Valgrind BoF
Ideas, new features and directions

Everybody!

Valgrind developers and users are encouraged to participate by joining the discussion. And of course by kindly (or bitterly:) complain about bugs you find important that are still Not YET solved for that many years!?@!!!

Discuss any kind of possible improvement (technical or functional) to Valgrind.

About Me

- Mark Wielaard <mjw@redhat.com>
- Maintainer of Valgrind in Fedora, Red Hat Enterprise Linux and Developer Toolset.
- Done little Valgrind hacks here and there
- Doesn't have the full deep overview
- But did collect some questions and suggestions

About you

- Please participate, ask questions, do suggestions, give opinions!
- Please take notes and report to the mailinglist < valgrind-developers@lists.sourceforge.net >

- Lots of different stuff, ~5 minutes per “topic”.
  - Both “big” and “small” issues.
  - So lets speed up and slow down depending on interest

Appearing/Disappearing Code

- Support stack traces containing IP of "disappeared" code e.g. in memcheck, memory can be allocated by a piece of code that has disappeared at the time the stacktrace has to be shown.
- Support build-ids (can maybe look them up offline) `https://fedoraproject.org/wiki/Releases/FeatureBuildId`
- Better support compiled/JITted code. Allowing the JIT compiler to indicate to Valgrind the link between the JITted code and the source code. (See also GDB BoF, steal their code/design?)
  - MONO had an interface hack/patch
    - http://tirania.org/tmp/valgrind-mono.patch

Change the defaults?

- Revisit the default value of (some of) the command line options
  - Decrease helgrind redzone size from 16 to the minimum needed.
  - Change `-keep-stacktraces=alloc-then-free` to `alloc-and-free` default
  - Other relevant default options we should change?

XTree

- Implement a generalised "xtree"
- Massif has a data structure called an xtree. Basically, a bunch of stack traces, represented in the form of a tree, where each node of the tree contains the sum of all the memory size allocated by the called functions.
- The idea is to generalise this data structure, so as to make it usable in other contexts:
  - use the generalised one to replace the massif one.
  - also use it in memcheck (to allow massif like output from memcheck)
  - maybe other uses, e.g. to collect and show events or calls to various things, using a common infrastructure.

An interactive SQL relational interface to Valgrind data structures

- https://github.com/mfragkoulis/PiCO_QL/tree/master/src/Valgrind-mod
- Marios Fragkoulis

Client Requests as SDT markers

- SDT markers
  https://sourceware.org/systemtap/wiki/UserSpaceProbeImplementation
- Used by SystemTap, gdb, perf.
  - Source compatible with dtrace markers
- But is there anything wrong with Client Requests in the first place?

Instant leak detector

- Modify memcheck to report the last leaked pointer to a block.
- Integrate "omega" as a memcheck option or omega as a separate tool.
  `http://www.brainmurders.eclipse.co.uk/omega.html`

80 bit arithmetic on x86/AMD64

- Some complains because it surprises users.
- Is it an open problem? Would be too slow? Or just work nobody has done yet? How much work would it be?

VEX API redesign

- Currently geared toward the dynamic paradigm
- APIs do the whole process: lift, instrument, optimize, finalize, instruction select, compile back to binary.
- Issue for static analysis (PyVEX)
  - Have to patch out half of LibVEX_Translate
- Yan already has some patches?

Cross-arch VEX/Valgrind

- VEX has some host-platform and guest-platform homogeneity assumptions.
  - example, compiled as-is on x86, the MIPS translation code is broken due to the fact that neither MIPSBE nor MIPSLE is defined.
- What would it take to make valgrind cross-arch?
  - syscall layer
- How about starting with i686 on x86_64?

Which CPUID is it anyway?

- Valgrind isn't completely consistent in handling host CPU capabilities vs VEX emulation capabilities.
  - What can we do to improve that?
- Make it user tunable?

VEX split lift-to-IR and compile-back

- VEX assumes that any platforms that it implements are going to be lifted to IR and then compiled back down.
- For static analysis we only need lifting.
- Partially supported arches?

Improve memcheck leak heuristics

- In 3.9.0, some heuristics were added to memcheck to decrease the false positive rate of possible leaks for c++ objects (such as std::string).
- Add more of such heuristics?
- And/or have a more flexible way to define heuristics, e.g. using "user definable expressions"?
- Add a way to specify a stack trace to match for a heuristic?

helgrind improvements

- Currently, in race conditions errors, locks are only described by an address and their creation stack trace. Add more info (when possible) based e.g. on `--read-var-info=yes`
- Speed up helgrind 'mini stacktrace' capture avoid to take duplicate stack traces? Or have a way to detect only the top most IP has been updated since previous stack trace?
- Suppressions entries for helgrind with matching the stack trace of one or the other or both threads involved.

Making Valgrind multi-threaded

- parallelising Memcheck
- parallelising the rest of the framework
- Other tools

Valgrind and transactional memory

- Currently xbegin “fail early, fail often”
- Could we do something more interesting?
- Could we use tx in V itself?

Make Callgrind work sanely on ARM (and PPC)

- The Callgrind algorithm to track call and return is to be improved to work properly on these platforms.
- Is there a way to make this better?
  - E.g. by having a fast way working in most cases, and rely on unwind info in the difficult cases. Can we detect at instrumentation time that an instruction is a difficult case?

Redo the JIT framework to reduce baseline overheads

- Could we reuse some "compiler lib" (qemu tcg, llvm or gcclib as code generator)?
  - Could we reuse some "compiler lib" (qemu tcg, llvm or gcclib as code generator)? Destroys startup time?
- Any other suggestion to (significantly) improve the speed of Valgrind JITted code?

Release/bugfixing strategy/policy

- README_DEVELOPERS_processes
- Timed minor releases (every X months)?
- Split SVN Valgrind/VEX
  - Merge?
- GIT or Mercurial?

Packaging valgrind for distros

- handling patches (more frequent releases?)
- Suppressions (who should ship them?)
  - Can we push it to other packages/libraries?

Website/Bugzilla/IRC

- valgrind.org
- Web site pages in svn?,
  - So it can be updated by all developers with a patch as everything else
- Do we want a wiki?
  - For Developer? For Users?
- Make sure bugzilla sends a mail to the developer mailing list when there is a new bug/comment in a bug.
  - Currently you have “watch” Julian.
- Have a log of irc so development ideas can be seen by all?
  - Social impact? Do people want to be logged?
- Where to put tests/performance results?

Easy hacks/New Developers

- Create easy/mediam/hard hacks, like libreoffice is doing
  - Syscall wrappers?
  - New instruction sets?
  - Run valgrind in anger

    ```
    valgrind -q --trace-children=yes bash
    ```

    And fix anything that falls out?
- Have some GSOC ideas?

Darwin/MacOS and otherOS

- What do we need to continue to support it?

- What about other ports?
  - Solaris, can it be integrated?
  - Windows, what is its status?
    `http://sourceforge.net/projects/valgrind4win`