Android tracing, monitoring and debugging

Contents

Stable: 15.02.2021 - 10:40 / Revision: 28.01.2021 - 09:02

Stable: 15.02.2021 - 10:40 / Revision: 28.01.2021 - 09:32

A quality version of this page, approved on *15 February 2021*, was based off this revision.

This article provides useful information to start using Android™ tracing, monitoring and debugging environments.

Contents

1 Specific Linux® tracing and debugging tools

In Android™ various Linux tracing and debugging tools are available. To get the global picture, refer to Linux tracing, monitoring and debugging page.

2 Android tools

2.1 Application debugging

You can use Android Studio to debug your Android application. Please read the official documentation about Android Studio[1].

2.2 logcat

The Android logging system provides a mechanism for collecting and viewing system debug output. Logs from various applications and portions of the system are collected in a series of circular buffers, which then can be viewed and filtered by the `logcat` command. Use `logcat` from an ADB shell to view the log messages.

For more information refer to the Android developer site[2].

2.3 perfetto

`Perfetto` is an open-source project for performance instrumentation and tracing of Android platforms and user-space apps.

For more information refer to the perfetto page.

3 References

- https://developer.android.com/studio/debug
- https://developer.android.com/studio/command-line/logcat

Stable: 15.02.2021 - 12:33 / Revision: 12.02.2021 - 08:25

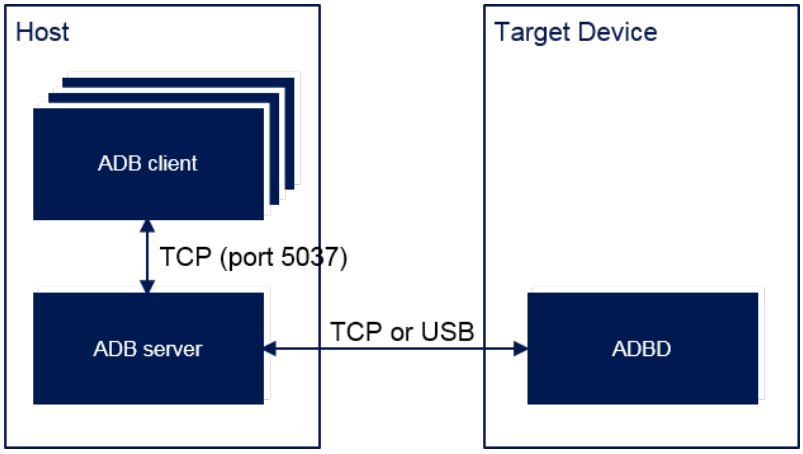
A quality version of this page, approved on *15 February 2021*, was based off this revision.

The Android Debug Bridge (ADB) is a versatile command-line tool that lets you communicate with a device (an emulator or a connected Android device).

This article is intended for Developer Package or Distribution Package users (see Which Package better suits your needs for more information).

Contents

1 Overview



- The **ADB client** is an executable that can be launched with a subcommand such as `adb shell` or `adb logcat`.
- The **ADB server** acts as a proxy between adb clients and adbd.
- The **ADBD daemon** is started by executing an init command on the target device.

The target device can be connected using different solutions:

- How to connect over USB (default)
- How to connect over Ethernet
- How to connect over Wi-Fi

As soon as it is connected, you can start testing your application with `Android Studio`.

It is also possible to execute all ADB commands (see ADB commands for more details).

For that purpose, search for the ADB tool in your environment: Host computer installation.

Information

If you are using a Distribution Package, please refer to ADB for Distribution Package

2 Host computer installation

ADB is part of the Android SDK (loaded using `Android Studio`) for Linux®, Windows® and OS X®.

At this stage, ADB commands can be executed directly from the `Android Studio` terminal:

```
cd <SDK path>
cd platform-tools
```

The *<SDK path>* can be determined by opening the SDK Manager from `Android Studio`.

3 Commands

The ADB commands enable the execution of a variety of actions on the target device, such as installing and debugging apps. They also provide access to a Unix shell that can be used to run a variety of commands.

See ADB commands summary for more information.

