Trace-cmd and kernelshark trace viewer

Contents

Stable: 09.10.2019 - 15:39 / Revision: 04.09.2019 - 09:19

Stable: 09.10.2019 - 10:09 / Revision: 04.09.2019 - 09:19

A quality version of this page, approved on *9 October 2019*, was based off this revision.

Contents

1 Article purpose

This article provides the basic information needed to start using the Linux tool: **trace-cmd**[1] and host trace viewer **kernelshark**[2]
.

2 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
		trace-cmd[1] command interacts with the Ftrace tracer that is built inside the Linux kernel. It interfaces with the Ftrace specific files found in the debugfs file system						

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	under the tracing directory. **kernelshark**[2] 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.	✗	✗	✓	✗	✗	✗

Note: in case you want to trace at boot time, it is recommended to use ftrace.

3 Installing the trace and debug tool on your target board

3.1 Using the STM32MPU Embedded Software distribution

3.1.1 Distribution Package

Since Yocto Thud version, **trace-cmd** recipe is no more present in openembedded-core meta package and so is not installed by default in all STM32MPU Software Packages.

Following recipes are proposed to integrate trace-cmd in STM32MPU Embedded Software distribution. It has to be added in *layers/meta-st/meta-st-stm32mp/recipes-kernel*.

- Create *trace-cmd* directory:

```
PC $> mkdir layers/meta-st/meta-st-stm32mp/recipes-kernel/trace-cmd
PC $> cd layers/meta-st/meta-st-stm32mp/recipes-kernel/trace-cmd
```

- Create *trace-cmd.inc* file with the following content:

```
SRCREV = "57371aaa2f469d0ba15fd85276deca7bfdd7ce36"
PV = "2.6.2"

inherit pkgconfig

FILESEXTRAPATHS =. "${FILE_DIRNAME}/trace-cmd:"

SRC_URI = "git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git;
branch=trace-cmd-stable-v2.6"

S = "${WORKDIR}/git"
```

- Create *trace-cmd_git.bb* file with the following content:

```
SUMMARY = "User interface to Ftrace"
HOMEPAGE = "http://git.kernel.org/"
LICENSE = "GPLv2 & LGPLv2.1"

require trace-cmd.inc

LIC_FILES_CHKSUM = "file://COPYING;md5=751419260aa954499f7abaabaa882bbe \
                    file://trace-cmd.c;beginline=6;endline=8;
md5=2c22c965a649ddd7973d7913c5634a5e \
                    file://COPYING.LIB;md5=edb195fe538e4552c1f6ca0fd7bf4f0a \
                    file://trace-input.c;beginline=5;endline=8;
md5=3ec82f43bbe0cfb5951ff414ef4d44d0 \
"

EXTRA_OEMAKE = "\
    'prefix=${prefix}' \
    'bindir=${bindir}' \
    'man_dir=${mandir}' \
    'html_install=${datadir}/kernelshark/html' \
    'img_install=${datadir}/kernelshark/html/images' \
    \
    'bindir_relative=${@oe.path.relative(prefix, bindir)}' \
    'libdir=${libdir}' \
```

```
    \
    NO_PYTHON=1 \
"

do_compile_prepend() {
    # Make sure the recompile is OK
    rm -f ${B}/.*.d
}

do_install() {
        oe_runmake DESTDIR="${D}" install
}
```

- Add *trace-cmd* package install to the st-image-weston image:

```
PC $> echo 'IMAGE_INSTALL_append += "trace-cmd"' >> layers/meta-st/meta-st-openstlinux
/recipes-st/images/st-image-weston.bbappend
```

- Rebuilt the *st-image-weston* target:

```
PC $> cd <build_directory>
PC $> bitbake st-image-weston
```

When testing the new image on the target board, trace-cmd can be found in the following path:

```
Board $> which trace-cmd
/usr/bin/trace-cmd
```

However, an additional configuration of the Linux kernel must be performed to obtain the trace.

This is the configuration used for **ftrace**. Please refer to Ftrace install to complete the configuration of the Linux kernel.

This is the reason why you can not use this tool in Starter and Developer Packages context.

4 Getting started

trace-cmd inherits all available configurations from Ftrace.

It is possible to get filtered trace with both function and graph function mode.

It is also possible to trace specific prints done in the source file.

You first need to mount the tracefs:

```
Board $> mount -t tracefs nodev /sys/kernel/tracing
```

To find out which tracers are available, simply cat the available_tracers file in the tracing directory:

```
Board $> cat /sys/kernel/tracing/available_tracers
function_graph function nop
```

Note that you must be placed on the target board within a directory path where you have access rights for writing the generated trace file.

