GSOC proposal (RTEMS)

Contact Information:

Name: Vidushi Vashishth
IRC nickname: reachvidu
Email: reachvidu@gmail.com, vidushivashishth96@gmail.com
Github Handle: VidushiVashishth

Title:

Enhancement of RTEMS Runtime Tracing

Introduction

RTEMS Tracing is an on-target software based system which enables tracking of the activities in user applications, 3rd party packages or the kernel in real time. The main RTEMS Trace components are:

1) RTEMS Trace Linker : It is a tool provided with the RTEMS tool set built using RTEMS Source Builder (RSB). It is a post link tool which produces a trace executable for the application to be tracked. RTEMS Trace Linker takes input in a compiled format (ELF) and instruments the application executable with additional code to enable software tracing. It can be controlled using configuration files and can be used to perform various tracing schemes.
2) Capture Engine : It is the performance monitoring and measurement framework. It is used by the instrumented code to log the trace data. Capture engine inputs trace records, filters data and provides concurrent buffering of data.
3) Common Trace Format (CTF) Integration : Common Trace Format (CTF) is a binary trace format that is designed to be very fast to write and allows C/C++ applications and bare-metal components to generate trace natively. It brings a range of features that will be beneficial for RTEMS. CTF files are generated using trace conversion libraries like Babeltrace. Babeltrace takes capture engine's trace buffers and configuration file (which contains relevant tracing information like triggers, enables and functions to be tracked) as input and converts it into CTF output.

Motivation

RTEMS trace linker should be able to generate appropriate configuration files and target code which can later be linked to post processing tools of CTF.

The current CTF integration lacks: real time trace record data extraction from the capture engine. Currently, the application to be traced is stopped to extract this data which is then input to Babeltracte for conversion to CTF format.

The improvements sought for through this project are:

- a viable transport mechanism which enables trace data transfer to the host in real/optimised time.
- Wrapping of barectf in rtems-tld to generate CTF records, suitable for buffering
- Making barectf output compatible with the capture engine
- Live tracing functionality for user defined traces / add kernel level tracing functionality (tracking thread switches and interrupt entry exit) to the tracing framework

The outcomes will achieve: A better CTF integration which does not interrupt the running application to be traced. Better debugging/tracing power for developers will be attained.

Status-quo

Currently several schemes can be utilised to generate tracing output :

- Trace Buffering
- Trace using PrintK
- Trace using CTF (currently under development)

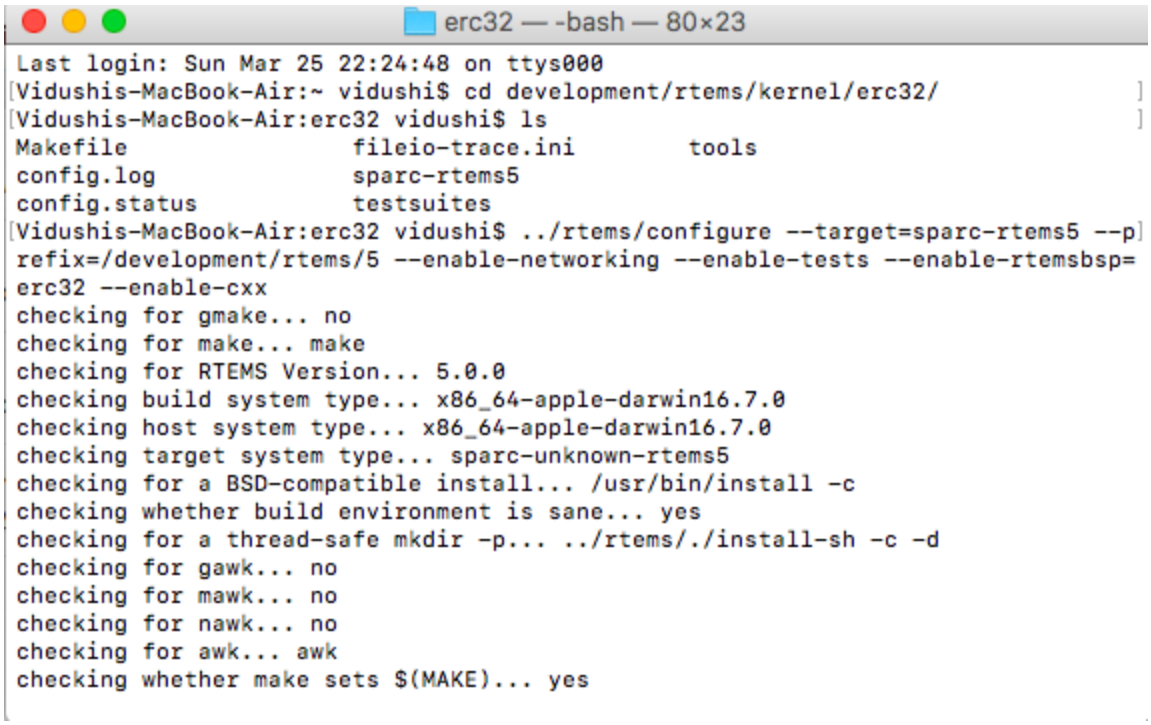
The current workflow of RTEMS tracing using CTF is explained below:

The Trace Linker uses the ELF file of the application to be traced and the user configuration file as input. It wraps functions declared in the configuration files with trace code that is used in generation of trace records for the capture engine. The linker generates the compiled file (ELF) to be run on the target and the associated CTF configuration file defining the trace records. The trace data and CTF configuration are then fed into the Babeltrace tool which converts it into CTF format. This output can be visualised using tools like Trace compass.

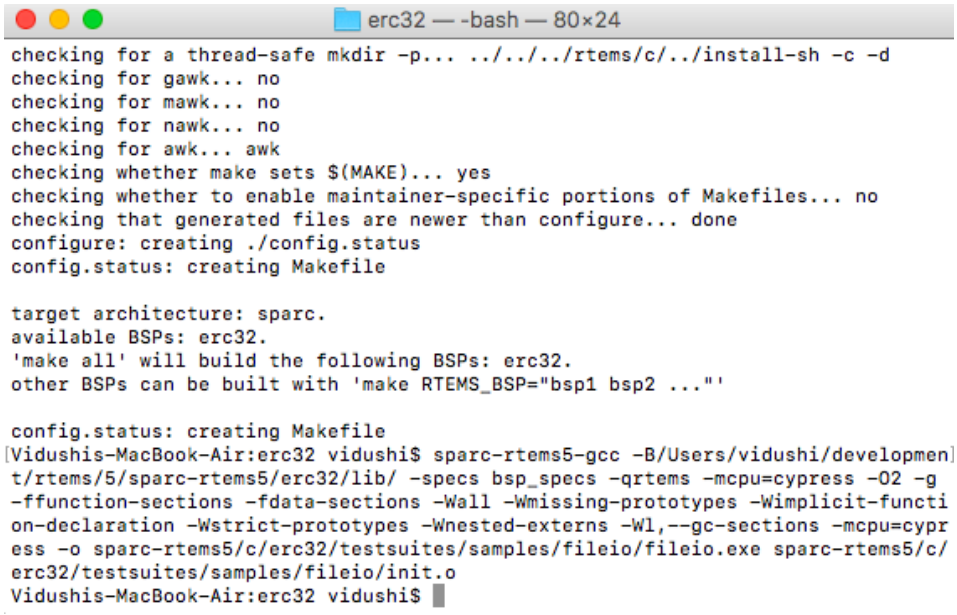
To understand the tracing framework better, I ran the trace buffering sample fileio test-case. I have set up the sparc/erc32 as my architecture/bsp pair. The prefix used for installing the rtems tools was “prefix = /development/rtems/5” .The following commands were run in the following progression to configure BSP, link application executable and trace. Finally the application is run on the sparc simulator :

1) ../rtems/configure --target=sparc-rtems5 --prefix=/development/rtems/5 --enable-networking --enable-tests --enable-rtemsbsp=erc32 --enable-cxx

Output:



2) sparc-rtems5-gcc -B/Users/vidushi/development/rtems/5/sparc-rtems5/erc32/lib/ -specs bsp_specs -qrtems -mcpu=cypress -O2 -g -ffunction-sections -fdata-sections -Wall -Wmissing-prototypes -Wimplicit-function-declaration -Wstrict-prototypes -Wnested-externs -Wl,--gc-sections -mcpu=cypress -o sparc-rtems5/c/erc32/testsuites/samples/fileio/fileio.exe sparc-rtems5/c/erc32/testsuites/samples/fileio/init.o



3) rtems-tld -C fileio-trace.ini -W fileio-wrapper --
   -B/Users/vidushi/development/rtems/5/sparc-rtems5/erc32/lib/ -specs bsp_specs
   -qrtems -mcpu=cypress -O2 -g -ffunction-sections -fdata-sections -Wall
   -Wmissing-prototypes -Wimplicit-function-declaration -Wstrict-prototypes
   -Wnested-externs -Wl,--gc-sections -mcpu=cypress -o
   sparc-rtems5/c/erc32/testsuites/samples/fileio/fileio.exe
   sparc-rtems5/c/erc32/testsuites/samples/fileio/init.o

This command throws an “error: Invalid rtems path” because of the failure of check in https://git.rtems.org/rtems-tools/tree/rtemstoolkit/rld-rtems.cpp#n52
I am currently familiarizing with the rld-rtems tools code base to identify the source of the problem.

4) sparc-rtems5-run sparc-rtems5/c/erc32/testsuites/samples/fileio/fileio.exe

This command runs fine but without traces being generated because of error in last step, there is no buffer to show.

```
Vidushis-MacBook-Air:erc32 vidushi$ sparc-rtems5-run sparc-rtems5/c/erc32/testsuites/samples/fileio/fileio.exe


*** BEGIN OF TEST FILE I/O ***
*** TEST VERSION: 5.0.0.efa0039ee94416e98636ae8228c04b1fafb42c90-modified
*** TEST STATE: USER_INPUT
*** TEST BUILD: RTEMS_NETWORKING RTEMS_POSIX_API
*** TEST TOOLS: 7.3.0 20180125 (RTEMS 5, RSB 4b3e0f8e3d6998b84a2503dd2e11578989b1672b, Newlib 3.0.0)
Press any key to start file I/O sample (20s remaining)
Press any key to start file I/O sample (19s remaining)
Press any key to start file I/O sample (18s remaining)
Press any key to start file I/O sample (17s remaining)
Press any key to start file I/O sample (16s remaining)
Press any key to start file I/O sample (15s remaining)
Press any key to start file I/O sample (14s remaining)
Press any key to start file I/O sample (13s remaining)
Press any key to start file I/O sample (12s remaining)
Press any key to start file I/O sample (11s remaining)
Press any key to start file I/O sample (10s remaining)
Press any key to start file I/O sample (9s remaining)
Press any key to start file I/O sample (8s remaining)
Press any key to start file I/O sample (7s remaining)
Press any key to start file I/O sample (6s remaining)
Press any key to start file I/O sample (5s remaining)
Press any key to start file I/O sample (4s remaining)
 =========================
 RTEMS FILE I/O Test Menu
 =========================
   p -> part_table_initialize
   f -> mount all disks in fs_table
   l -> list  file
   r -> read  file
   w -> write file
   s -> start shell
   Enter your selection ==>s
Creating /etc/passwd and group with four useable accounts:
  root/pwd
  test/pwd
  rtems/NO PASSWORD
  chroot/NO PASSWORD
Only the root user has access to all available commands.
 =========================
 starting shell
 =========================

Welcome to rtems-5.0.0 (SPARC/w/FPU/erc32)
COPYRIGHT (c) 1989-2008.
On-Line Applications Research Corporation (OAR).

Login into RTEMS
/dev/foobar login: root
Password:

RTEMS Shell on /dev/foobar. Use 'help' to list commands.
SHLL [/] # rtrace status
No trace buffer generated code in the application; see rtems-tld
```

I am currently working on the error of the second last step.

