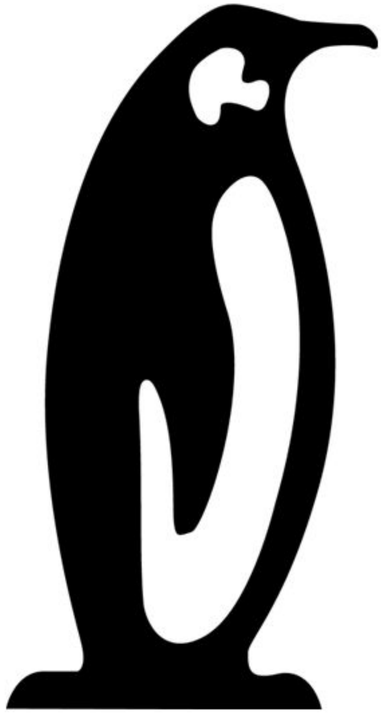

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Android Generic Boot Loader

Linux Plumbers Android Micro Conference

Dmitrii Merkurev <dimorinny@google.com>

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

Android boot is ... complicated

1. Read out the partition table
2. Trigger the boot screen animation or boot splash screen
3. Read BCB (misc) to identify boot mode / proper slot to boot from
4. Load boot, init_boot, vendor_boot, dtb, dtbo, vbmeta, etc
5. Read the kernel (decompress), ramdisks, device trees, and bootconfig out of these images
6. Execute AVB, modify bootconfig with the result hash
7. Interact with the TEE in a SOC specific manner
8. Apply runtime fixups for device tree, command line, bootconfig which may be specific to OEM/SOC
9. Place the kernel, command line, DTB, ramdisks, and bootconfig into RAM
10. Prepare board (flush disk caches, disable MMU, etc)
11. Do a kernel jump
12. Fastboot (including custom OEM commands)

We have something to reuse

1. Read out the partition table
2. Trigger the boot screen animation or boot splash screen
3. Read BCB (misc) to identify boot mode / proper slot to boot from
4. Load boot, init_boot, vendor_boot, dtb, dtbo, vbmeta, etc
5. Read the kernel (decompress), ramdisks, device trees, and bootconfig out of these images
6. Execute AVB, modify bootconfig with the result hash
7. Interact with the TEE in a SOC specific manner
8. Apply runtime fixups for DT, command line, bootconfig
9. Place the kernel, command line, DTB, ramdisks, and bootconfig into RAM
10. Prepare board (flush disk caches, disable MMU, etc)
11. Do a kernel jump
12. Fastboot (including custom OEM commands)

Problems to solve

- Fragmentation of the FW across the ecosystems
  - Android boot is getting changed regularly
    - new partitions
    - binary header structures updates (boot, vendor_boot, etc)
    - command line -> bootconfig
  - Bootloader release cycle == ABL release cycle

- Documentation alone may be not enough. Having reference implementation is useful.

Introduce GBL

Generic bootloader (GBL) is Android boot flow UEFI application provided by Google.
The main value:

For partners, ecosystem:

- Reduce the vendor’s integration burden
- Provide production ready open source Android boot flow reference implementation

For Google:

- Faster uptake of Android Boot changes by partners
- Guaranteed using trusted components across ecosystem (libavb)

GBL is

- no_std Rust UEFI app (dynamic allocations use UEFI)
- Support x86 (both 32/64) / arm64 / riscv64 architectures
- Available as a part of AOSP, so fully opensourced
- Built by BAZEL
- Statically compiled against trusted components (ATF, libavb, libfdt, libufdt)
- Gonna be shipped as a part of dedicated esp _a/_b partition
- Already can be used to boot Cuttlefish

Why UEFI?

- Already adopted by some partners for production devices
- Supported by various firmware (EDK2, U-Boot)
- UEFI interfaces mechanism is a flexible way to implement vendor-specific logic
- Offers a variety of existing standardized interfaces for use such as block devices, network, etc
- The UEFI runtime is stable. Version 2.10 in use for more than 10 years.
- Advocated by ARM's SystemReady initiative

Alternatives considered:

- Coreboot
- u-boot as a reference implementation

We're also interested / looking at:

- efidroid
- no-bootloader

Supported protocols

- EFI_BLOCK_IO_PROTOCOL
- EFI_BLOCK_IO2_PROTOCOL (optional for async I/O)
- EFI_DEVICE_PATH_PROTOCOL
- EFI_DEVICE_PATH_TO_TEXT_PROTOCOL
- EFI_LOADED_IMAGE_PROTOCOL
- EFI_SIMPLE_NETWORK_PROTOCOL
- EFI_SIMPLE_TEXT_INPUT_PROTOCOL
- EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL
- UEFI memory allocation service API
- RISCV_EFI_BOOT_PROTOCOL

