Linux: the first second

Alison Chaiken
alison@she-devel.com
January 25, 2018

All code for demos

Related blog post at opensource.com
Panic Concern ensues among automakers shipping Linux.
Slow boot: **Linux boot on 8-bit AVR**

“uARM is certainly no speed demon. It takes about 2 hours to boot to bash prompt”.

**System:**

8-bit micro,
external storage,
external RAM,
32-bit ARMv5 emulation.
How Linux starts

- What precisely does “off” mean?
- Fun with bootloaders
- ACPI vs DTB
- The kernel as PID 0
- How does PID 1 start?
- What is an initrd?
Applying power

[Image of power switches]

https://www.flickr.com/photos/cauromartins/1313958855/in/album-72157647926238117/
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.

IPMI: run from Baseboard Management Controller
AMT: run from Platform Controller Hub
Platform Initialization (PI) Boot Phases

OS-present App, a.k.a. exploit home

GRUB (x86) or u-boot (ARM)

Source: Minnich et al., ELCE2017
Purism, System76, Dell turn AMT off

Dell Now Shipping Laptops With Intel’s Management Engine Disabled

By Joel Hruska on December 4, 2017 at 4:10 pm | 9 Comments

Source: ExtremeTech, December 2017
ARM Bootloader:
u-boot
Fun with u-boot's sandbox
(demo placeholder)

- How-to:
  
  make ARCH=sandbox defconfig
  make
  ./u-boot
- Even more fun:
  
  make_test_disk.sh
  file test.raw; gdisk -l test.raw
  ./u-boot
  host bind 0 test.raw
  printenv
  gpt read host 0
  fatls host 0:1
  fdt addr $fdt_addr_
  fdt header
```bash
$\$
```
```
# uname -m
x86_64
```
```
# pwd
/home/alison/gitsrc/u-boot
```
```
# ./u-boot
```
```
```
```
U-Boot 2017.11-00060-g6b18e4693c (Nov 19 2017 - 13:06:48 -0800)
```
```
DRAM: 128 MiB
```
```
MMC:
```
```
Using default environment
demo placeholder
```
```
In: serial
```
```
Out: serial
```
```
Err: serial
```
```
SCSI: Net: No ethernet found.
```
```
IDE: Bus 0: not available
```
```
Hit any key to stop autoboot: 0
```
```
reading bzImage
```
```
FAT: Misaligned buffer address (00007ff5aff71008)
```
```
7972624 bytes read in 16 ms (475.2 MiB/s)
```
```
setting up X86 zImage [ 0 - 7972624 ]
```
```
## Transferring control to Linux (at address 0000000000000000)
```
```
sandbox: continuing, as we cannot run Linux
```
```
=>
```
How the system reaches the kernel initialization stage
Kernel's “address book”: ACPI or Device-tree

- ACPI tables in SPI-NOR flash.
- At boot:
  'dmesg | grep DT'
- Examine:
  'acpidump | grep Windows'
- Get source: run iasl to extract
- Modify: boot-time 'BIOS' menu.

- device-tree in /boot.
- At boot:
  each driver reads the DTB.
- Examine:
  'strings /boot/foo.dtb'
- Get source: from kernel
- Modify: edit source, run dtc, copy to /boot.
Starting up the kernel
The kernel is an ELF binary

- Extract vmlinux from vmlinuz:
  - `<path-to-kernel-source>/scripts/extract-vmlinux \` /boot/vmlinux-$(uname -r) > vmlinux

- vmlinux is a regular ELF binary:
  - file vmlinux; file /bin/ls
  - readelf -e vmlinux; readelf -e /bin/ls
Quiz:
How do ELF binaries start?
Quiz:
Where do argc and argv come from?
Inspecting the start of ls with GDB

[alison@hildesheim coreutils-8.28]$ gdb src/ls
Reading symbols from src/ls...done.
(gdb) b __init
Breakpoint 1 at 0x3338
(gdb) run
Starting program: /home/alison/embedded/LCA/demos/coreutils-8.28/src/ls