4.1 Making a trace using function tracer mode

The example below shows how to filter on a set of functions that contains uart:

- To get a trace on UART, you can connect to the target board with the new console (i.e. through ssh), and then type a simple command, like "*ls*"
- Then press Ctrl-C to stop the trace

```
Board $> trace-cmd record -p function -l *uart*
  plugin 'function'
Hit Ctrl^C to stop recording
^CCPU0 data recorded at offset=0x15d000
    4096 bytes in size
CPU1 data recorded at offset=0x15e000
    4096 bytes in size
```

See below for reading the trace.

4.2 Making a trace using graph function tracer mode

The example below shows how to filter on a set of graph functions that contains uart:

- To get trace on UART, you can connect to the target board with the new console (i.e. through ssh), and then type a simple command, like "*ls*"
- Then press Ctrl-C to stop the trace

```
Board $> trace-cmd record -p function_graph -l *uart*
  plugin 'function_graph'
Hit Ctrl^C to stop recording
^CCPU0 data recorded at offset=0x15d000
    4096 bytes in size
CPU1 data recorded at offset=0x15e000
    4096 bytes in size
```

See below for reading the trace.

4.3 Reading the trace

Command to read the trace:

```
Board $> trace-cmd report
```

By default, it reads the previously created file *trace.dat* at the position where the command is entered.

If you want to specify an other file name:

```
Board $> trace-cmd report -i mytrace.dat
or
Board $> trace-cmd report mytrace.dat
```

5 To go further

5.1 Visualizing trace using kernelshark

5.1.1 Host installation (Linux environment)

```
PC $> sudo apt-get install kernelshark
```

5.1.2 Opening and reading trace-cmd trace

kernelshark can read trace-cmd report and produce a graph and list view of the data. Please refer to kernelshark home page[2] for detail.

- Get the trace from the board. It is possible to get the trace directly from the SDCard under use, or remotely from the target board:

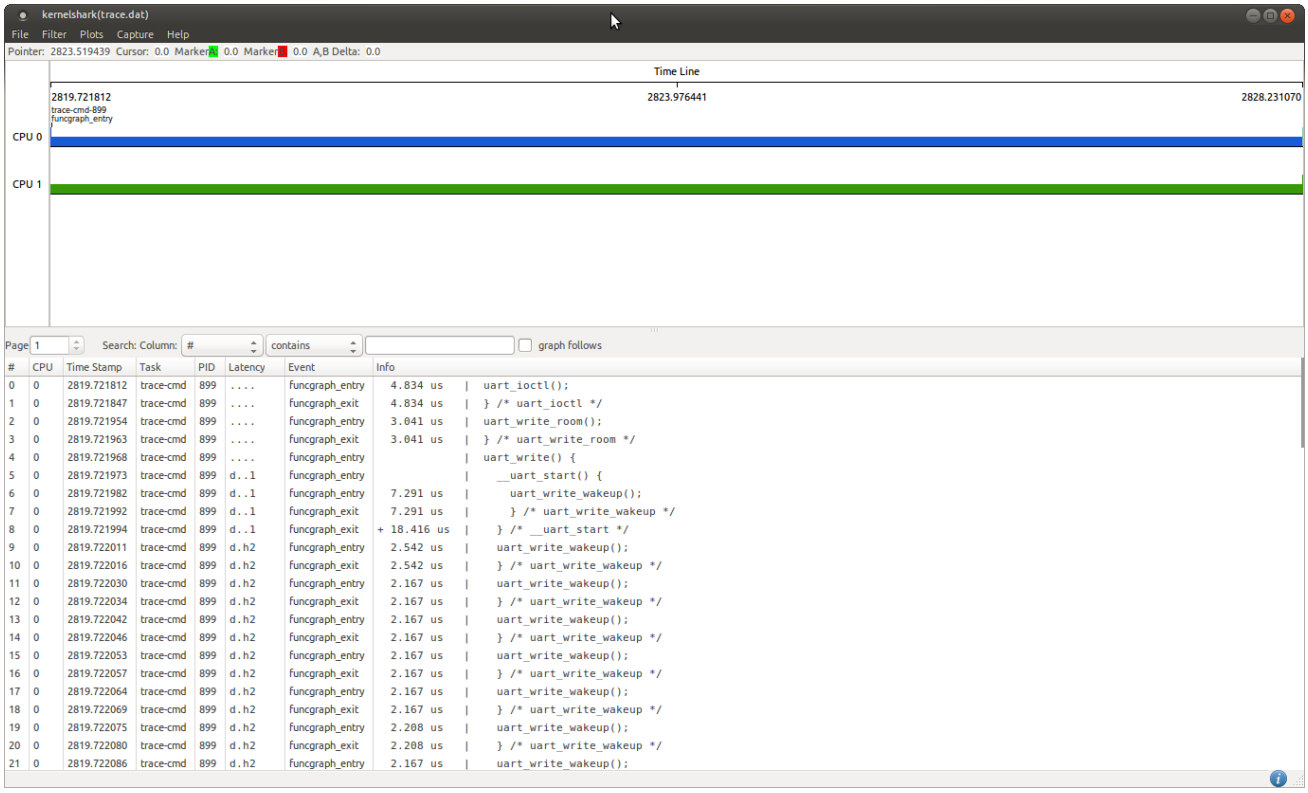
```
PC $> scp root@<board_ip_address>/<path_to>/trace.dat <path_on_host>
```

