May 04, 2017

A Method To Do Binary Analysis And Provide Smart Advisory Using Runtime Architecture Layer To Improve Quality And Performance in Enterprise Applications

Suprateeka R. Hegde
Hewlett Packard Enterprise

Shridhar Prakash Joshi
Hewlett Packard Enterprise

A Method To Do Binary Analysis And Provide Smart Advisory Using Runtime Architecture Layer To Improve Quality And Performance in Enterprise Applications

Abstract

Enterprise applications, by their very nature, need security, performance and robustness in addition to being reliable and scalable. Providing useful diagnostics and analysis, generating optimized code to produce such enterprise applications, have always been a research topic in compilers and system software technologies, and has so far resulted in many successful implementations through compiler optimizations and performance analysis tools. Here we propose a newer and smarter unique technique by tapping the potential of Runtime Architecture Layer of an Operating System Platform, which has been untapped so far.

RTA (Runtime Architecture) Layer mainly includes assemblers, static and dynamic linkers, loaders, runtime libraries and ELF (Executable and Linkable Format) binary information. Collectively, this layer provides a substantial amount of crucial information that helps to improve security, performance and robustness significantly, which is of highest importance in enterprise application development.

Problem statement

Applications in an enterprise world are usually mission critical in nature. It is imperative that the developer tool chain used to build such applications provide all the features that help in achieving security, performance, reliability and scalability.

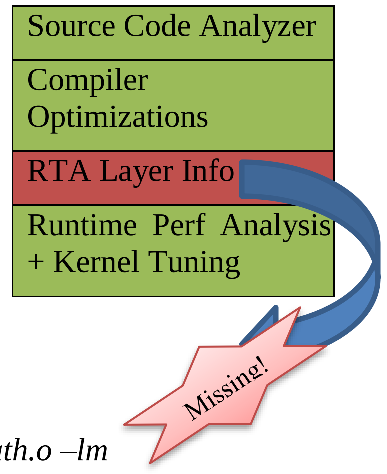
In a constant endeavor to provide such features, research in system software have been successful by implementing various source code analyzers, compiler optimizations and performance analysis tools. While source code analyzers and compiler optimizations have helped in generating efficient machine code from the source, performance analysis tools have helped in analyzing various bottlenecks at runtime.

However, in between compilation and runtime, there is RTA Layer which is responsible for all machine dependent aspects like ABI (Application Binary Interface), program startup, relocations, shared libraries, loaders, stack unwinding, executable format and many more.

Through research, we found that there is substantial amount of useful information hidden in this RTA Layer, which is untapped by any of the existing tools mentioned earlier. We found that in the domain of tool chain based analysis and diagnostics, there is huge gap at the RTA Layer on both enterprise UNIX and Linux.

This idea significantly boosts a Unix/Linux platform’s Developer Tool Chain features and capabilities, especially for enterprise application development.

Our solution



Based on the problem and our research, we propose a diagnostics and advisory tool that taps all the hidden information available in the RTA layer. This tool bridges the gap shown in the Figure 1.0.

With expertise and complete comprehension of developer tool chain and its usage, we designed our tool and solution in a development friendly way. Our tool integrates well with existing *makefiles*. All that needs to be done is prefixing the path of *rtadiag* tool for the compiler/linker line.

Tool Invocation Syntax:

rtadiag <options> compiler/linker <options> <source/object files>

Eg: *$/usr/bin/rtadiag --with-db=myprog_diag.db /usr/bin/cc main.c math.o –lm*

In the above example, our tool would provide diagnostic and advisory messages on the standard output in addition to creating a diagnostic database file by name myprog_diag.db.

Database Usage:

In order to get maximum and sustained benefits, our tool provides the creation of a database. All diagnostics are stored in a database corresponding to application. Such a database can be used to compare the diagnostics at a later stage after fixing or cleaning up the first set of diagnostics. It can also be used to ensure that further application development does not result in newer diagnostics.

The flow of our tool/solution are as follows:

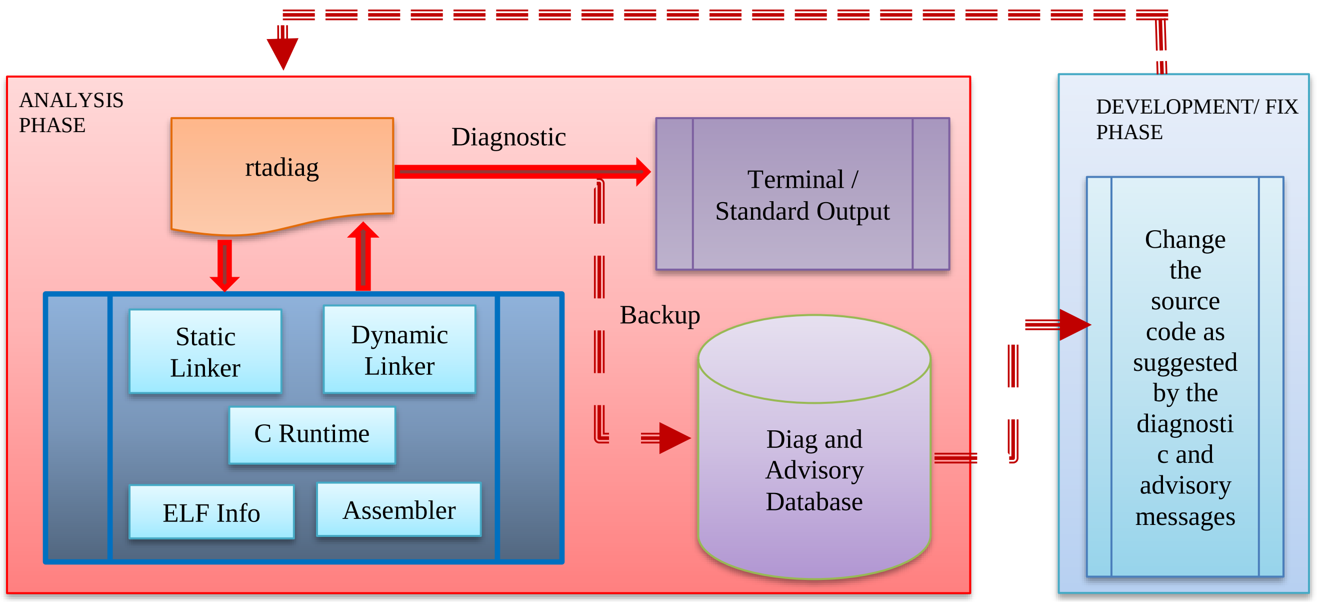

ANALYSIS PHASE
rtadiag
Diagnostic
Terminal / Standard Output
Static Linker
Dynamic Linker
C Runtime
ELF Info
Assembler
Backup
Diag and Advisory Database
DEVELOPMENT/ FIX PHASE
Change the source code as suggested by the diagnostic and advisory messages

Figure 1.0

Following are some of the highlights of unique Diagnostics and Advisory that our tool provides using the RTA Layer. Each diagnostic has a detailed documentation associated with it on how to fix the problem. Descriptions below are shortened for brevity.

- *RTA #102: Too many memory mapped data segments detected. Consider using segment merging (+mergeseg) linker option for better performance.*

  In certain cases, merging data segments of all load modules improves paging and hence performance [1]. It may also help in IPO (Inter-Procedural Optimizations) by using optimized relocations.

- *RTA #096: Number of dynamic symbols have exceeded optimal threshold. Consider using PROTECTED or HIDDEN attributes of symbols to improve both security and performance. Consider tuning +nbucket linker option for better hash performance.*

  A very high number of dynamic symbols increase symbol lookup and processing time because of more hash-chain [2] conflicts. It also exposes APIs and Data unnecessarily which impacts security. Hiding and protecting such APIs and Data symbols makes application better.

- *RTA #95: Number of shared libraries and associated symbols exceeded optimal threshold. Consider using +fastbind linker option.*

  Some application may have too many shared libraries linked with large number of symbols spread across them. In such cases program startup time is very high. Using the option +fastbind and many more suggested techniques may bring down startup time and increase performance.

- *RTA #050: Order of object files present on link line may degrade cache performance by reducing locality-of-reference. Consider using LORDER tool or use the following suggested order.*

  While writing makefiles, it is not possible to manually analyze the right order of objects files in terms of dependency and call graph analysis. Our tool provides the information on ordering the objects for best performance.

- *RTA #011: Too many relocations for the symbol <name> in file <name>. Consider using temporary variable as caching mechanism.*

  Some badly written programs may call a single API or refer to a single data variable from shared library, multiple times in a single translation unit. This degrades performance. Suggested to improve the coding style to use temporary variables for caching the actual value, instead of calling every time.

- *RTA #501: Segment Alignment Skew detected. Consider tuning segment alignment using the linker options +pd/+pi to align segments suitable for this machine.*

  Sometimes tuning segment alignment helps to improve memory performance by the underlying kernel. This is a per system setting and may not be applicable on a different system.

- *RTA #695: Relocation types that may degrade performance are seen in large numbers. Read section 2.4 "Choosing the right data type and structure to ensure better relocations" of the rtdiag documentation.*

A program badly written with incorrect data types and at incorrect references may degrade performance by using relocations and instructions that are slow. Our tool detects such relocs and advises to use the data types and references in a way that improves performance.

References

[1] Improving Application Performance Using Linker [http://link.osp.hpe.com/u/1y8n]

[2] Generic System V Application Binary Interface [http://www.sco.com/developers/gabi/latest/contents.html]

[3] Intel Itanium Processor Specific ABI [ftp://download.intel.com/design/Itanium/Downloads/245370.pdf]

[4] AMD64 Processor Specific ABI [http://www.x86-64.org/documentation/abi.pdf]

Disclosed by Suprateeka R Hegde and Shridhar P Joshi, Hewlett Packard Enterprise