Breakpoint 1, __init (argc=0x1, argv=0x7ffffffffe2e8, envp=0x7ffffffffe2f8)
at ../csu/init-first.c:52
52   {
    (gdb) bt
#0  __init (argc=0x1, argv=0x7ffffffffe2e8, envp=0x7ffffffffe2f8) at ../csu/init-first.c:52
#1  0x00007fffffff7de742a in call_init (l=0x7ffffffdf5000, argc=argc@entry=0x1, argv=argv@entry=0x7ffffffffe2e8, env=env@entry=0x7ffffffffe2f8) at dl-init.c:58
#2  0x00007fffffff7de7576 in call_init (env=0x7ffffffffe2f8, argv=0x7ffffffffe2e8, argc=0x1, l=<optimized out>) at dl-init.c:119
#3  __dl_init (main_map=0x7ffffffffe150, argc=0x1, argv=0x7ffffffffe2e8, env=0x7ffffffffe2f8) at dl-init.c:120
#4  0x00007fffffff7dd8eda in __dl_start_user () from /lib64/ld-linux-x86-64.so.2
#5  0x0000000000000001 in ?? ()
Examining ELF binary start with GDB
(results depend on toolchain and libc)

- Compile your C program with `-ggdb`.
- `gdb <some-binary-executable>`

- set backtrace past-main on
- set backtrace past-entry on
- Type 'run'
- frame 1; list

- Type 'info files'
- Find 'Entry point'.
- Type 'l *(hex address)'
- Type 'l 1,80'
- Type 'info functions' or 'info sources'

demo placeholder
The kernel as PID 0

- **Userspace** processes need to start need:
  - stack,
  - heap,
  - STD* file descriptors
  - environment
- glibc and libgcc allocate these resources.
  - Source is in `start.S` (ARM) and `libc-start.c`.
- Corresponding *kernel* resources provided via inline ASM.
  - Reads cmdline, device-tree or ACPI.
Examining ARM32 kernel start with GDB
(demo placeholder)

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

   arm-linux-gnueabihf-gdb vmlinux
3. Type:

   info files
4. Find 'Entry point'.
5. Type:

   l *(hex address)
6. Type

   l 1,80
What's in ARM's head.S?

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

```
(gdb) l *(0x0),*(0x60)
0x0 is at arch/arm/kernel/head.S:367.
367   bl   __hyp_stub_install_secondary
368   #endif
369   safe_svcmode_maskall r9
370
371   mrc   p15, 0, r9, c0, c0           @ get processor id
372   bl   __lookup_processor_type
373   movs  r10, r5                     @ invalid processor?
374   moveq r0, #'p'                   @ yes, error 'p'
375   THUMB(   it   eq )                @ force fixup-able long branch encoding
376   beq   __error_p
377
378   /*
379   * Use the page tables supplied from __cpu_up.
380   */
381   adr   r4, __secondary_data
382   ldmia  r4, {r5, r7, r12}          @ address to jump to after
383   sub   lr, r4, r5                  @ mmu has been enabled
```
Examining x86_64 kernel with GDB
(demo placeholder)

1. Type 'file vmlinux'. (If zImage, extract with linux/scripts/extract-vmlinux).
2. Type:
   
3. Type:
   
4. Find '.init.text'.
5. Type:
   
   l *(hex address)

6. Type

   l 200,290
What's in x86_64 head_64.S?

(gdb) info files
Symbols from "/home/alison/gitsrc/linux-trees/linux/vmlinux".
Local exec file:
"/home/alison/gitsrc/linux-trees/linux/vmlinux", file type elf64-x86-64.
warning: Cannot find section for the entry point of /home/alison/gitsrc/linux-trees/linux/vmlinux.
    Entry point: 0x1000000
0xffffffff81000000 - 0xffffffff820916eb is .text
0xffffffff820916ec - 0xffffffff820918c0 is .notes
0xffffffff820918c0 - 0xffffffff82093870 is __ex_table
0xffffffff82200000 - 0xffffffff823e5562 is .rodata
0xffffffff823e5568 - 0xffffffff823e9240 is .pci_fixup
0xffffffff823e9240 - 0xffffffff823fa450 is __ksymtab
0xffffffff823fa450 - 0xffffffff82408210 is __ksymtab_gpl
0xffffffff82408210 - 0xffffffff8240c694 is __kcrctab
0xffffffff8240c694 - 0xffffffff8240fe04 is __kcrctab_gpl
0xffffffff8240fe04 - 0xffffffff82435a23 is __ksymtab_strings
0xffffffff82435a40 - 0xffffffff82435af0 is __init_rodata
0xffffffff82435af0 - 0xffffffff82437738 is __param
0xffffffff82437738 - 0xffffffff82438000 is __modver
0xffffffff82600000 - 0xffffffff837c3340 is .data
0xffffffff837c3340 - 0xffffffff837d18e4 is __bug_table
0xffffffff837d2000 - 0xffffffff837d3000 is .vvar
0x0000000000000000 - 0x0000000000000001cdd8 is .data..percpu
0xffffffff8387f000 - 0xffffffff8387c373 is .init.text

