfs (rw)
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,relatime,size=10240k,nr_inodes=1524441,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,relatime,size=2442500k,mode=755)
```

(initramfs) df -h

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Available</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>udev</td>
<td>10.0M</td>
<td>0</td>
<td>10.0M</td>
<td>0%</td>
<td>/dev</td>
</tr>
<tr>
<td>tmpfs</td>
<td>2.3G</td>
<td>72.0K</td>
<td>2.3G</td>
<td>0%</td>
<td>/run</td>
</tr>
</tbody>
</table>
Booting into Rescue Shell

```
Begin: Running /scripts/local-block ... done.
Begin: Running /scripts/local-block ... done.
Begin: Running /scripts/local-block ... done.
Begin: Running /scripts/local-block ... done.

Gave up waiting for root device. Common problems:
- Boot args (cat /proc/cmdline)
  - Check rootdelay= (did the system wait long enough?)
  - Check root= (did the system wait for the right device?)
- Missing modules (cat /proc/modules; ls /dev)

ALERT! /dev/disk/by-uuid/123-seems-unlikely does not exist. Dropping to a shell!

modprobe: module ehci-orion not found in modules.dep
[ 31.960748] uhci_hcd: USB Universal Host Controller Interface driver
[ 31.971962] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[ 31.975625] hidraw: raw HID events driver (C) Jiri Kosina
[ 31.976950] usbcore: registered new interface driver usbhid
[ 31.977035] usbhid: USB HID core driver

