ABI Working Group

Main Discussion Thursts

- What do people think ABI is?
- State of ABI tooling/Dreams of ABI tooling.
- Monitoring ABI of binaries.
- ABI concerns for tool developers.
- What we should do?

What do people think ABI is?

- ABI is getting compatibility between tools and applications. The MPI problem.
- But from broader community: people think of ABI as a compiler standard, library compatibility, standards efforts, distro contracts, …

State of ABI tooling/Dreams of ABI tooling

- Libabigail from RedHat is the production ABI tool
  - Looks for ABI breaks between program’s versions
  - Examine compatibility between library dependencies .
  - Production tool
  - Relies on DWARF
  - Can be integrated into your CI
- Smeagle is in-development research out of LLNL/UW
  - Fingerprint binaries to build binary-centric ABI model
  - Mixes DWARF and binary analysis.

Monitoring ABI of binaries.

- Ben says “All your external libraries should have their ABI monitored under CI.”
  - But Jim says “API and ABI are similar/distinct”. What should we monitor?
  - But Matt says “I want to have long arguments about this”
- Could build a new dynamic linker that looks at ABI fingerprints embedded in binary, and better decides what libraries we load.

ABI concerns for tool developers.

- Almost everyone uses LD_PRELOAD, with its ABI problems.
  - Conflicts between tool symbols and application symbols
    - Move to LD_AUDIT. Buggy, but lots of potential.
    - We need a "Tool API" on top of LD_AUDIT.
    - Deprecate (soft) LD_PRELOAD
  - The MPI problem—tools maintain per-MPI runtimes.
    - "That's not purgatory, that's a circle."
    - No great solutions here
    - Mix JIT and ABI analysis to build wrappers on load
    - MPI Shims are in use by some compute centers. Have not been widely used by tools. Have to be maintained by computing center.
    - Allinea MAP does "compile on thin MPI wrappers on load" trick. Just requires MPICC being set.

What we should do?

- The “fragile LD_LIBRARY_PATH problem” of complex library versioning requirements stomping on each other in unsolvable ways.
  - Todd will blog this
  - We need to help users/vendors understand how to package: What are the repercussions of using RUNPATH.
  - Matt: “RPATH is best practice”
  - Johnathan: “RPATH is still a problem, doesn’t mix well with LD_PRELOAD”

- Everyone: “We should rethink library search semantics.”

Original Notes

Survey: What do you think ABI is:

- Compatibility for tools, getting MPI agreement between tool and MPI
- Similar. And all the other libraries.
- We also see ABI in compiler standards, library compatibility, language standards arguments, etc

What's the state of ABI:

- Abigail (libabigail) is an analysis program for understanding the ABI of a program. Looks for ABI breaks between program versions, compare compatibility between libraries. Production worthy, can be integrated into your CI.

- Smeagle is research ABI fingerprinting program. Uses a more binary-centric approach, under development at U. Wisc/LLNL.

What should we care about in ABI as tool developers:

- Ben: All external libraries we make should have their ABI monitored. Should monitor ABI of dependencies.
- Jim: API and ABI are related and different. Can break one without the other.
- Tools are more complex. Tool and application come from different build systems.
- Ben: Should maintain ABI fingerprints of tool libraries, and check on load.
- Matt doesn't agree. Need big infrastructure changes first.
- Could make simpler model with on-load compatibility checks.

We would like to build that on-load check for ABI

- Libabigail xml encodes all of the ABI.
- Matt: But the checks between xml are hard. Involves understanding language semantics.
- Matt: Smeagle model (not implementation, yet) would be better for this.
- Ben disagrees

Ben talks about versioning to encode ABI in structure. Describes examples. Hard to maintain and write code against. But can be targeted towards specific ABI problem areas.

Back to "What should we care about in ABI as tool developers:" – Let's focus on LD_PRELOAD of RT library problem.

- Move to LD_AUDIT. You get a separate library namespace, your libraries don't conflict/talk-to application libraries.
    - But LD_AUDIT has lots of bugs. Is more reliable current/upcoming glibc versions 2.34+
    - Would benefit from a "tool API" on top of LD_AUDIT.
- HPCToolkit survived without AUDIT with "broken teeth". Had problems with symbol conflicts (libunwind). Bad hackery on MPI comm rank.
- O|SS would have a RT for every MPI. Had to rely on user to say what MPI implementation we used. "Not purgatory, that's a circle."
- We don't have a great solution here. Just got to build RTs with every MPI.
- JIT wrappers around an ABI analysis.
- MPI shims. Could be hardened. Have been used in production to swap MPIs, but ?not tools?. Julia is trying to use this to distribute MPI-independent binaries. Lots of manual work by computer center to deploy wi4mpi.
- Allinea MAP does "RT for every MPI" trick, but by asking user to provide MPICC at runtime, and building a translation shim at tool launch.

The "fragile LD_LIBRARY_PATH problem". Complex library tries can lead to mixed versions of libraries that are incompatible.

- RPATH is a better practice. Each library should RPATH to its dependencies.
- No executable should require LD_LIBRARY_PATH.
- Todd will blog this
- If you use RUNPATH, you have extra requirements on getting your NEEDED correct.
- (But you probably shouldn't use RUNPATH, RPATH deprecation aside)
- Matt would like to get RPATH un-deprecated and even encouraged as best practice.
- Johnathon doesn't think RPATH is best practice. Mess between LD_PRELOAD and RPATH.
- All agree: We should rethink library load semantics.