Live Tracing support and integrating Transports in an API

Google Summer of Code Program 2017 Project Proposal

Vivek Kukreja
vivekkukreja5@gmail.com
Indian Institue of Technology(ISM)
Dhanbad, India

Project Abstract

RTEMS provides a software tracing framework to trace system events and applications. These traces are generated in the native binary format. As a participant in GSOC 2016, I modified the tracing framework to generate traces in the Common Trace Format (CTF). This allows developers to visualize and analyse the traces using third party tools such as Babeltrace. I also added Ethernet transport for moving traces to host using the rtems-libbsd drivers.

This year under GSoC I am proposing modifications to the tracing framework to support live tracing. During the first and second phase of the project, I will implement a generic framework to support live tracing over multiple transport mechanisms. I will implement and demonstrate this functionality, which will internally employ a unified extensible API for configuring various transports automatically. Using the API to implement support for other transport mechanisms in future will be possible. Further, I will do performance benchmarking and investigate the opportunity to improve the performance of the tracing framework in such a way that the holes in the generated trace can be reduced and impact on performance of the applications minimized.

Further, I will focus on reducing the manual effort needed in instrumentation of applications. To generate CTF metadata file that describes the format of the traces, we need the signatures of the functions to be traced and the sizes of the function parameters. In the current implementation the user must provide the function signatures and parameter types in a configuration file. By integrating suitable libDWARF functionalities, we can find the signature of the function to be traced and its parameters types and sizes by parsing the ELF binary. With this improvement, the developer will only need to provide name of the function to be traced.

Project Description

State of the art

RTEMS provides a software tracing framework to trace system events and software applications. To trace software applications, developers create a .ini configuration file and provide signature of

the functions to be traced. The trace linker tool then instruments the binary ELF code during linking. It produces a trace containing values of the function parameters and entry and exit timestamps, each time the function is called. The trace is generated in the native binary format. RTEMS also supports tracing for system events like context switches. The traces from various sources are combined in an SMP safe fashion, and saved on the device.

As a participant in GSOC 2016, I modified the tracing framework in RTEMS to generate traces in the Common Trace Format (CTF) instead of the native binary format. CTF is a widely supported trace format. Traces generated in CTF can be visualized and analysed by third party tools such as Babeltrace. Further, I used libbsd drivers to move the traces over Ethernet to the host machine. I developed scripts for Babeltrace and tested the functionality. This code is being prepared to be merged to the mainline.

In GSoC 2017, I want to continue my work on the tracing framework. I have proposed three modifications in the following sections.

First Phase: Transports and unified API

I will begin by implementing Serial and USB based transport mechanisms for moving traces to host. For setting up serial transport, one approach is to redirect the output stream (trace buffer) to a serial port on the RTEMS machine using native C and shell functionalities in RTEMS. This can be used to transport the traces to host by enabling serial communication in qemu or other simulators.

In my previous effort I used rtems-libbsd drivers to setup an Ethernet based transport mechanism using NFS communication. I tested the approach using qemu for arm/xilinx_zynq_a9_qemu BSP. For enabling serial over USB transport, I will study the implementation and interface of the USB driver, referring the examples from rtems-libbsd repository to build a suitable transport mechanism.

Further, there is a need to integrate the available transport mechanisms and create a holistic Application Programming Interface, consistent for users across various BSP's and architecture's. I will examine the above transport mechanisms and consult with my mentors to create a minimal interface for configuring each transport, consisting of simple functions like init, read, write and close independent of the underlying transport used. This API will be exposed to the user for bulk tracing. In the next phase I will implement Live-tracing where this API will be used internally so that the transport mechanism can be abstracted from the Capture Engine or application. These live-traces will be buffered and sent in blocks to optimise performance.

This API can be extended to implement other transport mechanisms too. I will create simple examples to demonstrate the use of this API by enabling different transports in an application.

Second Phase: Live Tracing

In this phase I will create support for live-tracing for the Trace linker and Capture Engine. Internally the live tracing module will employ the new unified API for transports to move live

traces to host in a timely fashion. I will investigate viable implementations of this functionality and consult with my mentors regarding their feasibility and performance.

I will work on optimizing the current implementation of trace generation and collection. Currently under function tracing, a single buffer is used by the instrumented functions for buffering traces in an SMP safe manner employing locks. Thus every time user reads the trace, program execution must be paused before the unsafe buffer can be accessed. I will try to employ a ring buffer in place of the current bulk buffer to avoid using locks. I will work with mentors to come up with other ideas to improve performance in this area.

