Fun tasks maintaining Valgrind

Perftools Toronto 2018

Mark J. Wielaard

redhat.

TL;DR (Summary)

There are lots of little issues that come up maintaining Valgrind.

They are fun (really!) things that teach you about the kernel, compiler and linker.

How does valgrind work I

- Select (static linked) core/tool executable
  - memcheck-amd64-linux
  - Loaded at fixed high address
- Setup LD_PRELOAD
  - vgpreload_memcheck-amd64-linux.so
  - Contains overrides, helpers that run on ‘virtual processor’
- Load actual executable ELF and ld.so in memory
- Setup stack, other environment, etc. as if ld.so was started by kernel...

How valgrind works II

- Translates all instructions (basic blocks) into IR (intermediary representation) that is explicit about which bits are set/moved where.
- Hands IR to selected tool for ‘transformation’ (might include ‘dirty handlers’, which are ‘function calls’).
- Translates augmented IR back to native instructions and executes them.

memcheck

- Most used tool
- Tracks which bits are defined
  - Instruments IR to keep track of shadow memory and registers
  - Warns when a flow of control depends on undefined data
  - Intercepts malloc, free, etc.
  - Keeps track of stack, etc.

Two “common” issues

- Syscalls used in new and interesting ways
  - Either because of a kernel update, but probably because of a glibc update
- Unknown instructions (or better modelling of instructions)
  - Update of gcc, new optimization

We see them first, since Fedora is first and we care about all (RHEL) arches.

And valgrind is “lazy”, if we haven’t seen it, it might not be implemented...

utimensat

change file last access and modification times

- Tar “suddenly” started emit errors when run under valgrind when unpacking some files
  - Turned out it was using a gnulib wrapper for utimensat, but after an upgrade used “direct” glibc/linux utimensat.
  - In the new case not all timespec fields are set (or used by the kernel)

utimensat

```
int utimensat(int dirfd, const char *pathname,
              const struct timespec times[2], int flags);

             struct timespec {
                 time_t tv_sec;        /* seconds */
                 long   tv_nsec;       /* nanoseconds */
             };
```

commit `790f5f3018f807153339e441e7aea1414f4b5c8d`

GCC 8.1 is too smart

- Conditional jump or move depends on uninitialised value(s)

  if (! state→initial…) ...

  where state is an struct that starts with

  ```
  bool initial : 1; /* Use just one bit */
  bool frob : 1; /* … */
  ```

- GCC generates:

  ```
  cgijl %r1,0,0x487e560
  ```

  (signed) Compare Grande Immediate and Jump on Low

- commit `51736549e33fc8468e47861031a70c7f8cadd691`

Larger future task

- Add an elfutils/libdw based (external) DWARF reader
  - Current one is buggy and slow
  - Adds a pull in debuginfo from remote option
  - All the funky DWARF5, split-dwarf, etc. goodness.