Several useful commands:

- `adb devices`: check connected devices
- `adb logcat`: trace
- `adb root`: give root privileges
- `adb shell`: open a console on the device
- `adb remount`: remount read-only partition in read-write to allow updating their containing (use overlayfs mechanism)
- `adb push <file>`: push a file on the device
- `adb pull <file>`: pull a file from the device
- `adb install <package>`: install an application package (APK) on the device

4 Device connection

4.1 How to connect over USB

Follow the steps below to connect your device:

- Configure your host computer
- Configure your target device
- Check the device connection

4.1.1 Host computer configuration

First check if you can already see the device connected: Template:PC adb devices

The installation and configuration depend on the host computer OS used:

- Linux Host (Ubuntu) case
- Windows Host case
- OS X Host case

4.1.1.1 Linux® Host (Ubuntu)

In Linux environments, the Android USB drivers are built-in. The only action required is to set the target device information.

Information

To peform this action, you need administrator rights. In addition, you may need to add *sudo* in front of each executed command

To do this, open a terminal and

- Create (or update if it already exists) the *51-android.rules* file in */etc/udev/rules.d/* with the following information
  - *idVendor* = 0483 (STMicroelectronics vendor)
  - *idProduct* = 0adb (ADB on STMicroelectronics device)
  - *Mode* = 0666 (read/write permissions)
  - *Group* = plugdev (Unix group which owns the device node)

Information

Check if you belong to the *plugdev* group by executing the following command:

```
$ groups
```

Example:

```
# ADB on STMicroelectronics devices
SUBSYSTEM=="usb", ATTR{idVendor}=="0483", ATTR{idProduct}=="0adb", MODE="0660", GROUP="
plugdev"
```

- Make sure that the access rights to the created file are the correct ones:

```
chmod a+r /etc/udev/rules.d/51-android.rules
```

At this stage, your USB driver is correctly installed and configured.

4.1.1.2 *Windows Host*

In Windows environments, it is required to install the USB driver and set the target device information.

Get back a compatible driver here after (Windows 10): st-android-winusb.zip

Install it:

- Connect the device to your computer's USB port.
- Open Windows **Settings**, then select **Devices**, then '"Devices and Printers"'
- Right-click the name of the device you connected, and then select **Properties**
- In the **Hardware** tab, select again **Properties**
- In the **Driver** tab, you can select **Update Driver** and then select **Browse my computer for driver software** and click **Next**
- Locate the USB driver folder you just loaded
- Click **Next** to install the driver.

At this stage, the USB driver is correctly installed and configured.

4.1.1.3 *OS X Host*

In OS X environments, the Android USB drivers are built-in and no manual configuration is required to install and configure the USB driver.

4.1.2 Target device configuration

Prerequisite: Use ST Android distribution to start the target device and access the Android user interface.

On the target device:

- Disconnect the USB cable between the target device and the host computer (if it is connected).
- Find the *Settings > Developer options* configuration screen on your target device (if it is not visible, select *Settings > About device* and click the *Build number* menu entry seven times).
- Enable the USB Debugging option from the *Settings > Developer options* menu.
- Reconnect the USB cable between the target device and the host computer.
- If an alert is displayed on your target device requesting permission to "Allow USB debugging," click *OK. It is recommended to check the "Always allow from this computer" option to avoid this from happening each time the target device is connected to the same host computer.*

4.1.3 Checking the device connection

First, ensure that the device is connected on the host computer through an USB cable, and that it has been started.

With `Android Studio`, it's easy to check if the target device is available. For example, click on `Logcat` at the bottom and check the devices seen in the list, `No connected devices` is seen otherwise.

Information

The name of the STM32MPU devices start with *STMicroelectronics*

4.2 How to connect over Ethernet

To use ADB over an Ethernet connection, connect the host computer and the target device on the same network.

First, follow the instructions provided in How to connect over USB.

Then:

- Connect the USB cable between the target device and the host computer.
- Open a terminal on your host computer.
  - Go to the *<SDK path>/platform-tools/* directory (see Host computer installation to know how to get the <SDK path>).
  - Execute *adb shell ifconfig* to get the target IP address <device_ip_address>.
  - Execute *adb tcpip 5555* to set the TCP/IP connection over port 5555.
- Disconnect the USB cable between the target device and the host computer.
- Execute *adb connect <device_ip_address>* to reconnect on the device through Ethernet

4.3 How to connect over Wi-Fi

Follow the instructions provided in the *Android Studio* website: Connect to a device over Wi-Fi

5 ADB for Distribution Package

5.1 Installing the Host computer

By default, *ADB* is built when you compile the ST Android distribution. It is automatically added to your environment's $PATH.