Once we have a qualified approach for live tracing of applications, we can employ a similar strategy for Capture Engine where a new record event can be created to dump the trace to an alternate buffer periodically or when time permits.

Further, I will do performance benchmarking and optimise the live-tracing module to reduce buffer-overflows and holes in the trace generated. I will test and demonstrate this functionality using examples where a user can configure various transports with live tracing.

Third Phase: Integrating libDWARF functionalities

In this phase, I will try to reduce the manual effort needed from the developer to use the Tracing Framework in RTEMS.

The CTF metadata file describes the format of the CTF traces. Function tracing in RTEMS allows developers to generate traces for entry and exit into functions, along with the values of the function parameters. To generate CTF metadata, the sizes of the function parameters should be known at link time. Currently, the developer must provide the signature of the function to be traced along with the sizes of its various parameters in the .ini configuration file. This process can be automated by extracting this information from the ELF binary of the application.

LibDWARF provides the functionality to parse the ELF binary and perform various operations on its sections. Porting a full-fledged version of libDWARF is not a feasible task for a single milestone in GSoC since several other features need to be addressed, so I will investigate the specific functionalities of libDWARF that are well-suited to the above use-case and come up with an approach that makes the current function tracing system more generic to use.

Project Deliverables

GSoC 2017 Timeline: https://developers.google.com/open-source/gsoc/timeline

May 30 (coding begins)

Before the coding phase begins, I will setup RTEMS for few arch/BSP's like sparc/erc32 and arm/xilinx_zynq_a9_qemu bsp along with the various simulation tools required like gdb and qemu. I will resolve any previous issues with trace related examples in the current repo and

simultaneously refine and port my previous code for CTF tracing and Ethernet based transport. By this time, I will also complete the planning phase and create the requirements specification document for the first milestone.

June 26-30 (First Evaluation)

In the first phase I will create more transport mechanisms for traces generated by applications and the Capture Engine. I will implement USB and serial transports as discussed above. Additionally, I will develop a standardised adapter interface to allow easy porting of various transport mechanisms for internal use with live tracing or to directly configure transports in an application. I will create tracing examples that demonstrate this API by configuring and using different transports. I will deliver documentation to explain the implementation and usage of this functionality along with simple examples.

July 24-28 (Second Evaluation)

In this phase I will make enhancements to the Capture Engine and Trace Linker to support simultaneous read/write of trace buffers for enabling live-tracing support. Some Capture CLI and native RTEMS CLI functionalities will be modified and new ones added to support and configure live tracing and the transports. The new unified Transport API will be used by the live tracing module internally, for periodically moving these live traces to host. I will create simple examples and develop unit-tests for this functionality. I will test the results for buffer overflow problem and timing issues, against the native tracing support that requires pausing of execution to print traces. I will refer with previous contributors and other experts in case of any unforeseen issues in this endeavour.

August 29-September 5 (Third Evaluation)

Before the third phase, I will document my previous changes to both function tracing and capture tracing. In this phase I will work on modifying the trace-linking process to make it more generic and automated. In this phase I will concentrate on CTF tracing of applications using rtems-tld. As discussed above, the types and sizes of the parameters of the functions to be traced must be defined in the user configuration file beforehand.

In this phase I will study libDWARF functionalities that can be used to automatically fetch specific data types and sizes from an ELF section of a compiled C file. I will refer with my mentors and contributors from libDWARF to understand how libDWARF parses the ELF data section and fetches function signatures and datatypes. I will modify the current function tracing framework to come up with a generic solution for function tracing with CTF which is easier for the user to understand, configure and replicate.

September 6(Final Results Announced)

I will continue with code-cleanup and will consult my mentors for further changes in the code or documentation. I will prepare the code for merging to the main repository.

Post GSOC

I will work on merging my deliverables and will continue supporting the RTEMS tracing framework in the future. I look forward to a continued association with RTEMS and would like to work on other components too.

Proposed Schedule

March 20 - April 3 (Application Period)

- I will discuss the project with my mentors and study the various dependencies for each phase.
- I will create a Project proposal and incorporate suggestions from the experts.
- I will try to clarify the project requirements and scope, and estimate the effort.

