NXP Semiconductors
Application Note

Document Number: AN5129
Rev. 11.3.1, 5/2018

Collect Linux Hardware Trace for ARMv8 User Space and Kernel Space Applications

Contents

1 Introduction

This document describes the Linux probe-less trace component and presents multiple execution flows of it. The objective of this component is to encapsulate the trace configurator and probe into one small and cross-compiled component that gets uploaded on target machine. Its main use is to collect trace of a program that crashes without known reasons.

NOTE

This feature will be delivered as an archive or part of the ARMv8 CodeWarrior.

2 Overview

The component integrates the complete flow for a trace collection session under Linux OS based on an ARMv8 processor. It is based on ARMv8 platform configurator and probe controlled by xml files.

The main advantages ARMv8 standalone tracing tool are:

- Size – contains only what is needed
- Speed – all services are hosted on target machine and there are no delays caused by communication between multiple workstations or languages

- Nonintrusive – no need to instrument the target application
- Easy to use – collects all required / available information for decoding
- Simple API – can be easily integrated into any testing framework
- Data-driven – the configurator and probe can be easily tuned up and scaled to user needs

3 Execution Flow

The execution flow is described below:

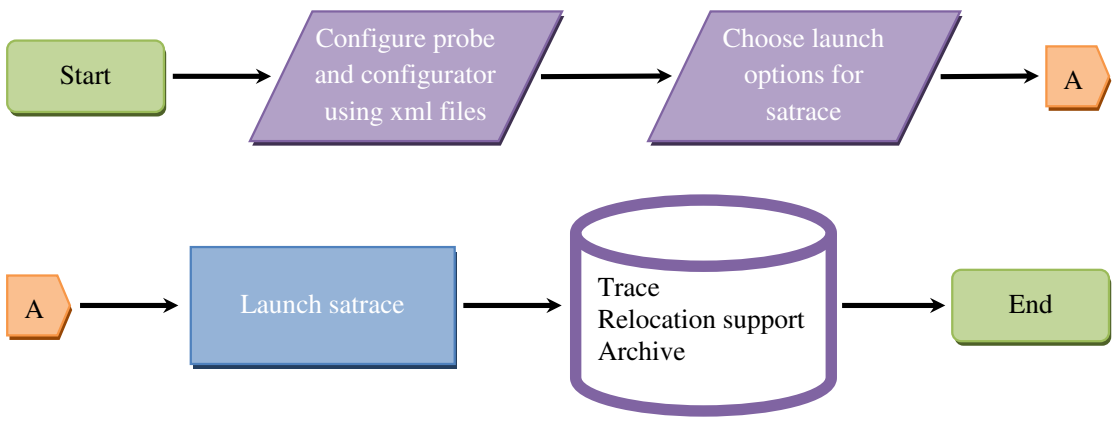

Figure 1. Flowchart for ARMv8 standalone Linux trace

4 Component Details

This section includes:

- API
- User space trace
- Kernel space trace

4.1 API

The Linux trace mechanism is delivered as an independent product. It runs on ARMv8 Linux machine. The root folder of the delivered product has the following file structure:

```
bin:  (All binary files needed for a trace session)
    ls.linux.satrace
```

```
config:  (Configurator and probe configuration file)
     Platform configuration files for all target supported.

lib:  (Dependencies)
   libls.linux.satrace.lib.so      libsae.ls.target.access.so
   libls.linux.satrace.lib.so.1.0  libsae.ls.target.access.so.1.0
   libls.target.agent.so           libsae.ls.tc.config.so
   libls.target.agent.so.1.0       libsae.ls.tc2.config.so
   libsae.ls.common.so             libsae.ls.tc2.config.so.1.0
   libsae.ls.common.so.1.0
```

NOTE

Use the apropriate satrace agent from `{CW_INSTALL_DIR}\CW_ARMv8\ARMv8\sa_ls\`:

- `linux.armv8-sdk1.8-ear6.satrace` for linux image with little endian, on all targets, starting with ARMv8 SDK ear6
- `linux.armv8.satrace` for linux image with little endian, on all targets, before ARMv8 SDK ear6
- `linux.armv8-ls1043-be.satrace` for linux image with big endian, on LS1043

Below are the listed options with a short description.