(gdb) 1 *(0xffffffff837f0000)
0xffffffff837f0000 is at arch/x86/kernel/head_64.S:287.
282 .endif
283     pushq $i       # 72(%rsp) Vector number
284     jmp early_idt_handler_common
285     i = i + 1
286 .fill early_idt_handler_array + i*EARLY_IDT_HANDLER_SIZE - ., 1, 0xcc
287 .endr
288     ENDPRE(early_idt_handler_array)
289
290     early_idt_handler_common:
291     /*
(gdb) 1 200,290
The kernel's main() function

```c
start_kernel() {
    boot_cpu_init();
    setup_arch(&command_line);
    page_alloc_init();
    pr_notice("Kernel command line: ");
    mm_init();
    sched_init();
    init_IRQ();
    init_timers(); timekeeping_init();
    console_init();
    rest_init();
}
```

"Activate the first processor."

- process the device-tree
- setup page tables and start virtual memory
- All timestamps before are [0.000000]
- All on one core!
Boot ROM in CPU → bootloader (u-boot, GRUB) → head_64.S

“Kernel startup entry point” → Decompress

Decompress start of zImage

main.c

start_kernel()

rest_init()

spawn 2\textsuperscript{nd} thread

kernel_init

cpu_idle

Finish core kernel bringup
Boot ROM in CPU

bootloader (u-boot, GRUB)

head_64.S
"Kernel startup entry point"

Decompress

Start of zImage

arch/arm/kernel/smp.c

smp_init()

kernel/smp.c

secondary_start_kernel()

main.c

start_kernel()

rest_init()

spawn 2nd thread

kernel_init

Bring up symmetric multiprocessing (SMP)

Boot secondary cores

cpu_idle
Kernel boot via **BCC**

x86_64_start_kernel: head_64.S
Boot ROM in CPU

bootloader (u-boot, GRUB)

head_64.S
“Kernel startup entry point”

Decompress

start of zImage

cpu_idle

Kernel startup entry point

Boot secondary cores

main.c

start_kernel()

rest_init()

spawn 2nd thread

kernel_init

do_initcalls()

Bring up devices, filesystems . . .

device drivers

probe devices
Boot ROM in CPU

bootloader (u-boot, GRUB)

head_64.S  "Kernel startup entry point"

Decompress

cpu_idle

Finally, userspace.

kernel/smp.c

smp_init()

cores

arch/arm/kernel/smp.c

secondary_start_kernel()

main.c

start_kernel()

rest_init()

spawn 2\textsuperscript{nd} thread

kernel_init

do_initcalls()

start userspace

init

device drivers
**Boot ROM in CPU**

**Bootloader** (u-boot, GRUB)

**head_64.S**

"Kernel startup entry point"

**Decompress**

**main.c**

**start_kernel()**

**rest_init()**

**kernel_init**

**do_initcalls()**

**cpu_idle**

**Boot secondary cores**

**kernel/smp.c**

**smp_init()**

**arch/arm/kernel/smp.c**

**secondary_start_kernel()**

**start userspace**

**init**

**device drivers**

**probe devices**

**spawn 2nd thread**
Summary

- Practicing with u-boot sandbox is comparatively relaxing.

- Viewing the kernel as ELF helps to understand early boot.

- Several processors and SW components participate in boot.

- Until the scheduler and SMP start, the boot process is relatively simple.
Acknowledgements

- Big thanks to Joel Fernandes and Akkana Peck for suggestions.
- Shout-out to Linaro for making ARM so much easier than x86.
Major References

- *Embedded Linux Primer* by Chris Hallinan and *Essential Linux Device Drivers* by Sreekrishnan Venkateswaran (books)
- *Booting ARM Linux* by Russell King and *THE LINUX/x86 BOOT PROTOCOL*(Documentation/)
- Program startup process in userspace at linux-insides blog, Michael Kerrisk's TLPI (book)
- Matthew Garrett's comprehensive series on UEFI
- Status of Intel Management Engine on various laptops (Coreboot) and servers (FSF)
- Nov, 2017 Intel Management Engine exploits and vulnerability detection tool
- All about ACPI talk by Darren Hart, ELCE 2013, Arch Wiki on hacking ACPI tables
- 'apt-get install debian-kernel-handbook'; GDB docs chapter 8
Cold-boot may become rare

