How to optimize the boot time

Stable: 25.09.2020 - 08:24 | Revision: 25.09.2020 - 08:24

A quality version of this page, approved on *25 September 2020*, was based off this revision.

Contents

1 Article purpose

The purpose of this document is to provide information on how to measure and improve the boot-time[1] of a typical STM32MP15 Linux system. This article is not an exhaustive list of possible optimizations, and those that are considered insufficiently reliable for industrial use are intentionally omitted.

2 Overview

On a typical STM32MP1 Linux system, the boot-chain is performed, respectively, by: the ROM code, TF-A, U-Boot, the Linux kernel, and the user-land[2]. All these components except the ROM code can be modified, and thus configured to start more quickly. For each of them, the procedure is the same: features that are not required at boot-time must be initiated after system boot (or disabled), and features that improve the boot time must be enabled.

3 Measuring boot-time

Before optimizing the performance of any piece of software, the time duration of each part must be considered so that the effort can be focused effectively.

3.1 Using a serial console

One of the easiest way sto measure the boot-time of a Linux system is to observe traces emitted on a serial console using timing software, such as serialgrab or like the script File:Measure-timing.txt based on microcom and p2f:

```
PC $> microcom -p /dev/ttyACM0 | bash Measure-timing.txt
Waiting for board reset...
NOTICE:  Model: STMicroelectronics STM32MP157C eval daughter on eval mother
NOTICE:  Board: MB1263 Var1 Rev.C-01
…
U-Boot 2018.11-stm32mp-r2 (Nov 14 2018 - 16:10:06 +0000)

CPU: STM32MP157AAA Rev.B
Model: STMicroelectronics STM32MP157C eval daughter on eval mother
…
Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
…
Freeing unused kernel memory: 1024K
Run /sbin/init as init process
…
ST OpenSTLinux - Weston - (A Yocto Project Based Distro) 2.6-openstlinux-4.19-thud-mp1-
19-02-20 stm32mp1 ttySTM0

stm32mp1 login:

Timing results:
FSBL:  1.23s
SSBL:  2.34s
Linux: 2.34s
init:  1.23s
total: 7.14s
```

Note: these are illustrative figures since the boot-time depends on too many parameters to provide meaningful figures here.

3.2 Using hardware timers

When measuring traces from a serial console is not precise enough, or when traces are not available, it is possible to use hardware timers. Some components of the boot chain provide timing information based on hardware timers; for instance with U-Boot, the following options can be enabled:

```
CONFIG_BOOTSTAGE=y
CONFIG_BOOTSTAGE_REPORT=y
```

This prints on the serial console — just before booting the OS — something similar to the output below (illustrative figures):

```
Starting kernel ...

Timer summary in microseconds (11 records):
       Mark    Elapsed  Stage
          0          0  reset
    123,456    123,456  board_init_f
  2,345,678  2,222,222  board_init_r
  3,456,789  1,111,111  id=64
  3,567,890    111,101  id=65
  3,678,901    111,011  main_loop
  4,567,890    888,989  bootm_start
  4,678,901    111,011  id=15
  4,789,012    110,111  start_kernel

Accumulated time:
                 23,456  dm_r
                567,890  dm_f
Booting Linux on physical CPU 0x0
```

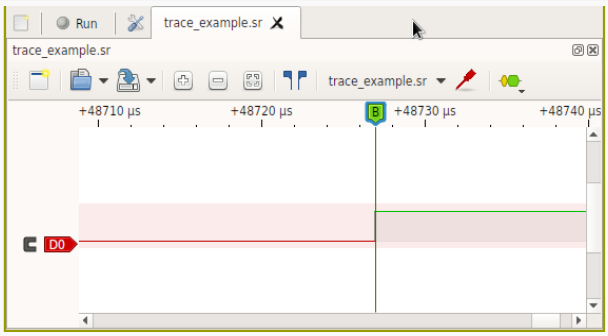
If the serial console is not available, this timing information can also be stored in the device-tree or in memory:

```
CONFIG_BOOTSTAGE_FDT=y
CONFIG_BOOTSTAGE_STASH=y
```

It is possible to add this feature to boot-chain components that do not implement it by default, like TF-A. See the file `arch/arm/cpu/armv7/arch_timer.c` from U-Boot for an example.