Proposed protocols

- GBL_EFI_OS_CONFIGURATION_PROTOCOL - to apply OEM/SOC specific fix-ups for kernel / bootconfig / device tree
- GBL_EFI_SLOT_PROTOCOL - to identify boot mode, choose proper slot to boot from
- GBL_EFI_FASTBOOT_USB_PROTOCOL - fastboot USB transport
- GBL_EFI_FASTBOOT_PROTOCOL - to customize GBL fastboot implementation for the vendor needs
- GBL_EFI_IMAGE_LOADING_PROTOCOL (optional) - to customize GBL allocation logic
- Something else after we learn more about your requirements?

GBL_EFI_OS_CONFIGURATION_PROTOCOL

Motivation:

- This protocol provides a mechanism for the EFI firmware to modify OS configuration data:
  - Devicetree
  - Kernel command line
  - Bootconfig

Header file:

https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/libefi_types/defs/protocols/gbl_efi_os_configuration_protocol.h

Methods to implement by FW:

- fixup_kernel_commandline
- fixup_bootconfig
- build_device_tree
- fixup_device_tree

not stable API, subject to change

GBL_EFI_SLOT_PROTOCOL

Motivation:

- To read and write A/B slot metadata, boot reason / subreason (normal, bootloader, recovery, fastboot, etc)

Header file:

https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/libefi_types/defs/protocols/gbl_efi_ab_slot_protocol.h

Methods to implement by FW:

- load_boot_data
- get_slot_info, get_current_slot, set_active_slot
- set_slot_unbootable
- mark_boot_attempt
- reinitialize get_boot_reason
- set_boot_reason
- flush

not stable API, subject to change

Fastboot

FW fragmentation affects fastboot as well.

Supported transports by GBL:

- TCP via EFI_SIMPLE_NETWORK_PROTOCOL*
- USB via custom GBL_EFI_FASTBOOT_USB_PROTOCOL

* GBL fastboot uses EFI_SIMPLE_NETWORK_PROTOCOL instead of high level protocols (TCP/UDP) to support FW with limited network capabilities (i.e u-boot).

GBL_EFI_FASTBOOT_USB_PROTOCOL

Motivation:

- Provide hardware-agnostic interface to implement fastboot over USB transport (instead of EFI_USB_IO_PROTOCOL)

Header file:
https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/libefi_types/defs/protocols/gbl_efi_fastboot_usb.h

Documentation:
https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/docs/GBL_EFI_FASTBOOT_USB_PROTOCOL.md

Methods to implement by FW:

- fastboot_usb_interface_start
- fastboot_usb_interface_stop
- fastboot_usb_receive
- fastboot_usb_send

not stable API, subject to change

GBL_EFI_FASTBOOT_PROTOCOL

Motivation:

- To allow OEM/SOC specific fastboot functionality (variables, commands)

Header file:
https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/libefi_types/defs/protocols/gbl_efi_fastboot_protocol.h

Methods to implement by FW:

- get_var, start_var_iterator, get_next_var_args
- run_oem_function
- get_policy
- set_lock
- clear_lock
- get_partition_permissions
- wipe_user_data

** not stable API, subject to change*

What's next

- Getting your feedback and incorporating it (android-gbl@google.com)
- Finalize UEFI interfaces drafts
- Bring UEFI support to more FW used across ecosystem. LittleKernel UEFI support is already in-progress:
  - Add basic UEFI loader
  - Add UEFI protocol headers
- Bring more standardization to the TEE
- Support your board (android-gbl@google.com)

Useful links

- Source code
  https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/
- Main readme (including how to run with QEMU, Cuttlefish)
  https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/README.md
- GBL documentation
  https://cs.android.com/android/platform/superproject/main/+/main:bootable/libbootloader/gbl/docs/
- GBL development
  https://android-review.googlesource.com/q/project:platform/bootable/libbootloader
- Point of contact android-gbl@google.com, dimorinny@google.com

Interested in your opinion/experience

- Anything I share here today

- GBL_EFI_OS_CONFIGURATION_PROTOCOL. Want to learn more about cases when you need to modify fdt/command line/bootconfig in bootloader runtime.
- Provide generic UI by utilizing EFI_GRAPHICS_OUTPUT_PROTOCOL?
- UEFI and multithreading/parallelization. Currently we optionally supported EFI_BLOCK_IO2_PROTOCOL which allows async disk IO. Any experience with parallelization with UEFI?

Thank you