Table 1. User space options

Command	Description
-A [--archive-file] arg (=[app_name].cwzsa)	Archive path
-b [--backtrace]	Shows backtrace on SEGFAULT
[app_name]	Name of the traced application.

Table 2. Common options

Command	Description
-T [--multithreading]	Enables multithreading support
-p [--pid] PID	Attach to a process giving a PID
--vmid vmid	Virtual machine ID
--start-trace address	Start tracepoint
--stop-trace address	Stop tracepoint
--include-range range	Include range
--exclude-range range	Exclude range

Table 3. Kernel space options

Command	Description
-K [--kernel] path	Archive path.
-i [--kernel-image] path	vmlinux image compiled with debugging symbols.

Table 4. System trace options

Command	Description
-S [--system] arg (=[app_name].scwzsa)	Archive path.
-i [--kernel-image] path	vmlinux image compiled with debugging symbols.
-b [--backtrace]	Shows backtrace on SEGFAULT.

Table 5. General options

Command	Description
-v [--verbose]	Verbose mode.
-V [--version]	Product version
-h [--help]	Displays this help message
-c [--config-file] path	Configuration file
--soc arg (=LS1088A)	Name of the SoC

4.2 User space trace

The relocation file contains a list of libraries linked with the traced application with their load addresses. This list will contain also injected libraries through `LD_PRELOAD` variable.

The trace file incorporates the raw trace collected by **Debug Trace Complex** (DTC) Probe from the location specified in the probe configuration file.

The `-A` option is the most verbose. It archives the applications, all its dependencies (shared libraries), trace file, the configuration file and relocation support. This is the default option. Its use increases the time and file-system space required for archiving. The main advantage is the generated `*.cwzsa` file. It's an archive file that can be imported and fully decoded using ARMv8 decoder or ARMv8 CodeWarrior.

The `-p` option offers the attach possibility. The user will start a process (for example, daemon or server) and using its process ID (PID) it will attach satrace to it in order to get a detailed overview of its performance.

The `-T` option enables multithreading support. There is also the possibility to trace between addresses or ranges.

[range] - An interval specified using one of the following formats:

- 0x2000-0x3000 - Address range [0x2000, 0x3000]
- libpthread - Executable code from libpthread.so
- init_linuxrc - Address range based on kernel function name; Covers all instructions from init_linuxrc
- init_linuxrc-init_linuxrc+8 - Includes/Excludes first 8 bytes from init_linuxrc
- ipv6.ko - Includes/Excludes 'ipv6' kernel module

[address] - An address specified using one of the following formats :

- 0x2000 - Hex address
- libpthread+200 - Offset from a shared libary (libpthread.so)
- init_linuxrc - Address based on kernel function name
- init_linuxrc+8 - Kernel function offset
- ipv6.ko - Kernel module offset

--vmid argument is compatible only with address range filters.

The –v option will generate a more detailed output at standard output about all the execution steps. Usually the SEGFAULT signal is the main reason of the C/C++ applications crash. Thus, a backtrace on SEGFAULT is welcome for an embedded world where each byte of file system matters. The –b option will dump all known stack frames without having support from a debugger. Before using this option the user must ensure that the traced application has been compiled with debug information (-g for GCC) and extra code for exception propagation (**-funwind-tables** for GCC) and all symbols are added to the dynamic symbol table (**-rdynamic** for GCC).

Before starting any trace session the user must ensure that Linux Kernel has been compiled with enabled `PID_IN_CONTEXTIDR` configuration option.