3.3 Using GPIOs

When the console is disabled, the boot-chain components can be modified to trigger events on GPIOs according to the current stage of the boot-process. Then a logical analyzer can be used to measure the time between each such event.



Example of trace captured by Sigrok and displayed by PulseView.

4 Optimizing boot-time

4.1 TF-A

4.1.1 Configuration

The TF-A execution time can be noticeably reduced by disabling features that are not required. To achieve this, the right options have to be specified when building the TF-A; for instance with a system that boots exclusively from NAND logic connected through the FMC:

```
PC $> make STM32MP1_DEBUG_ENABLE=0    \
          STM32MP1_UART_PROGRAMMER=0 \
          STM32MP1_USB=0             \
          STM32MP1_QSPI_NOR=0        \
          STM32MP1_QSPI_NAND=0       \
          STM32MP_FMC_NAND=1         \
          STM32MP_EMMC=0             \
          STM32MP_SDMMC=0            \
          …
```

These build options must of course be adjusted according to the expected usage and the TF-A version.

4.1.2 I2C timing

I2C driver computes device timing at init, in order to guarantee correct data hold and setup times. The execution time of the computation algorithm is not so important, but with all associated device tree accesses, it can last several 10ms. It has to be done for each instance, so it can happen several times in the boot sequence.

For specific needs, a hard-coded timing (and its related frequency) can be used in order to optimize boot performances. Dedicated fields can be added in the I2C instance related handle structure. Then, once retrieved, these boot values can be configured by the client, this avoids to compute timings (and to do several device tree accesses) on first init for each instance.

4.2 U-Boot

4.2.1 Delay

Before loading the Linux kernel and its device tree, U-Boot, by default, waits one second for a potential user input. This behavior — undesirable when boot time is a concern — can be easily removed by specifying `bootdelay=0` in `include/configs/stm32mp1.h`.

4.2.2 Configuration and device tree

More broadly, removing support for all unused devices from U-Boot configuration and unused nodes from the device-tree drastically reduces the U-Boot execution time, since this eliminates the initialization time of those devices and the time to parse the device-tree.

There are also a couple of U-Boot features that are specially designed to improve the boot-time, for example:

```
CONFIG_MTD_UBI_FASTMAP=y
CONFIG_SYS_MALLOC_CLEAR_ON_INIT=n
```

Regarding support for UBI fastmap (for NOR and NAND storage media), please see the "Linux" section below for more information.

4.2.3 Configuration access

By default U-Boot searches a boot configuration file `extlinux.conf` or a boot script `boot.scr.uimg` from a bootable partition ("bootfs" for OpenSTLinux), see U-Boot_overview#Generic_Distro_configuration for details. Such access to a file system on storage media can take a short of time, but fortunately it is possible to embed a boot command into the U-Boot binary instead. To do this, the whole U-Boot environment must be specified from scratch:

```
CONFIG_USE_DEFAULT_ENV_FILE=y
CONFIG_DEFAULT_ENV_FILE="path/to/env.txt"
```

Or the U-Boot "distro" mode must be disabled:

```
CONFIG_DISTRO_DEFAULTS=n
```

In the latter case, be aware that if booting from an ext2/4 partition — typically when booting from an SD card or from eMMC — the following options must be explicitly selected (they were previously selected implicitly by CONFIG_DISTRO_DEFAULTS):

```
CONFIG_CMD_EXT2=y
CONFIG_CMD_EXT4=y
```

