Uprobes: User-Space Probes

Jim Keniston: jkenisto@us.ibm.com

Srikar Dronamraju: srikar@linux.vnet.ibm.com

April 15, 2010

Topics

Overview

- What and why?
- Two versions of uprobes
- Features
- Uses
- Tie-ins to kprobes, ~~utrace,~~ SystemTap

Utraceless uprobes

gdbstub

Q&A

Overview: What and Why?

What:

- kernel API, analogous to kprobes
- breakpoints for user apps, handled in kernel

```
struct uprobe u;
...
u.pid = 1234;
u.vaddr = 0x080484a8;
u.handler = my_callback;
result = register_uprobe(&u);
```

Overview: What and Why?

Why?

- useful for dynamic, ad hoc instrumentation
- handlers have system-wide view: kernel and apps
- useful for multithreaded apps
- overcomes some limitations of ptrace:
  - Uprobes incur lower overhead.
  - “Who can probe whom” defined by uprobes client.

Overview: Two Uprobes Versions

- Utrace-based
  - Exploits utrace's signal, clone, exec, exit, and quiesce callbacks
  - First fully functional uprobes prototype October 2006
  - Ships as part of SystemTap runtime
  - Jan 2010 LKML review: uprobes maybe, NAK utrace
- Utrace-independent (AKA utraceless)
  - First LKML review March 2010
  - Uses Roland's tracehooks
  - Threads run during breakpoint insertion
  - Stripped-down implementation

Overview: Features

- no need to modify probed process's source or binary
- per-process
  - All threads in process can (independently) hit probepoint.
- breakpoint probes (uprobes) *and function-return probes (uretprobes)*
- (Kernel) handler runs on probe hit.
  - Handler runs in context of probed task.
  - Handler can sleep – e.g., for kmalloc or paging.

Overview: Uses

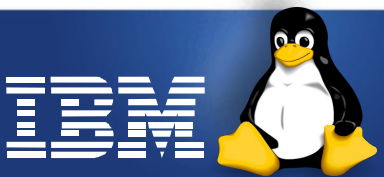
- *Typical use is via an ad hoc instrumentation module,* a la kprobes.
  - SystemTap uses uprobes for user-space probing.
- trace-events code under review
- TODO: perf interface
- gdbstub for uprobes/utrace on back burner
- System-call interface possible:
  - new system call API
  - enhancements to ptrace
- Architectures supported: x86 (32- and 64-bit), *powerpc, s390, ia64*

Tie-ins to Kprobes

- Kprobes-like API: [un]register_u*[ret]*probe()
- Probed instruction executed out of line (XOL):
  - Leave breakpoint in place; execute copy of probed instruction...
  - ... to avoid probe misses in multithreaded apps.
  - Can be “boosted” to avoid 2nd (single-step) trap.
  - *Single-stepping inline provided for jump-starting ports.*
- Uprobes-specific complications:
  - “Out of line” instruction copies must reside in probed process's address space. *Ditto the return-probe trampoline.*
    - Solution: XOL vma
  - Need to handle full instruction set (not just kernel instructions), guard against evil apps.

Tie-ins to Utrace and SystemTap

- *Utrace-based uprobes is packaged with the SystemTap runtime.*
  - *Probes C, C++ apps.*
  - *Exploits existing (DTrace) static probes to trace interpreted languages (Java, Python, tcl).*



2010 Uprobes –= Utrace

- Result of Jan 2010 uprobes review and NAK of utrace
- Intercepts breakpoint and single-step traps before they become SIGTRAPs.
- Exploits Roland's tracehooks for process-lifetime events.
- Background page replacement = no need to quiesce threads for breakpoint insertion/removal