BusyBox v1.22.1 (Debian 1:1.22.0-9+deb8u1) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/bin/sh: can't access tty; job control turned off
(initramfs)
```
OMG! My life is over! (rescue shell tips)

Inhale on a 4-count, then exhale on a 10-count.

- Oh no! 'help' scrolls pages of unreadable crap!
  Relax your jaw. Make circles with your neck.
- Read 'man busybox'.
- 'help | grep' works in busybox.
- Look in /bin and /sbin. There's fsck!!
- You have sed and vi (but not emacs ;-(- )
- Type 'reboot -f' when you are bored.
Hibernation and Suspension
{Hibernate, Suspend} == Freeze

- **Suspend**: save state to memory.
- **Hibernate**: save state to disk (swap).
- Main code is in `kernel/freezer.c`.
- Freezing of processes preserves state, protects filesystems.
- Each core must save its state separately.
- To watch, append `no_console_suspend` to bootargs.
How to suspend from CLI

`no_console_suspend` in bootargs gives debug output.
How to Hibernate from CLI

Only works for devices with swap space.
Summary

- u-boot's CONFIG_BOOTSTAGES, acpidump and systemd-bootchart provide details about boot.

- Practicing with QEMU is entertaining and low-risk.

- Examine your kernel with GDB and binutils.

- ARM and x86 boot processes are quite distinct.

- High-speed non-volatile memory is about to have a massive impact on Linux.
Major References

- *Embedded Linux Primer* by Chris Hallinan (book)
- *Booting ARM Linux* by Russell King and *THE LINUX/x86 BOOT PROTOCOL* (Documentation/)
- Program startup process in userspace at linux-insides blog
- Matthew Garrett's comprehensive series on UEFI
- Status of Intel Management Engine on various laptops (Coreboot)
- All about ACPI talk by Darren Hart, ELCE 2013
- Arch Wiki on hacking ACPI tables
do_bootm_states = u-boot state machine

Main Entry point for arm bootm implementation

BootROM (internal EEPROM)

resume?

Yes

Read Reset Controller registers
Jump to stored image

No

Das U-boot

“Main Entry point for arm bootm implementation”

bootm.c<common>

***

bootm_start() → bootm_find_os()

bootm_load_os() → bootm_os_get_boot_func()

boot_selected_os

bootm.c<lib>

do_bootm_linux()

bootm_jump_linux()
Boot ROM in CPU

Shim bootloader (Xloader, MLO, etc.)

bootloader (u-boot, GRUB, etc.)

head.S
“Kernel startup entry point”

Decompress the kernel

start of zImage

kernel/smp.c

secondary_start_kernel()

kernel/primaryboot.c

idle_threads_init()

main.c

start_kernel()

do_initcalls()

kernel_init

udev

init

cpu_idle

start userspace

spawn 2nd thread

Boot secondary cores

start of zImage

kernel/smp.c

smp_init()
How to create your own initrd

- Unpack one that already works with gunzip and 'cpio -i'
- Copy in your binary.
- Use gen_initramfs.h from kernel source tree:
  - scripts/gen_initramfs_list.sh -o <archive> <path to source>
- Run 'lsinitramfs <archive>}' to check the result.
- cp <archive> /boot; edit /boot/grub/grub.cfg
  
  CAUTION: your system boots fine, right? You're crazy to mess with the bootloader, you moron.
- Run grub-script-check.
The magnificent result!
Getting more detailed kernel messages at boot

• Remove 'quiet' from the kernel command line.
• How to keep 'quiet' from coming back:
  – edit /etc/grub.d/10_linux and add:
    export GRUB_DISABLE_SUBMENU=y
    export GRUB_CMDLINE_LINUX_DEFAULT=""

CAUTION: your system boots fine, right? You're crazy to mess with the bootloader, you moron.
• Always run 'grub-script-check /boot/grub/grub.cfg' afterwards.
Learning more with systemd-bootchart

- Make sure kernel is compiled with CONFIG_SCHEDSTATS=y.
- 'apt-get install systemd-bootchart'
- Interrupt grub by typing 'e'
- Append 'init=/lib/systemd/systemd-bootchart' to the line that starts with 'linux'
- After boot, open the SVG image in /run/log/ with a browser.
A change in compiling your own kernel

To: 823107-done@bugs.debian.org
Subject: Re: Bug#823107: linux: make deb-pkg fails: No rule to make target 'debian/certs/benh@debian.org.cert.pem'
From: Ben Hutchings <benh@decadent.org.uk>
Date: Sat, 30 Apr 2016 22:50:04 +0200

Closing, this is not a bug.

You wrote:
[...]
> Should I remove CONFIG_SYSTEM_TRUSTED_KEYS from .config before building
> the kernel? I hope not.
[...]

Yes, you must do that. Your custom kernel configuration should be based on the appropriate file provided in linux-source-4.5. These have the CONFIG_MODULE_SIG_ALL, CONFIG_MODULE_SIG_KEY and CONFIG_SYSTEM_TRUSTED_KEYS settings removed so that custom kernels will get modules signed by a one-time key.

Ben.
Appendix: running QEMU

#!/bin/bash
ROOTDIR=/home/alison/ISOs
HDNAME=debian-testing
VERSION=4.9.5

# Load kernel via GRUB; console shows in QEMU window.
#qemu-system-x86_64 -machine accel=kvm -name ${HDNAME} -boot c -drive file=
{ROOTDIR}/${HDNAME}.raw,format=raw -m 4096 -smp cpus=1 -net nic,model=e1000
-net user,hostfwd=tcp:127.0.0.1:6666-:22 -localtime -serial stdio

# Load kernel from external file; console shows in xterm; GRUB doesn't run.
qemu-system-x86_64 -machine accel=kvm -name ${HDNAME} -initrd
/home/alison/embedded/SCALE2017/kernel/initrd.img-${VERSION} -kernel
/home/alison/embedded/SCALE2017/kernel/vmlinux-${VERSION} -boot c -drive file=$
{ROOTDIR}/${HDNAME}.raw,format=raw -m 4096 -smp cpus=1 -net nic,model=e1000
-net user,hostfwd=tcp:127.0.0.1:6666-:22 -localtime -serial stdio -append
"console=ttyAMA0  console=ttyS0 root=UUID=8e6a1c7e-b3c4-4a37-8e21-56a137c9dded
ro"
[alison@hildesheim u-boot-imx6 (boundary-v2016.03)]$ file u-boot
u-boot: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /usr/lib/ld.so.1, not stripped
[alison@hildesheim u-boot-imx6 (boundary-v2016.03)]$ arm-linux-gnueabihf-gdb u-boot

(gdb) info files
Symbols from "/home/alison/gitsrc/u-boot-imx6/u-boot".
Local exec file: 
Entry point: 0x17800000
0x17800000 - 0x17852864 is .text
0x17852868 - 0x1786646e is .rodata
0x17866470 - 0x1786649c is .hash
0x178664a0 - 0x1786b25c is .data
0x1786b25c - 0x1786b268 is .got.plt
0x1786b268 - 0x1786bdd0 is .u_boot_list
0x17877a30 - 0x17877a90 is .dynsym
0x17877a90 - 0x17877a30 is .rel.dyn
0x17877a90 - 0x1787b7fd8 is .bss
0x17877a90 - 0x17877aba is .dynstr
0x17877a3c - 0x17877b3c is .dynamic
0x17877b3c - 0x17877b4d is .interp

(gdb) l *(0x17800000)
0x17800000 is at arch/arm/lib/vectors.S:54.
49
50 #ifdef CONFIG_SYS_DV_NOR_BOOT_CFG
51 .word CONFIG_SYS_DV_NOR_BOOT_CFG
52 #endif
53
54 b reset
55 ldr pc, _undefined_instruction
56 ldr pc, _software_interrupt
57 ldr pc, _prefetch_abort
58 ldr pc, _data_abort
The ARM bootloader

- Read fundamental configuration from fuses, switches and GPIOs.
- Then, for ARM:
  1. Setup and initialise the RAM.
  2. Initialise one serial port.
  3. Detect the machine type.
  4. Setup the kernel tagged list. device-tree
  5. Load initramfs.
  6. Call the kernel image.

Code in the SPL: board_init_f() and jump_to_image_linux()
Where do messages originate?

[ 54.590327] Starting kernel ...
[ 54.593459]
Uncompressing Linux... done, booting the kernel.

Linux version 3.0.35-2508-g54750ff (gcc version 4.6.3 #1 SMP PREEMPT
CPU: ARMv7 Processor [412fc09a] revision 10 (ARMv7), cr=10c53c7d
CPU: VIPT nonaliasing data cache, VIPT aliasing instruction cache

Machine: Freescale i.MX 6Quad/DualLite/Solo Sabre-SD Board
Memory policy: ECC disabled, Data cache writealloc
CPU identified as i.MX6Q, silicon rev 1.1
PERCPU: Embedded 7 pages/cpu @8c008000 s5440 r8192 d15040 u32768
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 227328

Kernel command line: console=ttymxc1,115200 ip= dhcp rootwait root=/dev/nfs
nfsroot=172.17.0.1:/tftpboot/alison/mx6q/fsl-mx6,v3,tcp

passed from u-boot
Image, zImage, uImage, vmlinux, vmlinuz?

- *Image* is the raw executable.
- *zImage* is compressed version of Image with prepended uncompresion instructions in ASM.
- *uImage* is a *zImage* with a u-boot header.
- *vmlinux* is ELF executable containing *Image* in .text section.
- *vmlinuz* is a stripped version of vmlinux.