Another possibility is to install ADB but executing *apt-get install android-tools-adb*, although this may be a hindrance to the development in case of ST Android distribution and OS ADB version mismatch (the server is restarted every time it is used with a mismatched version of an ADB client).

The device is not necessarily visible in the terminal when executing `adb devices`. If this is not the case:

- Create a file within ~/.android/ directory named adb_usb.ini.
- Add one line containing the STMicroelectronics idVendor:

```
0x0483
```

5.2 Implementing ADB

ADB source code is stored in *system/core/adb* (common files between host and target device are differentiated by the ADB _HOST tag).

Three properties can be configured for ADB:

- `persist.adb.trace_mask` used to enable or disable ADBD traces on target device.

  Possible values (*listed in* `system/core/adb/adb_trace.cpp`): 0, 1 (or all), adb, sockets, packets, rwx, usb, sync, sysdeps, transport, jdwp, services, auth, fdevent or shell

```
Board $> stop adbd
Board $> setptop persist.adb.trace_mask = <value>
Board $> start adbd
```

Information

Traces are available in `/data/adb/adb-<date_at_start>`

- `ro.adb.secure` used to enable or disable the authentication mechanism (1: authentication enabled, 0: authentication disabled). This property can not be modified dynamically. It is considered only in case of `eng` or `userdebug` image builds. It is ignored otherwise (the authentication mechanism is enabled by default for user build).
- `service.adb.tcp.port` used to enable the TCP/IP port if a wireless connection is used (this property is not set by default as it is not necessary for USB).

ADB is an Android specific gadget device available in user space. functionfs must be mounted since it is used to create this gadget device and register it to the USB Device Controller (UDC). Three endpoints are used (one for control, one for data input, one for data output).

It is also possible to create a composite device between ADB and other protocols, using the configfs mechanism.

ADB over USB is the default configuration within the Android baseline. By default, it is configured as follows:

- `persist.sys.usb.config`: [adb] USB ADB gadget device
- `sys.usb.config`: [adb] USB ADB gadget device
- `service.adb.root`: [0] no root privileges

The ADB over wireless connection can be set within your distribution (Ethernet or Wi-Fi) by adding the following property in */device/stm/<STM32Series>/<BoardId>/device.mk*:

```
PRODUCT_PROPERTY_OVERRIDES += service.adb.tcp.port=5555
```

Android debug bridge (Android specific)

Android debug bridge daemon (Android specific)

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Linux® is a registered trademark of Linus Torvalds.

Operating System

technology for wireless local area networking with devices based on the IEEE 802.11 standards

Read Only

USB Device Controller

Configuration File System (See https://en.wikipedia.org/wiki/Configfs for more details)

stm32mp1

eval disco (Generic term used, to complete configuration modules paths depending on used board)

Stable: 21.09.2021 - 09:12 / Revision: 21.09.2021 - 09:11

A quality version of this page, approved on *21 September 2021*, was based off this revision.