In both cases only `CONFIG_ENV_IS_NOWHERE` must be set to `y` (remove all the other CONFIG_ENV_IS...) , and the environment variable `bootcmd` must contain the expected boot command, for example:

```
CONFIG_BOOTCOMMAND="run bootcmd_ubi"
```

```
bootcmd_ubi=env set bootargs ubi.mtd=UBI root=ubi0:boot \
                         rootfstype=ubifs rootwait  \
                         rw console=ttySTM0,115200; \
          ubi part UBI;                             \
          ubifsmount ubi0:boot;                     \
          ubifsload 0xc2000000 /zImage;             \
          ubifsload 0xc4000000 /stm32mp157c-ev1.dtb; \
          bootz 0xc2000000 - 0xc4000000
```

4.3 Linux

4.3.1 Configuration and device-tree

For U-Boot, one of the most efficient ways to decrease the Linux boot time is to remove support for all unused devices from its configuration and device-tree, since this eliminates the time taken to initialize those devices. Also, support for devices and features that are required but not mandatory at boot-time can be compiled as modules, and then loaded by the user-land once the boot-process is done (see the "Init" subsection below).

4.3.2 Traces

Linux boot traces have a size of about 2 Kbytes, so they take about 2 seconds to transfer on a serial link configured at 128 Kbit /s. If this is an issue, the following kernel parameters (`bootargs` variable in U-Boot) can be used to remove these traces:

```
quiet loglevel=0
```

This is automatically done by U-Boot when silent mode is required (see CONFIG_SILENT_CONSOLE and CONFIG_SILENT_U_BOOT_ONLY): silent=1 in used U-Boot environment.

4.3.3 UBI volume

When partitions such as "bootfs" and "rootfs" are stored on a UBI volume, the following actions are highly recommended to greatly reduce the volume attachment time at boot-time, both by U-Boot and by the Linux kernel:

1. Decrease the size of the UBI volume

```
and
```

2. Enable the fastmap feature. For this, the `CONFIG_MTD_UBI_FASTMAP` option must be set to `y` both for Linux and U-Boot, and `ubi.fm_autoconvert=1` has to be added to the kernel boot parameters. Note that the very first boot is used to create the fastmap information, thus this one does not get faster.

A lot of other volume/file-systems exist for Flash media, such as JFFS, LogFS, F2FS, and so on. Depending on the use-case, some may be faster than others.

4.4 User-land

4.4.1 Init

The very first user-land process launched by the Linux kernel is `init`; it is in charge of launching other processes in the right order. As a consequence, removing all unnecessary services can greatly speed up this part of the boot-process. The way services can be removed highly depends on the system in use (systemd, OpenRC and so on), so please refer to its documentation.

Ultimately, when no services are required, the `init` system can be replaced by the final application itself; either by storing its binary in `/sbin/init`, or by passing the following Linux boot parameter:

```
init=/path/to/application/binary
```

4.4.2 Framework

For graphical applications, the choice of display framework can seriously impact the startup time of the application itself. For instance, one could use the Linux framebuffer instead of Weston or Xorg, since the former is set up faster than the latter.

Likewise for video applications; if the boot-time is a concern, direct use of the V4L2 interface instead of higher-level interfaces such as GStreamer is recommended.

5 Conclusion

There is no universal recipe to improve the boot-time of a Linux system, since it depends on the contraints of the final use-case. However the universal rule is: **always benchmark**, before and after optimizing.

6 Further reading

- https://www.e-consystems.com/Articles/Product-Design/Linux-Boot-Time-Optimization-Techniques.asp
- https://www.toradex.com/blog/embedded-linux-boot-time-optimization
- https://bootlin.com/blog/tag/boot-time
- https://bootlin.com/doc/training/boot-time/boot-time-slides.pdf

7 References

- Sometimes referred to as startup-time.
- Sometimes referred to as user-space.

Linux® is a registered trademark of Linus Torvalds.

Read Only Memory

Trusted Firmware for Arm Cortex-A

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Central processing unit

First Stage Boot Loader

Second Stage Boot Loader

Operating System

Universal Asynchronous Receiver/Transmitter

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

Memory Technology Device

SD memory card (https://www.sdcard.org)

former spelling for e•MMC ('e' in italic)