Create a small program that computes the sum of elements from `0` to `num` and crashes due to a segmentation fault.

```
#include <iostream>

    class SegFaultTest
    {
    public:
        SegFaultTest() {
            sum(5);
            function1();
        }

    private:
        void function1() { function2(); }

        void function2() { function3(); }

        void function3() { function4(); }

        void function4() { crash(); }

        void crash() {
            char * p = NULL;
            *p = 0;
        }

        int sum(int n) {
            if (n <= 0) {
                return n;
            }

            return n + sum(n - 1);
        }
    };

    int main(int argc, char ** argv)
    {
        SegFaultTest * f = new SegFaultTest();
            return 0;
    }
```

After saving the above program into `segfault.cpp` file, you should compile it with debugging symbols:

```
    g++ -g3 -funwind-tables -rdynamic segfault.cpp -o segfault
```

Figure out which line caused the crash. Launch the `segfault` executable using `ls.linux.satrace`.

```
    root@ls2085aqds:~# ./linux.armv8.satrace/bin/ls.linux.satrace -b ./segfault
    User space trace
    Application  : `./segfault`
    Arguments    :
    Relocation file  : `/home/root/segfault.rlog`
    Trace file  : `/home/root/segfault.dat`
    Starting `./segfault`
    Signal 11 (Segmentation fault), address is 0
        (1) ./segfault : SegFaultTest::crash()+0xf [0x8bc4]
```

```
        (2) ./segfault : SegFaultTest::function4()+0xd [0x8bae]
        (3) ./segfault : SegFaultTest::function3()+0xd [0x8b9a]
        (4) ./segfault : SegFaultTest::function2()+0xd [0x8b86]
        (5) ./segfault : SegFaultTest::function1()+0xd [0x8b72]
        (6) ./segfault : SegFaultTest::SegFaultTest()+0x15 [0x8b5a]
        (7) ./segfault : main+0x19 [0x8ad2]
        (8) /lib/libc.so.6 : __libc_start_main+0x110 [0x76c912b8]
    User application terminated because it didn't catch signal number : 11 (Segmentation
fault)
    Master process
    Collecting trace ...
    Archive file  : `/home/root/segfault.cwzsa`
    Creating archive ....
    Archiving /home/root/segfault.rlog
    Archiving /home/root/segfault
    Archiving /lib/librt-2.18-2013.10.so
    Archiving /lib/libdl-2.18-2013.10.so
    Archiving /lib/libpthread-2.18-2013.10.so
    Archiving /lib/libc-2.18-2013.10.so
    Archiving /lib/libm-2.18-2013.10.so
    Archiving /lib/ld-2.18-2013.10.so
    Archiving ./linux.armv8.satrace/config/PlatformConfig.xml
    Archiving /home/root/segfault.dat
```

For LSDK, run the following command:

```
./linux.armv8-sdk1.8-ear6.satrace/bin/ls.linux.satrace -b -v ./segfault
```

The `satrace` collects trace and archives all dependencies into `/home/root/segfault.cwzsa` archive. You can open the generated archive using ARMv8 CodeWarrior drag-and-drop action. Previous action will trigger the import wizard. After importing it, the user will be able to access all Software Analysis features using Analysis Results view.

4.3 Kernel space trace

The same component can be used for probe-less kernel space tracing. For this type of trace, Kernel space options are used. There are two options: `-K` and `-i`. The first option is used to start a kernel space trace session and will specify the name of the generated archive. It is the kernel space equivalent for `-A` option. The `-i` option is optional and will point to the vmlinux image of the system. This option will be useful only when the kernel image contains debug information, otherwise `-K` option will be more convenient to use.