Contents

1 Article purpose

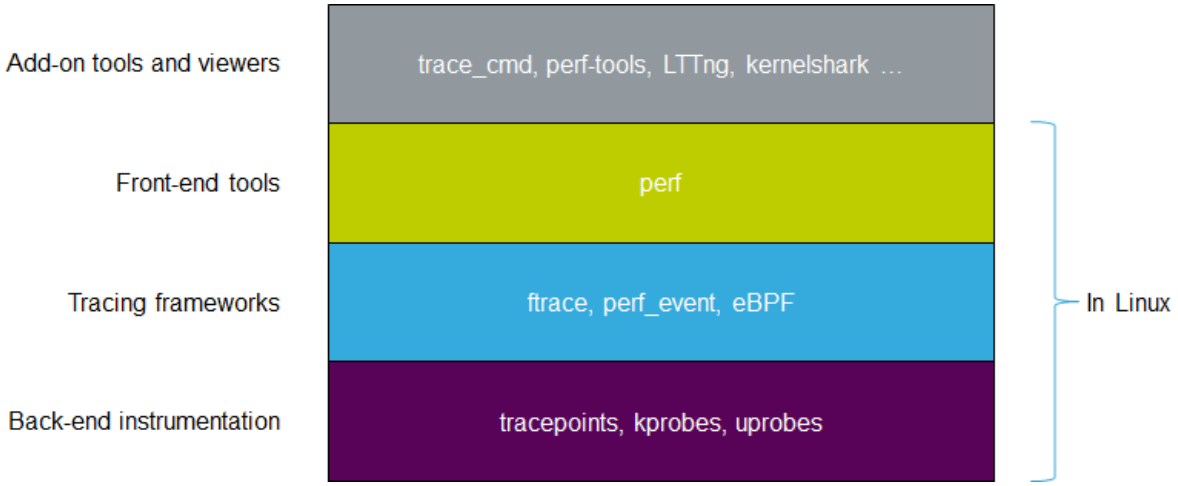
This article provides useful information to start using Linux® tracing, monitoring and debugging environments.

Two entry points are proposed in this article:

- Linux tracing, monitoring and debugging tools, which gives an overview of some Linux® tools including usage and application domain. **This chapter is useful when you already know the domain or the interface to search for**.
- Trace and debug overview per Linux software frameworks, which points to articles explaining how to get trace and debug information about the Linux® software frameworks that are relevant for the STM32MPU Embedded Software. **This chapter is useful when you know the Linux® framework to search for**.

2 Linux trace architecture overview

The Linux® trace architecture can be organized into four levels as shown in the figure below (*inspired by Brendan Gregg presentation*[1]):



2.1 Back-end instrumentation

The back-end instrumentation provides tracing sources built in the Linux® kernel. They are split into three categories:

- **tracepoints**: kernel static tracing, statically placed at logical places in the kernel. It provides key event details as a "format" string.
- **kprobes**: kernel dynamic tracing. It allows to trace function calls, returns and line numbers.
- **uprobes**: dynamic user-level tracing.

2.2 Tracing framework

Also named tracers, they use tracing sources.

Tracing frameworks include kernel in-tree tracers such as ftrace and perf_events, and out-of-tree tracers such as SystemTap and sysdig.

2.3 Front-end tools

Front-end tools come on top of tracers and help to configure them. For example:

- trace-cmd or LTTng for ftrace
- perf or perf-Tools for perf_events

2.4 Add-on tools and viewer

Add-on tools are also on top of tracers. However, they are not embedded inside the Linux® kernel.

Viewer tools propose Visual interpretation of trace data. For example:

- **kernelshark** for ftrace/trace-cmd

- Trace Compass[2] for LTTng (and more)
- Flame Graph[3] for perf

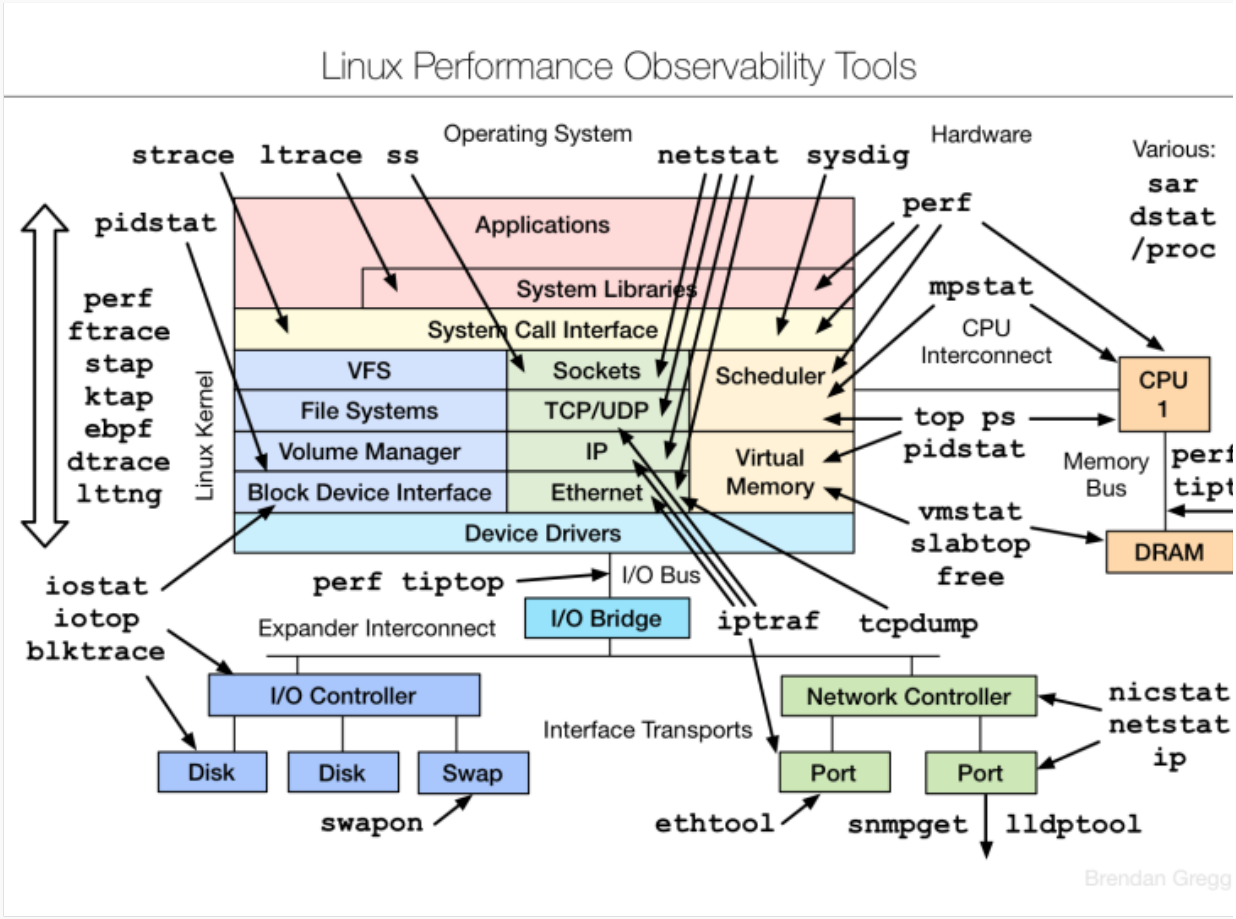
3 Linux tracing, monitoring and debugging tools

Linux® provides many tools that are either dedicated to one function or multifunction (generic).

They cover both Linux® kernel and Linux® user space.

3.1 Domain mapping

The following mapping, done by Brendan Gregg [4], shows the different existing tools associated to the different Linux® frameworks.



Note: The above image has been created by **Brendan Gregg** (Netflix) and can be found on his official web site.

3.2 Tool overview

The following table provides a brief description of the tool, as well as its availability depending on the software packages:

: this tool is either present (ready to use or to be activated), or can be integrated and activated on the software package.

: this tool is not present and cannot be integrated, or it is present but cannot be activated on the software package.

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
blktrace	Tracing tools	**blktrace**[5] generates traces of the I/O traffic on block devices (SD card, USB, eMMC...)	✗	✓	✓	✗	✗	✓
systemd core dump	Debugging tools	**systemd core dump**: generates core dump files on Linux	✓	✗	✓	✗	✗	✗
		ethtool[6] allows to query or control network driver and						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
ethtool	Monitoring tools	hardware settings, in particular for wired Ethernet devices.	✓	✓	✓	✗	✗	✗
ftrace	Tracing tools	**ftrace**[7] (Function Tracer) is a powerful kernel tracing utility that is able, for instance, to trace every kernel function calls and kernel events without adding any	✗	✗	✓	✗	✗	✓

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		extra code in your kernel source code						
GDB	Debugging tools	The GNU Project debugger, **GDB**[8], allows monitoring program execution, or what the program was doing at the moment it crashed.	⊗*	✓	⊗**	⊗	⊗	✓
			** Cross compile gdb and openocd binaries are required and only available from Developer Package.* *** It is recommended to use the Developer Package to run the gdb debug session, which provided all dependencies*			** Cross compile gdb and openocd binaries are required and only available from Distribution Package.*		
		ifconfig[9] is a system administration utility for networ						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
ifconfig	Monitoring tools	k interface configuration. **ifconfig is deprecated and has been replaced by *ip*** (A web page provides a comparison between ifconfig and ip [10])	✓	✓	✓	✓	✓	✓
		ip[11] shows / manipulates routing, devices, policy routing and tunnels						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
ip	Monitoring tools	of network interfaces. ip replaces the deprecated command ifconfig	✓	✓	✓	✓	✓	✓
kmemleak	Monitoring tools	**kmemleak**[12] provides a means to detect possible kernel memory leaks in a similar way to a tracing garbage collector, with the difference that the orphan	✗	✓	✓	✗	✗	✓

