



# Systemtap Overview and Roadmap

February 2009

Frank Ch. Eigler <[fche@redhat.com](mailto:fche@redhat.com)>  
systemtap lead



# need it?

- when you need to see into the whole system
- when you need more than preset bulk tracing of events/values as in ftrace, lttng
- when you need less disturbance than a stop&examine debugger



# scope

- system-wide programmable tracing/probing
- compatible with a wide range of kernels, distributions
- operates on live system, no patch/reconfigure/recompile/reboot
- measure time, access any data, explore control flow, correlate events, inject faults



# sample – syscall tracing

```
probe syscall.open {  
    printf ("%s(%d) open (%s)",  
           execname(),pid(),argstr)  
}  
probe timer.s(4) { exit () }
```

```
# stap that.stp
```

```
vmware-guestd(2206) open ("/etc/redhat-release", O_RDONLY)  
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)  
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)  
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)  
df(3433) open ("/etc/ld.so.cache", O_RDONLY)  
df(3433) open ("/lib/tls/libc.so.6", O_RDONLY)  
df(3433) open ("/etc/mtab", O_RDONLY)  
hald(2360) open ("/dev/hdc", O_RDONLY|O_EXCL|O_NONBLOCK)
```



# sample – OOM image dump

```
/* use embedded C to trigger kernel dump */
```

```
%{
```

```
#include <kernel.h>
```

```
%}
```

```
function dump(msg) %{  
    panic("%s", THIS->msg);
```

```
%}
```

```
probe kernel.function("__oom_kill_task") {  
    dump("__oom_kill_task called\n");  
}
```

```
# stap -g panic-on-oom.stp &
```



# operation part 1

- compile probe script foo.stp:
  - parse script
  - combine it with tapset (library of scripts by experts)
  - combine it with debugging information, probe catalogues, event source metadata
  - generate C code with safety checks
  - compile into kernel module with kbuild
  - result: vanilla kernel module



# operation part 2

- run probe module foo.ko:
  - load into kernel
  - detach (flight-recorder mode) or consume trace live
  - unload
- probe module may be cached, reused, shared with other machines running same kernel



# recent developments

- probing user-space programs
- attaching to user + kernel markers
- organizing more samples, documentation
- easing deployment
- easing usability by kernel developers
- better error messages



# user-space probing

- finally, system-wide, seamless, symbolic
- based upon dwarf debugging data
- dynamically instrument binaries, shared libraries, potentially at the statement level
- easily trace variables

```
probe process("a.out").function("*").call {  
    log($$parms)  
}
```



# user-space probing

- measure average dbms query execution times

```
function time() { return gettimeofday_us() }
probe process("psql").function("SendQuery").call
{
    entry[tid()]=time()
}
probe process("psql").function("SendQuery").return
{
    tid=tid()
    if (! ([tid] in entry)) next
    query=user_string($query)
    queries[query] <<< time() - entry[tid]
    delete entry[tid]
}
/* and another probe to format report */
```



# user-space probing

```
#      uS query
12     990 DELETE FROM num_result;
6      3909 COMMIT TRANSACTION;
6      132 BEGIN TRANSACTION;
6      143 SELECT date '1999-01-08';
4      3651 insert into toasttest
values(decode(repeat('1234567890',10000), 'escape'));
4      3786 insert into toasttest
values(repeat('1234567890',10000));
4      1218 SELECT ' ' AS five, * FROM FLOAT8_TBL;
3       804 END;
3       295 BEGIN;
3      1032 INSERT INTO TIMESTAMPTZ_TBL VALUES ('now');
```



# user-space markers

- use dtrace instrumentation already in some code (postgresql, mysql, shells, scripting languages)
- initially based upon dwarf debugging data

```
probe process("postgres").mark("*") {  
    log($name) # $arg1 ... available  
}
```



# kernel markers/tracepoints

- statically compiled into kernel/programs
- supplements dynamic instrumentation
- higher performance, reliable data
- shared hook sites between tracing tools



# samples/documentation

- samples installed, categorized, also online
  - <http://sourceware.org/systemtap/examples>
- “beginner's guide”
  - <http://tinyurl.com/ar8wat>
- wiki
  - <http://sourceware.org/systemtap/wiki>



# deployment/usability

- remote script compile server
  - no deployment-side debuginfo/compiler needed
  - “stap-client” prototype impersonates “stap”
  - “stap-server” runs on nearby machine
  - avahi-based autoconfiguration



# under construction

- java probing & backtracing
- unprivileged user support
- kernel tracepoint, ftrace interfacing
- gui-controlled integrated general monitoring
- system-wide backtracing for deep profiling
- better quality and smaller quantity of debuginfo



# **systemtap**



<http://sourceware.org/systemtap>