- Open trace with kernelshark

It can be done by opening kernelshark and then opening trace.dat file from the menu, or using the following command:

```
PC $> kernelshark <path_on_host>/trace.dat
```

Here is the example of a trace filtered with UART function_graph mode:



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	2819.721812	trace-cmd	899	....	funcgraph_entry	4.834 us | uart_ioctl();
1	0	2819.721847	trace-cmd	899	....	funcgraph_exit	4.834 us | } /* uart_ioctl */
2	0	2819.721954	trace-cmd	899	....	funcgraph_entry	3.041 us | uart_write_room();
3	0	2819.721963	trace-cmd	899	....	funcgraph_exit	3.041 us | } /* uart_write_room */
4	0	2819.721968	trace-cmd	899	....	funcgraph_entry	| uart_write() {
5	0	2819.721973	trace-cmd	899	d..1	funcgraph_entry	| __uart_start() {
6	0	2819.721982	trace-cmd	899	d..1	funcgraph_entry	7.291 us | uart_write_wakeup();
7	0	2819.721992	trace-cmd	899	d..1	funcgraph_exit	7.291 us | } /* uart_write_wakeup */
8	0	2819.721994	trace-cmd	899	d..1	funcgraph_exit	+ 18.416 us | } /* __uart_start */
9	0	2819.722011	trace-cmd	899	d.h2	funcgraph_entry	2.542 us | uart_write_wakeup();
10	0	2819.722016	trace-cmd	899	d.h2	funcgraph_exit	2.542 us | } /* uart_write_wakeup */
11	0	2819.722030	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
12	0	2819.722034	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
13	0	2819.722042	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
14	0	2819.722046	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
15	0	2819.722053	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
16	0	2819.722057	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
17	0	2819.722064	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
18	0	2819.722069	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
19	0	2819.722075	trace-cmd	899	d.h2	funcgraph_entry	2.208 us | uart_write_wakeup();
20	0	2819.722080	trace-cmd	899	d.h2	funcgraph_exit	2.208 us | } /* uart_write_wakeup */
21	0	2819.722086	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();

6 References

- [1.01.1] https://lwn.net/Articles/410200/
- [2.02.12.2] http://rostedt.homelinux.com/kernelshark

- Useful external links

Document link	Document Type	Description
Using KernelShark	User Guide	https://lwn.net
Trace-cmd presentation	User Guide	https://lwn.net
KernelShark quick tutorial	Presentation	https://elinux.org

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

Universal Asynchronous Receiver/Transmitter

Stable: 17.06.2020 - 15:27 / Revision: 16.01.2020 - 13:38

Contents

1 Article purpose

This article provides the basic information needed to start using the Linux tool: **trace-cmd**[1] and host trace viewer **kernelshark**[2]
.

2 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
		trace-cmd[1] command interacts with the Ftrace tracer that is built inside the Linux kernel. It interfaces with the Ftrace specific files found in the debugfs file system						

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	under the tracing directory. **kernelshark**[2] 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.	✗	✗	✓	✗	✗	✗

Note: in case you want to trace at boot time, it is recommended to use ftrace.

3 Installing the trace and debug tool on your target board

3.1 Using the STM32MPU Embedded Software distribution

3.1.1 Distribution Package

Since Yocto Thud version, **trace-cmd** recipe is no more present in openembedded-core meta package and so is not installed by default in all STM32MPU Software Packages.

Following recipes are proposed to integrate trace-cmd in STM32MPU Embedded Software distribution. It has to be added in *layers/meta-st/meta-st-stm32mp/recipes-kernel*.