Deliverables

1) Combine CTF with the functionality of the Trace Linker. The rtems-tld tool can be programmed to internally invoke the CTF tools to generate relevant configuration files and target code to integrate with the CTF post processing tools.
2) Transport mechanism of trace records over to the host.
3) Investigate and deliver live tracing functionality over TCP that can be added to the tracing system / add kernel level tracing functionality (interrupt entry/exit and thread switches)
4) A blog post with weekly progress.

First phase:

The objective of this phase will be to integrate CTF code generation with the RTEMS trace linker. First I will understand how the current RTEMS trace system works. This will involve running sample codes (fileio example) and going through the trace buffering and trace linker wiki documentations [1]. I have partially accomplished this.

The next step will involve barectf [3] or LLTng [4]. barectf is a command-line generator of C tracers which outputs CTF packets natively. barectf takes YAML configuration files as its input. I will investigate how barectf works and its code generation. This will involve going through documentations of both CTF [2] and barectf and trying their examples. rtems-tld can build code generated by barectf for a target. This is because it already has the knowledge of target compiler and required flags. barectf currently suffers from the following limitations:

- Requires the stdin.h header file which is new in C99. I will have to check for this support in our c library and if not present I will have to create a header file of new datatypes (int8_t, int16_t, int32_t etc)
- All barectf generated tracing functions cannot to be called from an interrupt handler. I will have to come up with a synchronization mechanism to handle this.
- CTF compound data-types (arrays, structs) are not supported yet. Except in a few specific metadata locations.

After making improvements to barectf I will create an rtems-tld generator to create C code which binds to the barectf calls. Barectf output support will then be integrated with the Capture Engine. This will require understanding the Capture Engine code base and making adjustments and enhancements to it.

At accomplishment of each step I will define unit test-cases and evaluate my code. This can easily be done using existing simulators and hardware testing will not be required.

Second phase:

In this phase I will have to ascertain a method of transporting buffers from capture engine to the host machine. Keeping data gathering and data transportation on different threads is required. Integration of transport mechanisms with the capture engine is not a viable option. I will have to set up qemu for ARM machines as a working environment, if I use qemu fat files and use file transfer as a method of transport. I will

also understand libdebugger's abstraction of transport interfaces through the present examples [5, 6].
This will require more research and discussions with contributors to come up with a viable option. I intend to do this during Community Bonding period.

Third Phase:

In this phase I intend to implement *one* of the following two features post discussion with mentors during community bonding period. The chosen feature will be the one which offers higher value to the rtems community:

1) In this phase I intend to implement live tracing over TCP(as suggested on the mailing list). This will involve investigating and understanding of libdebugger's implementation of remote interfaces. I will explore Linux Trace Toolkit's functionality in Eclipse [7]. I will also research thoroughly about Capture Engine's capabilities for concurrent read and write support. If required I will make the necessary enhancements. Buffers from the capture engine will have to be transferred through the implemented interface over a chosen transport mechanism(Ethernet, USB etc) to the application's host. These will then be visualised on the host using relevant tools.
2) Focus on basic functionality of the tracing system and implement kernel level tracing. This would involve introducing tracing support for thread switching and interrupt entry/exit.

Timeline:

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

March 27th - April 23rd : I will read up on relevant documentations and try to analyse the strengths and weaknesses of current tracing system. I will prioritise the weaknesses through discussions with the mentors and begin by charting out a plan for catering to high priority and interrelated problems in an efficient order. I will familiarise myself with RTEMS trace-linker and capture engine's code bases. I will run and understand sample examples. I will identify and fix any bugs encountered on the way. I will identify the development environments required and set them up. I already have sparc-erc32. I will install arm/qemu as well.