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		objects are not freed, but only reported via /sys/kernel/debug/kmemleak.						
		trace-cmd[13] command interacts with the Ftrace tracer that is built inside the Linux kernel. It interfaces with the Ftrace specific files found in the debugfs						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
trace-cmd	Tracing tools	file system under the tracing directory. **kernelshark**[14] is a front-end reader of trace-cmd output. "trace-cmd record" and "trace-cmd extract" create a trace.dat (trace-cmd.dat) file. kernelshark can read this file, and produce a graph and list view of the corresponding data.	✗	✗	✓	✗	✗	✗

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
ltrace	Tracing tools	**ltrace**[15] is used to display the calls to shared libraries made by a userspace application. **ltrace** is a userspace application. *Its use is very similar to strace .*	✗	✗	✓	✗	✗	✗
		LTTng[16] is an open source tracing framework for Linux kernel and						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
LTTng	Tracing tools	user spaces. It is a powerful tool that can be used for many purposes. LTTng traces need to be processed /displayed with a host tool such as Trace Compass[17], based on Eclipse plugin for increased portability.	✗	✗	✓	✗	✗	✗
		netdat						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
netdata	Monitoring tools	**a**[18] is a system for distributed real-time performance and health monitoring. It provides unparalleled insights, in real-time, of everything happening on the system it runs (including applications such as web and databa	✓	✓	✓	✗	✗	✗

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		servers), using modern interactive web dashboards.						
netstat	Monitoring tools	**netstat** [19] prints network connections, routing tables, interface statistics, masquerade connections, and multicast membership information.	✓	✓	✓	✓	✓	✓
	Monitor	**perf**[20] is a Linux user				✗*	✗*	✗*

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
perf	ing tools	space tool, which allows getting system performance figures	✓	✓	✓	*Note: **simpleperf**[21] is present as equivalent but with less options*		
strace	Tracing tools	**strace**[22] is able to intercept and record the system calls which are called by a process and the signals which are received by another process.	✓	✓	✓	✓	✓	✓
		sysprof[23] is a statistical,						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
sysprof	Monitoring tools	system-wide profiler for Linux. It helps in finding the functions in which a program spends most of its time. **sysprof** proposes a user interface available directly on the board display screen.	✓	✓	✓	✗	✗	✗
		The **sysstat** [24] tool suite contains utilities to monitor the						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		system performance and usage activity. It contains various utilities, common to many commercial Unix distributions, as well as tools that can be scheduled (via a scheduler such as cron) to collect and historize performance and activity data: • iostat: reports CPU statistics and input						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
sysstat	Monitoring tools	/output statistics for block devices and partitions. • **mpstat**: reports individual or combined processor related statistics. • **pidstat**: reports statistics for Linux tasks (processes): I/O, CPU, memory, etc. • **sar**: collects, reports and saves system activity information (CPU, memory, disks,	✓	✓	✓	✗	✗	✗

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		interrupts, network interfaces, TTY, kernel tables, etc.) sadf: displays data collected by sar in multiple formats (CSV, XML, JSON, etc.). This command can also be used to exchange data with other programs or to draw graphs illustrating the various activities collected by sar using SVG						

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		(Scalable Vector Graphics) format.						
tcpdump	Monitoring tools	**tcpdump**[25] is a common packet analyzer that runs under the command line. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which	✓	✓	✓	✓	✓	✓

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		the computer is connected.						
		The **top**[26] program provides a dynamic real-time view of a running system. It can display system summary information as well as a list of tasks currently being managed by the Linux kernel. The						