Source: Micron

- Non-volatile RAM $\rightarrow$ suspend even for brief inactivity.
- Minimal diff between 'suspend' and 'hibernate'? 
- Linux drivers: Matthew Wilcox, XIP $\rightarrow$ DAX

Specs: ArsTechnica

AKA, 'Optane' by Intel
About Initrds
Booting into Rescue Shell

```
Begin: Waiting for root file system ... Begin: Running /scripts/local-block
done.

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

ALERT! UUID=maybe-it-will-work does not exist. Dropping to a shell!

BusyBox v1.27.2 (Debian 1:1.27.2-2) built-in shell (ash)
Enter 'help' for a list of built-in commands.

(initramfs) bin/hello_world.sh
Never gonna give you up!
```

(initramfs) -
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.
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>

(initramfs)
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
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' or 'exit' when you are bored.
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/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!
The Lenovo laptop on which the slides were created has known IME vulnerabilities described by unpatched CVEs. This has nothing to do with Meltdown and Spectre.
Bootloaders according to Intel

- Configure serial console (optional)
- Shadow ROM to RAM
- Initialize SATA (optional)
- Find and initialize video OPROM (optional)
- Find and initialize expansion ROMS
- Initialize ACPI tables
- Boot to OS or RTOS

**Start** → **Early Init** → **Advanced Init** → **Runtime**

- Memory configuration
- Configure GPIOs
- Start system timers
- Initialize USB
- Initialize SMM

- Chipset basic initialization
- Configure product specific features
- Upload CPU microcode
- Configure SIO - optional
- Initialize kbd/mouse - optional
- Initialize SMRAM
- Initialize memory mapping
- Initialize services - optional

- Switch to Big Real Mode
- CPU basic initialization
- Configure PIRQ & Int handlers
- Initialize stack, jump to advanced initialization
- Advanced CPU initialization
- Advanced Cache initialization
- Configure PCI resources
- Initialize legacy services (optional)
Coming soon to a system near you

Source: Anandtech
Investigating your laptop's PCH

- Try:
  
  `lsmod | grep pch`

- Try:
  
  `find /lib/modules/$(uname -r)/ -name "*pch*"`

- Then (for example):

```
[alison@hildesheim LCA]$ modinfo pch_udc
filename:   /lib/modules/4.13.0-1-amd64/kernel/drivers/usb/gadget/udc/pch_udc.ko
license:   GPL
author:    LAPIS Semiconductor, <tomoya-linux@dsn.lapis-semi.com>
description: Intel EG20T USB Device Controller
```

**EG20T = Intel Topcliff PCH**
Why bootloaders have two parts

- ARM: “SPL”, “XLoader” or “MLO” in addition to u-boot.img.
- **Problem**: DRAM 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.
## Warm vs. power-on reset

<table>
<thead>
<tr>
<th></th>
<th>Clears memory? 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 boot-mode pins.</td>
<td>Won't fail.</td>
<td>Slightly slower.</td>
<td>Plug-in device</td>
</tr>
<tr>
<td>Warm Reset</td>
<td>DDR set to 'self-refresh', then reset clocks and jump to stored address.</td>
<td>Faster; retains 'reset reason' and RAM data.</td>
<td>Can fail.</td>
<td>'reboot'; watchdog; JTAG</td>
</tr>
</tbody>
</table>
Advanced Configuration and Power Interface

Source: Intel
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

***
bootm_start() → bootm_find_os()

→
bootm_load_os() → bootm_os_get_boot_func()

→
boot_selected_os

bootm.c<common>

bootm.c<lib>

do_bootm_linux()

bootm_jump_linux()

“Main Entry point for arm bootm implementation”

Das U-boot
Where do messages originate?

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
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 <ben@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/vmlinuz-${VERSION} -boot c -drive file=${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-b3c4-4a37-8e21-56a137c9ddef ro"
```
Finding u-boot start with GDB

(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
0x1786bdd0 - 0x17877a30 is .rel.dyn
0x1786bdd0 - 0x178b7fd8 is .bss
0x17877a90 - 0x17877aba is .dynstr
0x17877abc - 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()
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.
Older x86 had ARC processor as GMCH instead of PCH.

Credit: Intel 2009

Source: https://recon.cx/2014/slides/Recon%202014%20Skochinsky.pdf