April 24th - May 13th (Community Bonding period) : I will chart out appropriate requirement specifications, design plans and ascertain success criterias. I will provide

this in the form of relevant documentation. I will gain reaffirmation of my planned approach from mentors and decide any relevant modifications. In a nutshell the following should be complete before coding phase begins:

- Read documentations of barectf, CTF, LTTng, Trace linker
- Understand code bases of barectf, Capture Engine, Trace linker, transport mechanisms, libdebugger's implementation of tcp remote interfaces
- Have relevant development environments ready (sparc/erc32 and arm/qemu)
- Execute function tracing and trace linker samples (fileio) (almost done)

May 14 - June 11 (Phase 1) : I will create a bare-bones Proof Of Concept(PoC) of my approach to accomplish the programming tasks aforementioned in Phase 1. This way I will be able to identify any unforeseen challenges that I might have missed while planning. I will then accordingly incorporate changes, build relevant test cases and begin coding. By the end of this phase I must have :

- Identified the weaknesses in barectf and modified it to accommodate these weaknesses.
- Created a rtems-tld generator which binds it to the barectf calls.
- Make capture engine enhancements for it to support barectf output.

June 12 - June 15 (Phase 1 evaluation) : I will be evaluated on delivering the aforementioned three functionalities.

June 16 - July 9 (Phase 2) : I will begin work on transporting mechanisms to transfer record data to the host and attain satisfactory results. By the end of this phase I must have:

- Implemented file transfer mechanism using qemu fat files.
- Modified libedebugger's transport interface implementation to suit my use case.
- Integrate the outcome of phase 1 and the transport mechanism and successfully run the existing tracing sample (fileio) in this modified framework.

July 10 - July 13 (Phase 2 evaluation) : Will make improvements to my evaluation submission and prepare for next phase. I will be evaluated on delivering the aforementioned goals.

July 14 - August 5 : I will take my live tracing functionality to completion. By the end of this phase I must have:

- Implemented a live trace with user defined traces.
- Set-up a visualization mechanism for live traces. (In eclipse or other IDEs)

August 6- August 14 (Final week): I will cross check and update documentations of my work. Will make last minute improvements to mentor and self satisfaction. Will update my blog and git repositories.

Post GSoC: I will remain an active contributor of the organisation. I would also explore other transportation mechanisms for the live tracing functionality delivered. I am willing to explore other projects for contribution (I have quite an inclination towards exploring the rumps kernel project).

Conflict of Interest of Commitment

Apart from the one week of exams in May I have no other commitments during the period of the internship.

Major Challenges foreseen

- Coming up with a viable method for transporting buffers to the host machine.
- Working with some transport mechanisms (like USB). I do not have experience in using these.
- Ensuring the overheads incurred due to introduction of new functionality are as optimised.

Personal Statement

I am a fourth year undergraduate student pursuing Bachelor's in Engineering with major in Information Technology from Netaji Subhas Institute of Technology, University of Delhi. I will get over with my final exams by end of May 2018.

My programming languages of choice are C/C++/python. I have had experience in working with large codebases as well as making something from scratch through previous internships. Operating systems and Networking have been two of my favourite subjects in college curriculum. I have finished several research projects in the field of Networking and Cloud computing which have been published in International conferences and Journals. I have a strong hold on Data Structures and Algorithms.

I believe I am the right choice for this project and will work on this full time over the summer. I will give weekly updates about my progress and ensure I deliver according to

the timeline set. The project requires being efficient in C/C++ and Python, all of which I have been using for about 4 years now.

Some of my other relevant *achievements* are:

- Offered full time employment at Adobe Systems pvt lmt. as a software developer starting September, 2018.
- Worked as a software development intern at Samsung R&D Bangalore for two months last summer (2017). Was offered a full time employment opportunity on the basis of my work there.
- Sponsored by my college to present two research papers at the 51st IEEE Conference on Information Sciences and Systems (CISS) 2017, held at Johns Hopkins University in March 2017.
- Offered merit scholarship for excellence in academic performance awarded to top 5 scholars of the department.

References

1. https://devel.rtems.org/wiki/Developer/Tracing
2. http://diamon.org/ctf/
3. https://github.com/efficios/barectf
4. https://github.com/lttng/lttng-tools/blob/master/doc/live-reading-protocol.txt
5. https://git.rtems.org/rtems/tree/cpukit/libdebugger/rtems-debugger-remote-tcp.h
6. https://git.rtems.org/rtems/tree/cpukit/libdebugger/rtems-debugger-remote-tcp.c
7. https://github.com/lttng/lttng-tools/blob/master/doc/live-reading-protocol.txt