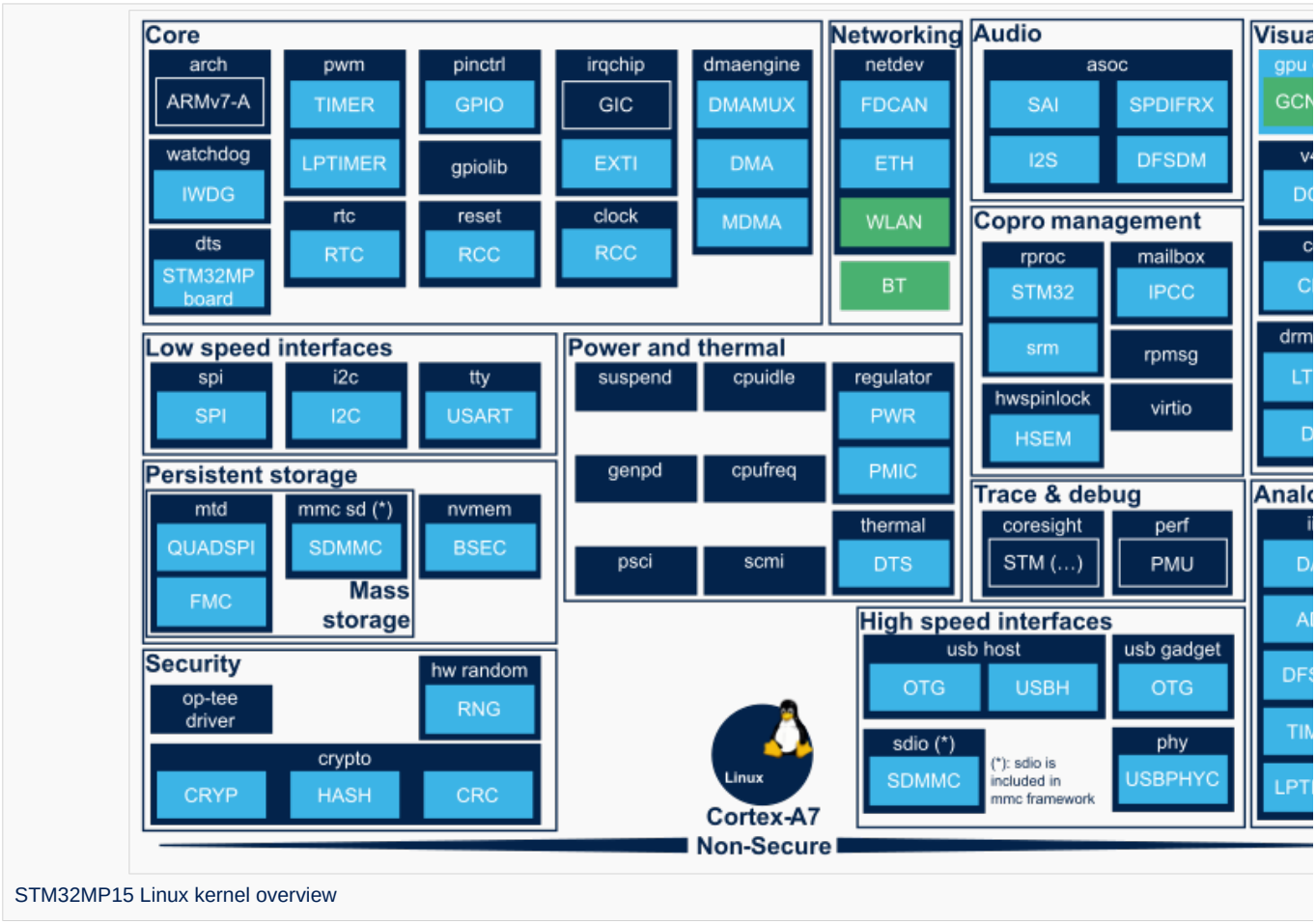
Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
top	Monitoring tools	types of system summary information shown and the types, order and size of information displayed for tasks are all user configurable and that configuration can be made persistent across restarts	✓	✓	✓	✓	✓	✓

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		. (*Extracted from man page*[26])						
valgrind	Monitoring tools	**valgrind**[27] is an instrumentation framework for building dynamic analysis tools. Some Valgrind tools can automatically detect many memory management and threading bugs, and	✓	✓	✓	✗	✗	✗[28]

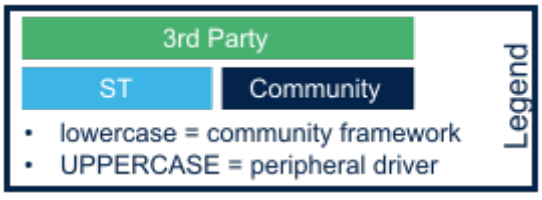
Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
		profile your programs in detail. This is tool for Linux application analysis.						