April 3 - April 24 (Acceptance Waiting Period)

- During this phase, I will setup RTEMS for arm/xilinx_aynq_a9_qemu and run the tracing examples.
- I will understand the requirements of RTEMS developers and identify any limitations for implementing live tracing.
- I will explore and resolve any issues I might identify in the current repository.
- I will refine my previous changes for CTF tracing and prepare for merging to the main repository.

May 4 - May 30 (Community Bonding Period)

- I will continue coordinating with my mentors and produce detailed Requirements Specification and Design documents for each phase.
- I will explore the project and consult with my mentors if there is a need to expand or modify the scope of each deliverable.
- I will start studying for the first phase and try to produce viable tests for the each milestone beforehand.

May 30 - June 30 (First Phase)

- In this phase, I will start by merging the Ethernet transport mechanism for Capture Engine and Trace Linker I set up in GSoC 2016.
- I will start by setting up serial transport using redirection to serial port and USB transport will be addressed using rtems-libbsd drivers.
- Following preliminary testing of these transports, I will start work on integrating these mechanisms under a holistic API for transporting traces that will be tested across different BSP's for performance and consistency.
- I will test this API across multiple arch/bsp pairs and document the extensible interface for future developers.

July 1 - July 28 (Second Phase)

- First, I will concentrate on supporting live-tracing under the trace-linker by solving the unsafe buffer problem. Live tracing will internally use the above Transport API.
- Further, I will create suitable new tests or reuse old ones for testing the live tracing functionality over available transports.
- Once I have the results, I will consult with my mentors and start working on live-tracing of User Extensions by the Capture Engine on the same lines.
- I will create multiple tests and do performance benchmarking of live tracing over different transports.

July 1 - July 28 (Third Phase)

- I will begin by testing my previous changes for CTF tracing. I will explore the application tracing framework and resolve any issues in the implementation or relevant examples.
- I will study the libDWARF implementation to understand how ELF's are parsed, and compare it with the native approach of RTEMS for reading and manipulating ELF sections.
- I will refer with my mentor and other libDWARF collaborators to port the required ELF reading functionalities to the function tracing framework.
- I will implement an easier generic approach for function tracing.
- Upon approval by the mentors, I will start testing and documenting the new approach for potential users.

Future Improvements

I will explore the tracing framework to further improve the performance and reduce the overhead of live tracing. I will create documentation for each deliverable and continue consulting my mentors to fix any bugs that might be discovered in my implementation.

Continued Involvement

After this project, I will continue supporting and maintaining the RTEMS tracing framework and will help developers with any issues that may arise. I look forward to working on other components of RTEMS like the Scheduler and Memory Management module.

Conflict of Interests or Commitment

I have to present my Masters dissertation at my university in the last week of May during the community bonding phase. I also have to present my dissertation results at DRDO, Bangalore India around the same time. Once the coding phase begins, I will be able to invest full time to the project.

Major Challenges foreseen

- While implementing live tracing I may face some problems related to buffer overflow as mentioned above. I will have to fine tune the solution for performance and timing issues while running examples that produce large traces. I will produce tests and metrics for performance for each transport under different scenarios.
- Using libDWARF we can perform a range of operations on ELF sections of a binary. I have to port some functionalities to RTEMS to be able to parse the ELF sections and fetch signatures of some user-defined function and sizes of its parameters. I will have to consult with libDWARF maintainers to come up with a easy generic approach for function tracing in RTEMS.

References

- Trace Buffering: https://devel.rtems.org/wiki/Developer/Tracing/Trace_Buffering
- Capture Engine: https://devel.rtems.org/wiki/TBR/UserManual/Capture_Engine
- libDWARF: https://www.prevanders.net/dwarf.html

Relevant Background Experience

- I have strong C Programming Skills, and a strong background in concepts of Operating Systems. I understand the techniques for writing optimized code.
- I have worked on Embedded Systems and Networks related Projects and I will be able to understand the use cases of RTEMS developers.
- I have a good understanding of Software Engineering principles. I understand the importance of testing and documentation, which are crucial for Open Source Projects.

Personal

I am a final year student pursuing Masters degree in Computer Science from IIT (ISM) Dhanbad, India. My core interests lie in the subjects of Operating Systems and Computer Networks. I have a strong foothold on Algorithms and Data Structures. I have worked on Embedded projects and in Master thesis I have worked on WSN routing protocols and Mobility in IP Networks. I am very interested in Linux Kernel development, and I have previous experience working with RTEMS unders GSoC 2016.

Experience

Language Skill Set

- C/C++ Programming: Proficient
- Python Programming: Good