- Create *trace-cmd* directory:

```
PC $> mkdir layers/meta-st/meta-st-stm32mp/recipes-kernel/trace-cmd
PC $> cd layers/meta-st/meta-st-stm32mp/recipes-kernel/trace-cmd
```

- Create *trace-cmd.inc* file with the following content:

```
SRCREV = "57371aaa2f469d0ba15fd85276deca7bfdd7ce36"
PV = "2.6.2"

inherit pkgconfig

FILESEXTRAPATHS =. "${FILE_DIRNAME}/trace-cmd:"

SRC_URI = "git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git;
branch=trace-cmd-stable-v2.6"

S = "${WORKDIR}/git"
```

- Create *trace-cmd_git.bb* file with the following content:

```
SUMMARY = "User interface to Ftrace"
HOMEPAGE = "http://git.kernel.org/"
LICENSE = "GPLv2 & LGPLv2.1"

require trace-cmd.inc

LIC_FILES_CHKSUM = "file://COPYING;md5=751419260aa954499f7abaabaa882bbe \
                    file://trace-cmd.c;beginline=6;endline=8;
md5=2c22c965a649ddd7973d7913c5634a5e \
                    file://COPYING.LIB;md5=edb195fe538e4552c1f6ca0fd7bf4f0a \
                    file://trace-input.c;beginline=5;endline=8;
md5=3ec82f43bbe0cfb5951ff414ef4d44d0 \
"

EXTRA_OEMAKE = "\
    'prefix=${prefix}' \
    'bindir=${bindir}' \
    'man_dir=${mandir}' \
    'html_install=${datadir}/kernelshark/html' \
    'img_install=${datadir}/kernelshark/html/images' \
    \
    'bindir_relative=${@oe.path.relative(prefix, bindir)}' \
    'libdir=${libdir}' \
```

```
    \
    NO_PYTHON=1 \
"

do_compile_prepend() {
    # Make sure the recompile is OK
    rm -f ${B}/.*.d
}

do_install() {
        oe_runmake DESTDIR="${D}" install
}
```

- Add *trace-cmd* package install to the st-image-weston image:

```
PC $> echo 'IMAGE_INSTALL_append += "trace-cmd"' >> layers/meta-st/meta-st-openstlinux
/recipes-st/images/st-image-weston.bbappend
```

- Rebuilt the *st-image-weston* target:

```
PC $> cd <build_directory>
PC $> bitbake st-image-weston
```

When testing the new image on the target board, trace-cmd can be found in the following path:

```
Board $> which trace-cmd
/usr/bin/trace-cmd
```

However, an additional configuration of the Linux kernel must be performed to obtain the trace.

This is the configuration used for **ftrace**. Please refer to Ftrace install to complete the configuration of the Linux kernel.

This is the reason why you can not use this tool in Starter and Developer Packages context.

4 Getting started

trace-cmd inherits all available configurations from Ftrace.

It is possible to get filtered trace with both function and graph function mode.

It is also possible to trace specific prints done in the source file.

You first need to mount the tracefs:

```
Board $> mount -t tracefs nodev /sys/kernel/tracing
```

To find out which tracers are available, simply cat the available_tracers file in the tracing directory:

```
Board $> cat /sys/kernel/tracing/available_tracers
function_graph function nop
```

Note that you must be placed on the target board within a directory path where you have access rights for writing the generated trace file.

4.1 Making a trace using function tracer mode

The example below shows how to filter on a set of functions that contains uart:

- To get a trace on UART, you can connect to the target board with the new console (i.e. through ssh), and then type a simple command, like "*ls*"
- Then press Ctrl-C to stop the trace

```
Board $> trace-cmd record -p function -l *uart*
  plugin 'function'
Hit Ctrl^C to stop recording
^CCPU0 data recorded at offset=0x15d000
    4096 bytes in size
CPU1 data recorded at offset=0x15e000
    4096 bytes in size
```

See below for reading the trace.

4.2 Making a trace using graph function tracer mode

The example below shows how to filter on a set of graph functions that contains uart:

- To get trace on UART, you can connect to the target board with the new console (i.e. through ssh), and then type a simple command, like "*ls*"
- Then press Ctrl-C to stop the trace

```
Board $> trace-cmd record -p function_graph -l *uart*
  plugin 'function_graph'
Hit Ctrl^C to stop recording
^CCPU0 data recorded at offset=0x15d000
    4096 bytes in size
CPU1 data recorded at offset=0x15e000
    4096 bytes in size
```

See below for reading the trace.

4.3 Reading the trace

Command to read the trace:

```
Board $> trace-cmd report
```

By default, it reads the previously created file *trace.dat* at the position where the command is entered.

If you want to specify an other file name:

```
Board $> trace-cmd report -i mytrace.dat
or
Board $> trace-cmd report mytrace.dat
```

5 To go further

5.1 Visualizing trace using kernelshark

5.1.1 Host installation (Linux environment)

```
PC $> sudo apt-get install kernelshark
```

5.1.2 Opening and reading trace-cmd trace

kernelshark can read trace-cmd report and produce a graph and list view of the data. Please refer to kernelshark home page[2] for detail.