Run the `satrace` with `-K` and `-i` options. After few seconds, send a `SEGINT` signal by pressing `CTRL+C` on your keyboard.

```
root@ls2085aqds:~# ./linux.armv8.satrace/bin/ls.linux.satrace -K kernelTest -i ~/vmlinux
Kernel space trace
Archive : `kernelTest.kcwzsa`
Hit CTRL+C to stop trace.
Collecting trace ...
Kernel image  : `/home/root/vmlinux`
Archive file  : `kernelTest.kcwzsa`
Creating archive ....
Archiving /home/root/vmlinux
Archiving ./linux.armv8.satrace/config/PlatformConfig.xml
Archiving kernelTest.dat
```

For LSDK, run the following command:

```
./linux.armv8-sdk1.8-ear6.satrace/bin/ls.linux.satrace -K kernelTest -i ~/vmlinux
```

The generated archive can be open by an ARMv8 CodeWarrior with a drag-and-drop action. This action will trigger the import wizard.

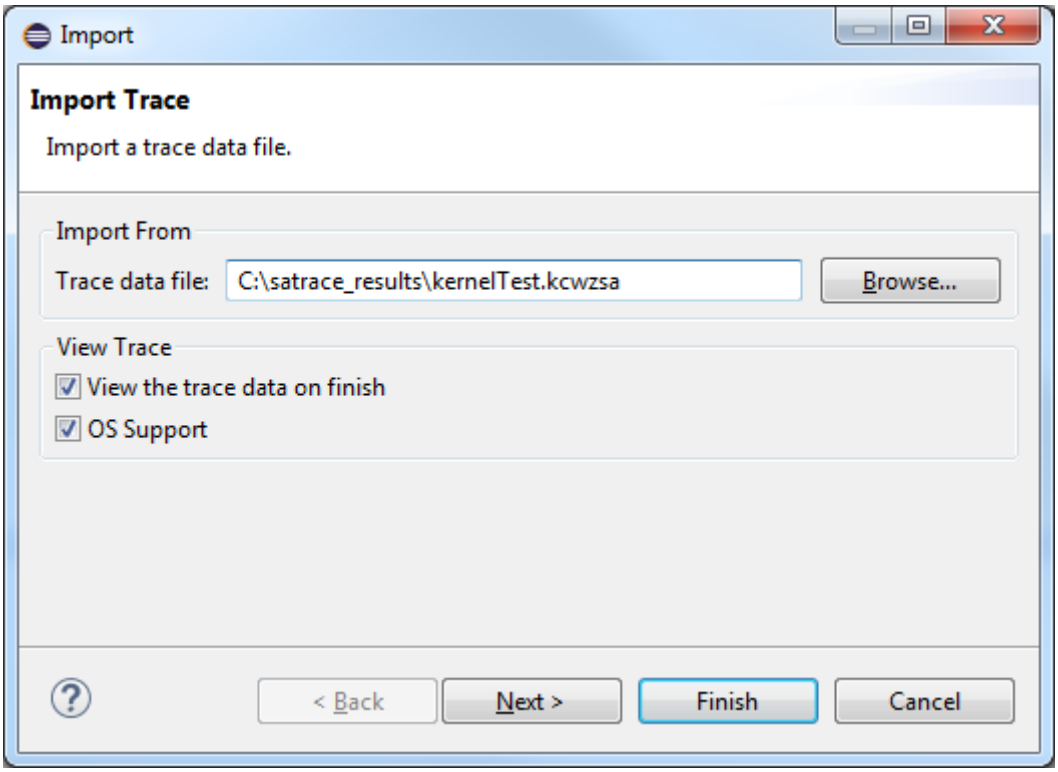

Figure 2. Import wizard

5 Conclusions

The `ls.linux.satrace` component can be used by any Linux user who wants to know the cause of the crash, or wants to follow the function calls or needs to evaluate his software without any hardware probe. After saving the trace file into an archive that contains all needed files for a full decoding, can be viewed in CodeWarrior. The user benefits from all advantages offered by CodeWarrior ARMv8. You have the profiling data code coverage, call tree, and performance analysis as well.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number AN5129
Revision 11.3.1, 5/2018