utrace: a new in-kernel API for debugging and tracing user tasks

Roland McGrath

redhat

utrace is ...

- ... not a tracer (like ftrace et al)
- ... not a user-level ABI (like ptrace)
- ... not a clever choice of name!

- ... an in-kernel API (for loadable kernel modules to use)
- ... a multiplexing layer (not just one new kind of tracing)
- ... intended to support general-purpose user debugging
  - can change user threads' behavior (stop, step, perturb)
- ... a means to build new interfaces and other new features
- ... a clean platform for reworking the ptrace() internals

redhat

utrace prerequisites

- tracehook.h (2.6.27)
  - clean, well-documented set of calls from kernel core
  - utrace patch touches tracehook.h (`#ifdef CONFIG_UTRACE`)
    - no need to touch core code directly, merges easy
- arch code
  - user_*_step (2.6.25)
  - user_regset (2.6.25)
  - asm/syscall.h (2.6.27)
  - etc. (see arch/Kconfig comments)
  - **There is no arch-specific code at all in utrace itself.**
- 2.6.29 arch support (`HAVE_ARCH_TRACEHOOK`)
  - ia64, powerpc, s390, sh, sparc, x86
- Future arch support: Ask your arch maintainer!

redhat

utrace goals

- Establish platform for new work
  - API for kernel modules
  - allows multiple separate uses: “tracing engines”
  - bottom layer, usable by non-gurus
    - block_device:fs :: utrace:tracing engine
    - net_device:net proto :: utrace:tracing engine
- Help you do it right
  - non-invasive (no interference with signals, wait, etc.)
  - low-overhead
  - arch-independent
  - maintain system invariants (SIGKILL)
  - callbacks at safe points

redhat

utrace API uses

- In progress
  - Uprobes (Jim Keniston et al)
  - Systemtap
  - kmview (Renzo Davoli)
  - Seccomp clean-up or replacement
    - bone-simple with utrace, no asm hacking required
  - ptrace() clean-up (Oleg Nesterov)
- Ideas/vaporware
  - UML helper module
    - Share code with kmview?
  - New user-level debugger ABIs (ptrace killer)
  - This space for rent

redhat

utrace API concepts

- tracing engine = your code, calls into utrace API
- API calls are per-thread (aka task)
- asynchronous attach/detach
  - struct utrace_engine pointer is handle
- event callbacks (at safe points)
  - place to access thread state, user memory, etc.
    - via user_regset, other kernel APIs or data structures
- control
  - stop
  - resume, step, interrupt, report
  - detach
- report & quiesce: explicit synchronization via callbacks

redhat

utrace events

- SYSCALL_ENTRY, SYSCALL_EXIT
  - entry/exit distinguished, unlike ptrace
- SIGNAL, SIGNAL_{IGN,STOP,TERM,CORE}
  - signal disposition distinguished, unlike ptrace
  - no signal event for SIGKILL (only EXIT/DEATH/REAP)
- EXEC
- CLONE
  - thread/child tracking can be set up by callback
- JCTL
  - not possible with ptrace
- EXIT, DEATH
- REAP
  - not possible with ptrace
- QUIESCE (catch all)

redhat

utrace callbacks: “safe points”

- close to user mode, no entanglements
  - returning to user (before signals), or syscall entry
  - no locks, preemptible
  - can block (modulo “well-behaved” interaction rules)
- can use user_regset (read or modify)
  - only places user_regset calls are kosher
  - except DEATH, REAP
- can stop here (`TASK_TRACED`)
  - same as ptrace() stops; ps shows “T”, etc.
- QUIESCE callback
  - catch-all at any event that any engine traces
- `UTRACE_REPORT, UTRACE_INTERRUPT`
  - engine can request QUIESCE via utrace_control()

redhat

utrace API

- struct utrace_engine_ops
  - callback function pointers for each event type
- struct utrace_engine
  - void *data
  - utrace_engine_get() / utrace_engine_put()
- struct task_struct vs struct pid
  - choose your refcount/RCU poison
- enum utrace_resume_action
- utrace_attach_task() or utrace_attach_pid()
  - attach new engine, or look up attached engine
- utrace_set_events() or utrace_set_events_pid()
- utrace_control() or utrace_control_pid()
- utrace_barrier() or utrace_barrier_pid()
- utrace_prepare_examine(), utrace_finish_examine()

redhat

utrace callbacks

- run in traced thread
  - except sometimes REAP (runs in parent calling wait())
  - always at “safe point”
- arguments: engine, resume action, + event-specific
- return value
  - resume action (resume/stop/step/etc.) + event-specific
- well-behaved callbacks
  - don't run too long (using traced thread’s CPU time!)
  - don't block much (could break other engines, SIGKILL!)
  - use UTRACE_STOP to sleep: woken via utrace_control()
- synchronizing with callbacks
  - death races: utrace_set_events()/utrace_control() errors
  - utrace_barrier()

redhat

Callback example

```
static u32 syscall_exit(enum utrace_resume_action action,
                struct utrace_engine *engine,
                struct task_struct *task,
                struct pt_regs *regs)
{
   printk("pid %d syscall-exit %ld\n",
          task->pid, syscall_get_error(task, regs));
   return UTRACE_RESUME;
}
...
static const struct utrace_engine_ops my_ops = {
    .report_syscall_exit = syscall_exit,
};
...
```

redhat

utrace API future work

- API tweaks
  - callback order (engine priorities?)
  - syscall_entry inverse callback order?
  - UTRACE_STOP synchronization corners
  - stop/resume notification for syscall_entry
- extension events
  - avoid overloading signals
  - use for hardware trace events
  - dynamically-registered
  - tie-in with tracepoints?
- hw_breakpoint integration (use extension events)
- BTS integration (use extension event for buffer full)

redhat

Beyond utrace: lots of hacking to do!

- clean up ptrace() implementation
  - work in progress for 2.6.31 (Oleg Nesterov)
- entirely new user-level interfaces
  - fd-based, pollable
  - minimize kernel-user round-trips with debugger
- “groups & rules” engine
  - Underlies user-level interface + in-kernel uses (stap)
  - Trace many threads/processes uniformly (“groups”)
  - Event rules: filters & actions
    - Gather details (registers, etc.) & report to userland
    - Callback (e.g. to stap probe)
    - Manage groups (e.g. on clone, exec)

Questions?

roland@redhat.com | **people.redhat.com/roland**
utrace-devel@redhat.com | **sourceware.org/systemtap/wiki/utrace**