- Get the trace from the board. It is possible to get the trace directly from the SDCard under use, or remotely from the target board:

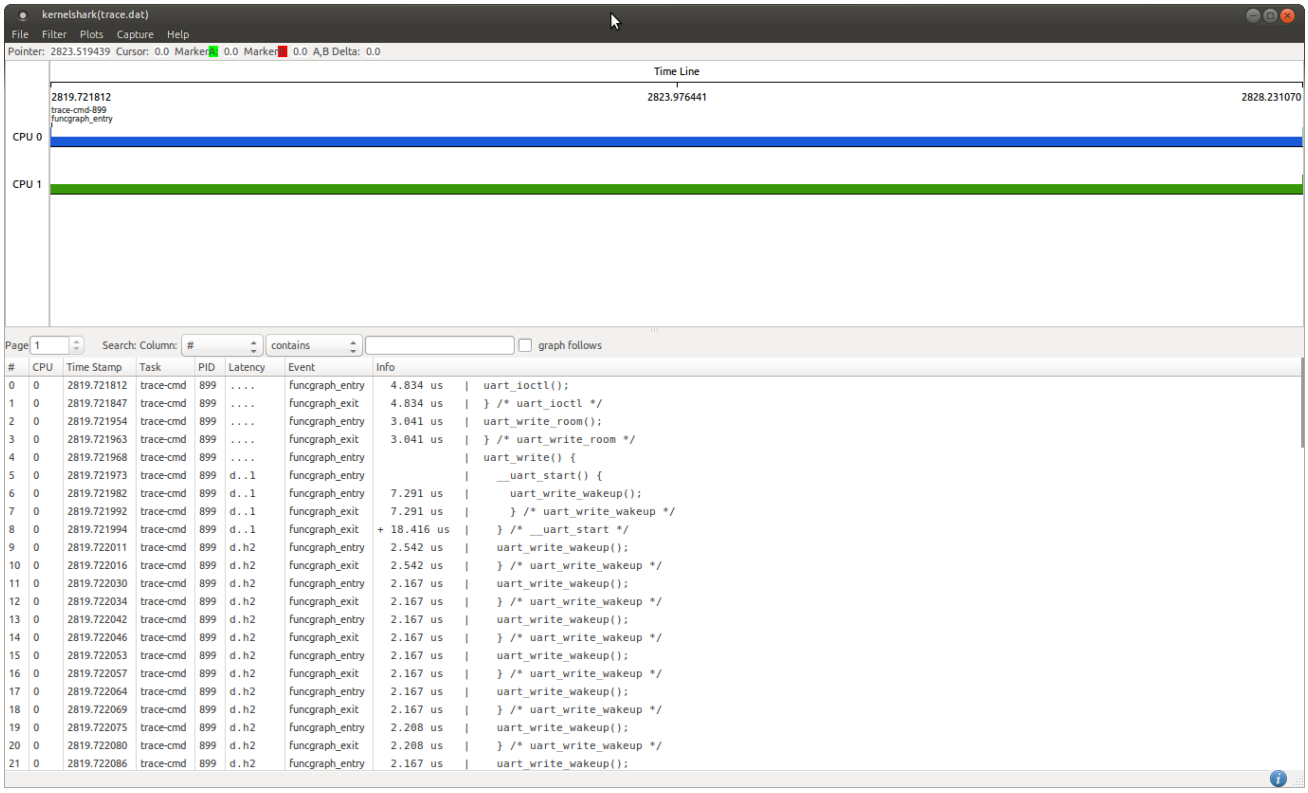
```
PC $> scp root@<board_ip_address>/<path_to>/trace.dat <path_on_host>
```

- Open trace with kernelshark

It can be done by opening kernelshark and then opening trace.dat file from the menu, or using the following command:

```
PC $> kernelshark <path_on_host>/trace.dat
```

Here is the example of a trace filtered with UART function_graph mode:



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	2819.721812	trace-cmd	899	....	funcgraph_entry	4.834 us | uart_ioctl();
1	0	2819.721847	trace-cmd	899	....	funcgraph_exit	4.834 us | } /* uart_ioctl */
2	0	2819.721954	trace-cmd	899	....	funcgraph_entry	3.041 us | uart_write_room();
3	0	2819.721963	trace-cmd	899	....	funcgraph_exit	3.041 us | } /* uart_write_room */
4	0	2819.721968	trace-cmd	899	....	funcgraph_entry	| uart_write() {
5	0	2819.721973	trace-cmd	899	d..1	funcgraph_entry	| __uart_start() {
6	0	2819.721982	trace-cmd	899	d..1	funcgraph_entry	7.291 us | uart_write_wakeup();
7	0	2819.721992	trace-cmd	899	d..1	funcgraph_exit	7.291 us | } /* uart_write_wakeup */
8	0	2819.721994	trace-cmd	899	d..1	funcgraph_exit	+ 18.416 us | } /* __uart_start */
9	0	2819.722011	trace-cmd	899	d.h2	funcgraph_entry	2.542 us | uart_write_wakeup();
10	0	2819.722016	trace-cmd	899	d.h2	funcgraph_exit	2.542 us | } /* uart_write_wakeup */
11	0	2819.722030	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
12	0	2819.722034	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
13	0	2819.722042	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
14	0	2819.722046	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
15	0	2819.722053	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
16	0	2819.722057	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
17	0	2819.722064	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
18	0	2819.722069	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
19	0	2819.722075	trace-cmd	899	d.h2	funcgraph_entry	2.208 us | uart_write_wakeup();
20	0	2819.722080	trace-cmd	899	d.h2	funcgraph_exit	2.208 us | } /* uart_write_wakeup */
21	0	2819.722086	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();