2010 Uprobes, cont.

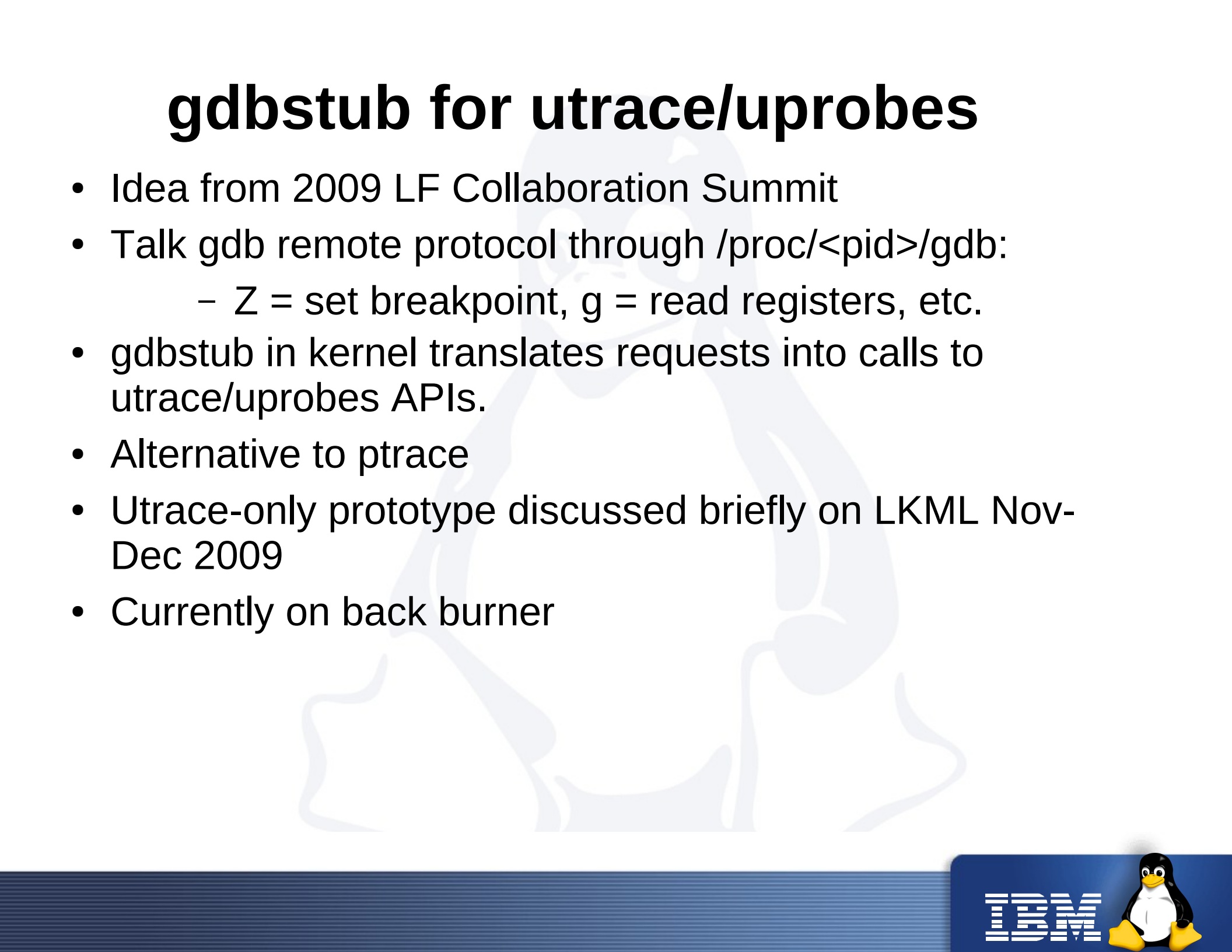
- Slimmed down for LKML reviews:
  - x86 only
  - 1 uprobe per probepoint
  - limited number of uprobes per process
  - no function-return probes
  - no option to single-step inline
  - built-in only: no uprobes.ko version
- Also on TODO list:
  - perf interface: exploit symbol table, debuginfo
  - uprobes booster: eliminate the single-step trap
  - bulk registration/unregistration
  - u[ret]probe objects reusable immediately after registration?
  - See also Issues

2010 Uprobes: Issues

- Per-process vs. per-executable (global) probes
  - How to trace process right from exec?
- Interrupt-context option for handlers
  - Performance (?) vs. complexity
- XOL area
  - Currently, uprobes adds XOL vma.
  - Which of 47 slot-allocation algorithms?
  - Add XOL area to thread-local storage?
  - Emulate instructions?
- Re-integrate ubp, XOL layers?

gdbstub for utrace/uprobes

- Idea from 2009 LF Collaboration Summit
- Talk gdb remote protocol through /proc/<pid>/gdb:
  - Z = set breakpoint, g = read registers, etc.
- gdbstub in kernel translates requests into calls to utrace/uprobes APIs.
- Alternative to ptrace
- Utrace-only prototype discussed briefly on LKML Nov-Dec 2009
- Currently on back burner



Legal Statement

This work represents the view of the author and does not necessarily represent the view of IBM.

IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.

Questions?

Backup slides

Single-stepping Inline

- Establish probepoint:
  - Replace original opcode with int3
- Breakpoint trap:
  - Run user's handler
  - Replace int3 with original opcode
  - Single-step original instruction
- Single-step trap:
  - Replace original opcode with int3
  - Continue at next instruction

- *Doesn't work for multithreaded apps.*

Single-stepping Out of Line

- Establish probepoint:
  - Replace original opcode with int3
  - Allocate XOL slot
  - Copy original instruction to XOL slot
- Breakpoint trap:
  - Run user's handler
  - Single-step instruction copy
- Single-step trap:
  - “Fix things up”
  - Continue at next instruction

- *Works for multithreaded apps*

Boosted Probepoint

- Establish probepoint:
  - Replace original opcode with int3
  - Allocate XOL slot
  - Copy original instruction to XOL slot
  - Append jump from XOL slot to next instruction
- Breakpoint trap:
  - Run user's handler
  - Continue at XOL slot

- *Works for multithreaded apps*
- *No single-step trap*

Interrupt-context handlers

- Handle bkpt trap as SIGTRAP: ~3 usec/hit
- Handle bkpt trap earlier in process context: 1.0 usec/hit
- Handle bkpt trap in interrupt context: 0.9 usec/hit

History

Spring 2006: Pre-utrace uprobes prototype skewered on LKML, soon discarded.

- “Probe per-process, not per-executable.”

June 2006: Utrace first posted to LKML.

Oct 2006: First working prototype (i386) of utrace-based uprobes

Winter-Spring 2006-2007: More features, more architectures, more testing

April 2007: ~~Uprobes posted to LKML~~ Utrace dropped from -mm tree.

History, cont.

Oct 2007: Uprobes tucked into SystemTap runtime.

Summer 2008: SystemTap += DWARF-based probing of user apps; utrace revamped

Winter 2008-2009: Uprobes refactored (instruction analysis, breakpoint innards, XOL, uprobes API)

2009: Utrace revamped again; ptrace re-implemented as utrace client, again.

January 2010: Utrace-based uprobes on LKML

March 2010: Utraceless uprobes on LKML