A Super Core Layer Level Abstraction for supporting 64-bit Timestamps

Thanks a lot for spending your invaluable time in reading my proposal.

Abstract

I wish to provide RTEMS with a Super Core Timestamp Class which forms an abstraction for the nanosecond timestamps implementation in RTEMS. Since timestamps are now handled as a Class the actual implementation can be tailored to serve the needs of the particular architecture and BSP that RTEMS supports. The class provides two variations which handle the nanosecond timestamps in ways different from each other. One variant handles timestamps in the standard POSIX compliant method of using timespec structure containing two separate fields for seconds and fraction of a second, viz. nanoseconds. The second variant uses a 64-bit unsigned integer representing nanoseconds since the POSIX epoch. This will make the timestamp math faster on many target architecture. Both the variants support all the math operations that can be performed on timestamps as supported by the current timespec structure implementations.

Project Details

Nanosecond timestamps currently use POSIX struct timespec. Math operations on this structure is computationally expensive. Hence this project will implement a 64-bit timestamp which is more efficient on many targets. Individual targets will be able to select the implementation that is more efficient for them.

The project begins with the verification of the correctness of all the test cases, which includes measuring execution times and the sizes of the object code/assembly code generated for each of these cases. These findings are consolidated. The next step of the project is to write a class named ScoreTimestamp (ST henceforth). This is an abstract class which supports all the time and clock related functionalities in RTEMS from now onwards. Now to begin with the first implementation of ST all the methods required to convert timestamps to and from ST to the traditional struct timespec and other supported formats like TOD(struct rtems_tod_control in RTEMS) are written.

The next stage is to replace all the operations performed on struct timespec to operate on ScoreTimestamp by providing methods for each of those operations however a few API calls will be spared. With this base infrastructure for abstraction ready verification tests are repeated.

Since ST is an abstract class with two target selectable implementations, the next stage (the heart) is to provide methods for representing nanosecond timestamps as an unsigned 64-bit integer. This involves implementing timestamps with a gcc native unsigned long long or more portable uint64_t. ST will provide methods to update the nanoseconds since last clock tick reported by BSP directly to this 64-bit timestamp, without using timespec anywhere. The details of the implementation will be provided during the design phase of this project. All the math operations that is performed on timespec will be rewritten to suit this implementation. This is not difficult since timestamp is now native gcc type all the native operations provided by gcc can be directly performed on this. Also methods for conversion to and from other standard time formats like struct rtems_tod_control will be provided.

Both the implementations will be retained. Now the entire infrastructure to operate on 64-bit timestamps is ready. A compile time option to conditionally select either of the implementations of ST by giving an option while running configure scripts or selecting a default implementation for individual targets depending on the test results if no option was used.

Finally, verification tests are repeated for all cases. Results are consolidated for evaluation.

Benefits to RTEMS

Empirical results[Ref-1] have shown the number of CPU instructions required to perform math operations drastically reduces when performed on a 64-bit integer. This proves that this implementation will be a huge performance gain for RTEMS which operates on Time Critical machines, where both time and resources are at a premium.

There is always a choice for programmers to use the traditional timespec based implementation on the targets where no reasonable gains exist if 64-bit integers are used (Empirical Results Ref-1).

The most important benefit is that, since a nanosecond requires 30 bits to represent a second and seconds are represented separately in a different field, 2 bits of the tv_nsec field in struct timespec is wasted adding to the known 2038 problem. With unsigned 64-bit timestamps 30 bits are used for nanoseconds and the remaining 34 bits can be used for representing seconds. So now 2^34 seconds can be counted thus enabling us to represent 545 years i.e. we can count upto 2515AD since POSIX Epoch!!! as against 2038AD with struct timespec. This is a huge gain.

Deliverables

1. An opaques class ST.
2. Two separate implementations for ST providing methods for conversion to and from ST format and struct timespec format or TOD formats, individual methods for performing math operations on both versions.
   I. An implementation that supports operations and data structure identical to timespec.
   II. Another implementation that supports operation on 64-bit integers. Methods to handle 64-bit timestamps like updating on report from BSP.
3. A compile time option to for the user to select either of the implementations for timestamps.

Non-Code deliverables include testing performed at 3 different stages to verify the working of ST class implementations. Also a detailed documentation for using the new ST implementations.

Project Schedule

The project is planned to be completed in 9 major phases. Every phase includes documenting the progress during that phase. The timeline for each of these phases is given below:

1. baseline testing (Community Bonding Period : Already started May 25th)
   Closely working with RTEMS community to learn more about RTEMS in depth, learning code structure of RTEMS, reading documentations related to RTEMS programming, Timestamps and Real Time Systems in general. Also baseline testing and consolidation of

its results. Learning how nanosecond timestamps are implemented in other projects like ext4.

2. Analysis and Design (May 26th June 8th)
   Analyzing previous test results. Drawing exact plans for the execution of the project which includes design of every aspect of the project and algorithms required for the project by discussing with the mentor.

3. Coding Stage I (June 9th June 22nd)
   Completing the coding of ST abstraction, and its first implementation including methods for conversion to and from other formats.

4. Coding Stage II (June 23rd July 1st)
   Completing the replacement of RTEMS wide functionalities and operations performed on timespec to use the new class.

5. Testing Phase II (July 2nd July 6th)
   Repeating the verification tests on the new class implementation and consolidation of results.

6. Coding Phase III (July 7th July 24th)
   This is the heart of this project which involves writing the second implementation of ST class, writing all the methods for conversion to and from 64-bit integers and other formats, implementing all the math operations and functionalities for this format.

7. Changes to RTEMS to support ST and Documentation
   (July 25th July 30th)
   Adding compile time options to select the implementation of ST and other general changes by interacting with the community. Also includes finishing detailed documentation of the code and how-to guides for ST class.

8. Requesting for community wide Reviews, testing and evaluation (July 31st August 11th)
   Final phase of testing of the overall project, obtaining and consolidating results and evaluation of the results. Requesting community to help me in final testing.

9. Packaging and Wrap-Up (August 12th August 18th)
   Final bug fixing if any and merging the project with the RTEMS CVS HEAD and Wrapping up the documentation.

About Me

I am a 3rd Year undergraduate pursuing Information Science and Engineering Student at BMSCE, Bangalore. I have been using and advocating Free and Open Source Softwares from past 5 years. Was one of the main co-ordinators of BMSLUG. Have co-ordinated various events including Swatantra Tech Fest, an all day Open Source Fest, Session by RMS, Hack Fest (a competition in our college). Have won few Coding Competitions since I was 13 years, the major one being a State level BASIC Programming competition at 13.

I have been actively participating in various Open Source Communities by reporting bugs to Ubuntu, GNOME etc and writing patches for Drupal.

I am also working on migrating the computers in our college from Windows to GNU/Linux. I am playing a major role in replacing Oracle Database used in our college with MySQL database. Above all this I have a fair understanding of concepts of C and have nearly 3 years of programming experience in C.

Lastly I want to express my deep commitment for this project and RTEMS. I would also like to mention that I will be fully available during this summer and assure 50-60 hours of dedicated work a week on this project. Also I will continue my commitments with RTEMS well after GSoC. If you find any part of this proposal is not clear please contact me.

Important Links and URLs

Ref-1	: http://madhusudancs.byethost13.com/rtems/empiricaltest
My CV	: http://madhusudancs.byethost13.com/sites/default/files/madhusudancsCV.pdf
Personal Blog	: http://madhusudancs.byethost13.com/
My RTEMS Blog	: http://madhusudancs.byethost13.com/rtems