6 References

- [1.01.1] https://lwn.net/Articles/410200/
- [2.02.12.2] http://rostedt.homelinux.com/kernelshark

- Useful external links

Document link	Document Type	Description
Using KernelShark	User Guide	https://lwn.net
Trace-cmd presentation	User Guide	https://lwn.net
KernelShark quick tutorial	Presentation	https://elinux.org

Linux® is a registered trademark of Linus Torvalds.

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

Universal Asynchronous Receiver/Transmitter

Subcategories

This category has the following 2 subcategories, out of 2 total.

1

- Linux tracing tools (7 P)

A

- Android tracing tools (1 P)

Pages in category "Tracing tools"

The following 4 pages are in this category, out of 4 total.

H

- How to debug OP-TEE
- How to debug TF-A BL2
- How to debug TF-A SP-MIN

U

- U-Boot - How to debug

Stable: 30.03.2021 - 10:18 / Revision: 16.02.2021 - 15:49

Contents

1 Article purpose

This article provides the basic information needed to start using the Linux tool: **trace-cmd**[1] and host trace viewer **kernelshark**[2]
.

2 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
		trace-cmd[1] command interacts with the Ftrace tracer that is built inside the Linux kernel. It interfaces with the Ftrace specific files found in the debugfs file system						

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	under the tracing directory. **kernelshark**[2] 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.	✗	✗	✓	✗	✗	✗

Note: in case you want to trace at boot time, it is recommended to use ftrace.

3 Installing the trace and debug tool on your target board

3.1 Using the STM32MPU Embedded Software distribution

3.1.1 Distribution Package

Since Yocto Thud version, **trace-cmd** recipe is no more present in openembedded-core meta package and so is not installed by default in all STM32MPU Software Packages.

Following recipes are proposed to integrate trace-cmd in STM32MPU Embedded Software distribution. It has to be added in *layers/meta-st/meta-st-stm32mp/recipes-kernel*.

- Create *trace-cmd* directory:

```
PC $> mkdir layers/meta-st/meta-st-stm32mp/recipes-kernel/trace-cmd
PC $> cd layers/meta-st/meta-st-stm32mp/recipes-kernel/trace-cmd
```

- Create *trace-cmd.inc* file with the following content:

```
SRCREV = "57371aaa2f469d0ba15fd85276deca7bfdd7ce36"
PV = "2.6.2"

inherit pkgconfig

FILESEXTRAPATHS =. "${FILE_DIRNAME}/trace-cmd:"

SRC_URI = "git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git;
branch=trace-cmd-stable-v2.6"

S = "${WORKDIR}/git"
```

- Create *trace-cmd_git.bb* file with the following content:

```
SUMMARY = "User interface to Ftrace"
HOMEPAGE = "http://git.kernel.org/"
LICENSE = "GPLv2 & LGPLv2.1"

require trace-cmd.inc

LIC_FILES_CHKSUM = "file://COPYING;md5=751419260aa954499f7abaabaa882bbe \
                    file://trace-cmd.c;beginline=6;endline=8;
md5=2c22c965a649ddd7973d7913c5634a5e \
                    file://COPYING.LIB;md5=edb195fe538e4552c1f6ca0fd7bf4f0a \
                    file://trace-input.c;beginline=5;endline=8;
md5=3ec82f43bbe0cfb5951ff414ef4d44d0 \
"

EXTRA_OEMAKE = "\
    'prefix=${prefix}' \
    'bindir=${bindir}' \
    'man_dir=${mandir}' \
    'html_install=${datadir}/kernelshark/html' \
    'img_install=${datadir}/kernelshark/html/images' \
    \
    'bindir_relative=${@oe.path.relative(prefix, bindir)}' \
    'libdir=${libdir}' \
```

```
    \
    NO_PYTHON=1 \
"

do_compile_prepend() {
    # Make sure the recompile is OK
    rm -f ${B}/.*.d
}

do_install() {
        oe_runmake DESTDIR="${D}" install
}
```

- Add *trace-cmd* package install to the st-image-weston image:

```
PC $> echo 'IMAGE_INSTALL_append += "trace-cmd"' >> layers/meta-st/meta-st-openstlinux
/recipes-st/images/st-image-weston.bbappend
```

- Rebuilt the *st-image-weston* target:

```
PC $> cd <build_directory>
PC $> bitbake st-image-weston
```

When testing the new image on the target board, trace-cmd can be found in the following path:

```
Board $> which trace-cmd
/usr/bin/trace-cmd
```

However, an additional configuration of the Linux kernel must be performed to obtain the trace.