4 Trace and debug overview per Linux software frameworks

The picture below allows accessing to different Linux software frameworks which provide specific trace and debug information in their "*How to trace and debug the framework*" dedicated chapter.



STM32MP15 Linux kernel overview



5 Tips

How to find Linux kernel driver associated to a device.

How to use the kernel dynamic debug.

6 Documentation and web articles

A lot of articles on the web mention Linux® kernel tracing and profiling. The following links provide a good introduction to these topics:

- Linux Performance Analysis - New Tools and Old Secrets: description of the Linux® technology and of the different tools available.
- Yocto project: Tracing and profiling: How to enable tracing and profiling tools using Yocto
- Brendan Gregg Linux performance page

Information

More general Linux performance information are available in nice slideshare presentation (Brendan Gregg), on the Brendan Gregg official web site or in LinuxCon2014 article.

Reference list:

- http://www.brendangregg.com/linuxperf.html
- https://www.eclipse.org/tracecompass
- http://www.brendangregg.com/flamegraphs.html
- http://www.brendangregg.com/linuxperf.html
- https://linux.die.net/man/8/blktrace
- https://linux.die.net/man/8/ethtool
- https://elinux.org/Ftrace
- https://www.gnu.org/software/gdb
- https://linux.die.net/man/8/ifconfig
- https://tty1.net/blog/2010/ifconfig-ip-comparison_en.html
- https://linux.die.net/man/8/ip
- http://www.procode.org/kmemleak/
- https://lwn.net/Articles/410200/
- http://rostedt.homelinux.com/kernelshark
- https://www.ltrace.org/
- http://lttng.org
- https://www.eclipse.org/tracecompass
- https://my-netdata.io
- https://linux.die.net/man/8/netstat
- https://perf.wiki.kernel.org/index.php/Main_Page
- https://source.android.com/devices/tech/debug/eval_perf
- https://strace.io/
- http://www.sysprof.com/
- http://sebastien.godard.pagesperso-orange.fr/
- http://www.tcpdump.org/
- 26.026.1 http://linux.die.net/man/1/top
- http://valgrind.org/
- Use AddressSanitizer (ASan) for Android: https://source.android.com/devices/tech/debug/asan

Linux® is a registered trademark of Linus Torvalds.

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

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

GNU dedugger, a portable debugger that runs on many Unix-like systems

Debug File System (See https://en.wikipedia.org/wiki/Debugfs for more details)

Central processing unit

TeleTYpewriter

Stable: 23.06.2020 - 13:54 / Revision: 02.06.2020 - 15:39

A quality version of this page, approved on *23 June 2020*, was based off this revision.

This article provides the basic information required to start using the **perfetto** [1] Android™ tool.

1 Introduction

The following table provides a brief description of the tool, as well as its availability depending on the software packages:

✓: this tool is either present (ready to use or to be activated), or can be integrated and activated on the software package.

✗: this tool is not present and cannot be integrated, or it is present but cannot be activated on the software package.

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
perfetto	Tracing tools	**perfetto**[1] is a performance instrumentation and tracing tool for Android™. • Open-source project for platform tracing • Trace processing and analysis • Web-based trace viewer UI	✗	✗	✗	✓*	✓*	✓

2 Getting started with perfetto

Perfetto instructions can be generated using the recording tool[2].

- Create record settings required
- Copy generated instructions
- Execute the copied instructions on a opened terminal with the device connected through USB, having ADB installed.

At the end of the trace execution, you can get back the trace:

```
adb pull /data/misc/perfetto-traces/trace
```

From this stage you can directly open the trace through the web viewer[3].

3 References

- [1.01.1] https://perfetto.dev/
- https://ui.perfetto.dev/#!/record
- https://ui.perfetto.dev

User Interface