This is the configuration used for **ftrace**. Please refer to Ftrace install to complete the configuration of the Linux kernel.

This is the reason why you can not use this tool in Starter and Developer Packages context.

4 Getting started

trace-cmd inherits all available configurations from Ftrace.

It is possible to get filtered trace with both function and graph function mode.

It is also possible to trace specific prints done in the source file.

You first need to mount the tracefs:

```
Board $> mount -t tracefs nodev /sys/kernel/tracing
```

To find out which tracers are available, simply cat the available_tracers file in the tracing directory:

```
Board $> cat /sys/kernel/tracing/available_tracers
function_graph function nop
```

Note that you must be placed on the target board within a directory path where you have access rights for writing the generated trace file.

4.1 Making a trace using function tracer mode

The example below shows how to filter on a set of functions that contains uart:

- To get a trace on UART, you can connect to the target board with the new console (i.e. through ssh), and then type a simple command, like "*ls*"
- Then press Ctrl-C to stop the trace

```
Board $> trace-cmd record -p function -l *uart*
  plugin 'function'
Hit Ctrl^C to stop recording
^CCPU0 data recorded at offset=0x15d000
    4096 bytes in size
CPU1 data recorded at offset=0x15e000
    4096 bytes in size
```

See below for reading the trace.

4.2 Making a trace using graph function tracer mode

The example below shows how to filter on a set of graph functions that contains uart:

- To get trace on UART, you can connect to the target board with the new console (i.e. through ssh), and then type a simple command, like "*ls*"
- Then press Ctrl-C to stop the trace

```
Board $> trace-cmd record -p function_graph -l *uart*
  plugin 'function_graph'
Hit Ctrl^C to stop recording
^CCPU0 data recorded at offset=0x15d000
    4096 bytes in size
CPU1 data recorded at offset=0x15e000
    4096 bytes in size
```

See below for reading the trace.

4.3 Reading the trace

Command to read the trace:

```
Board $> trace-cmd report
```

By default, it reads the previously created file *trace.dat* at the position where the command is entered.

If you want to specify an other file name:

```
Board $> trace-cmd report -i mytrace.dat
or
Board $> trace-cmd report mytrace.dat
```

5 To go further

5.1 Visualizing trace using kernelshark

5.1.1 Host installation (Linux environment)

```
PC $> sudo apt-get install kernelshark
```

5.1.2 Opening and reading trace-cmd trace

kernelshark can read trace-cmd report and produce a graph and list view of the data. Please refer to kernelshark home page[2] for detail.

- Get the trace from the board. It is possible to get the trace directly from the SDCard under use, or remotely from the target board:

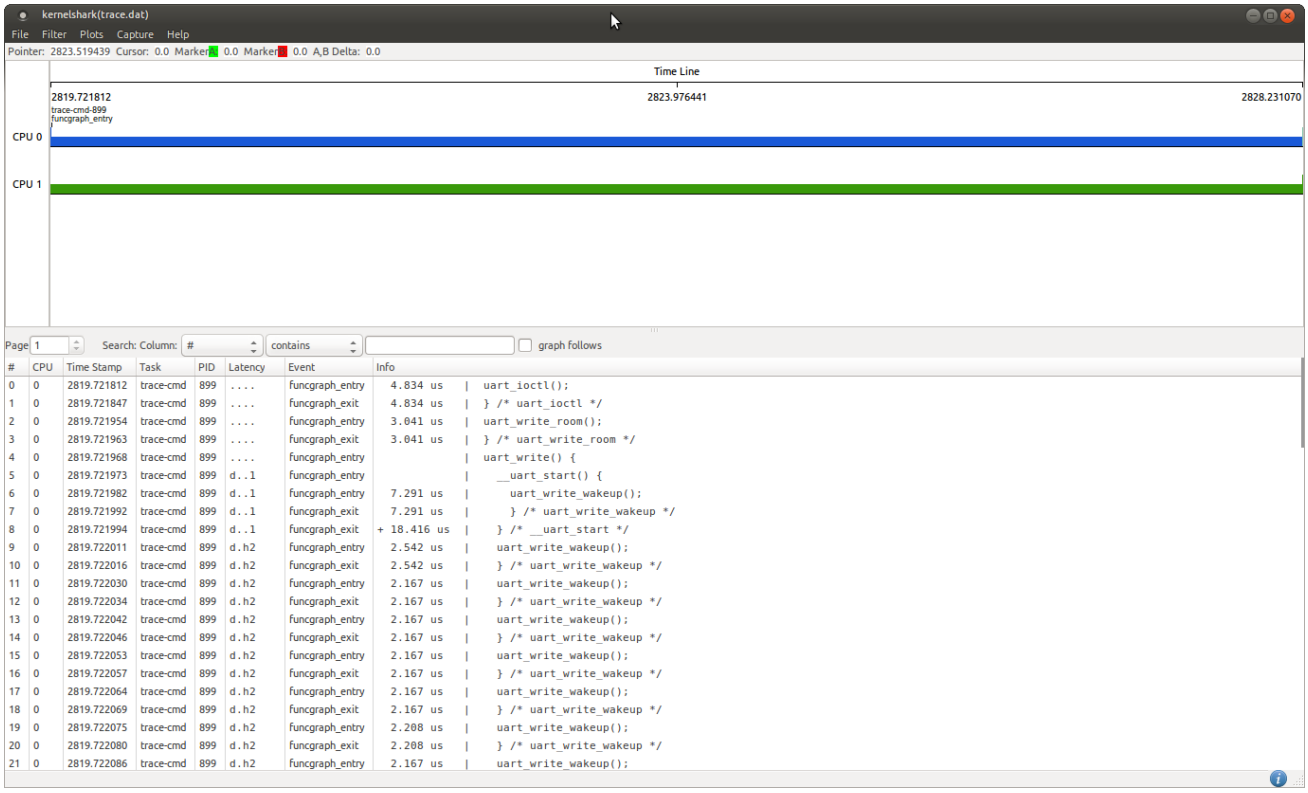
```
PC $> scp root@<board_ip_address>/<path_to>/trace.dat <path_on_host>
```

- Open trace with kernelshark

It can be done by opening kernelshark and then opening trace.dat file from the menu, or using the following command:

```
PC $> kernelshark <path_on_host>/trace.dat
```

Here is the example of a trace filtered with UART function_graph mode:



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
0	0	2819.721812	trace-cmd	899	....	funcgraph_entry	4.834 us | uart_ioctl();
1	0	2819.721847	trace-cmd	899	....	funcgraph_exit	4.834 us | } /* uart_ioctl */
2	0	2819.721954	trace-cmd	899	....	funcgraph_entry	3.041 us | uart_write_room();
3	0	2819.721963	trace-cmd	899	....	funcgraph_exit	3.041 us | } /* uart_write_room */
4	0	2819.721968	trace-cmd	899	....	funcgraph_entry	| uart_write() {
5	0	2819.721973	trace-cmd	899	d..1	funcgraph_entry	| __uart_start() {
6	0	2819.721982	trace-cmd	899	d..1	funcgraph_entry	7.291 us | uart_write_wakeup();
7	0	2819.721992	trace-cmd	899	d..1	funcgraph_exit	7.291 us | } /* uart_write_wakeup */
8	0	2819.721994	trace-cmd	899	d..1	funcgraph_exit	+ 18.416 us | } /* __uart_start */
9	0	2819.722011	trace-cmd	899	d.h2	funcgraph_entry	2.542 us | uart_write_wakeup();
10	0	2819.722016	trace-cmd	899	d.h2	funcgraph_exit	2.542 us | } /* uart_write_wakeup */
11	0	2819.722030	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
12	0	2819.722034	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
13	0	2819.722042	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
14	0	2819.722046	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
15	0	2819.722053	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
16	0	2819.722057	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
17	0	2819.722064	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();
18	0	2819.722069	trace-cmd	899	d.h2	funcgraph_exit	2.167 us | } /* uart_write_wakeup */
19	0	2819.722075	trace-cmd	899	d.h2	funcgraph_entry	2.208 us | uart_write_wakeup();
20	0	2819.722080	trace-cmd	899	d.h2	funcgraph_exit	2.208 us | } /* uart_write_wakeup */
21	0	2819.722086	trace-cmd	899	d.h2	funcgraph_entry	2.167 us | uart_write_wakeup();

6 References

- [1.01.1] https://lwn.net/Articles/410200/
- [2.02.12.2] http://rostedt.homelinux.com/kernelshark

- Useful external links

Document link	Document Type	Description
Using KernelShark	User Guide	https://lwn.net
Trace-cmd presentation	User Guide	https://lwn.net
KernelShark quick tutorial	Presentation	https://elinux.org

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

Universal Asynchronous Receiver/Transmitter