



Writing SystemTap Scripts

红帽亚太区

张汉辉

eugeneteo@kernel.sg

GNOME.Asia 2008 峰会

Have you ever asked yourself...

- Is there a memory leak?
- Where are the hotspots in my program?
- Why is there so much I/O going on?
- What performance statistics should I collect?
- What is the stack size my program is using?
- Why did the Out-of-memory (OOM) killer kick in?
- How can I generate a procedure call-graph quickly?
- Why does my battery drain so quickly?
- And the list goes on...

Existing Tools

- **Existing tools in Linux**
 - Profiling – oprofile, perfmon2, VTune
 - Statistics – iostat, lockstat, netstat, vmstat
 - Tracing – strace, ltrace, gdb
 - Printing – printf, printk
 - Debugging & Live patching – gdb, kdb, kgdb, crash
- **Elsewhere**
 - DTrace, apptrace, mdb, truss



Introducing SystemTap

- **Dynamic instrumentation tool for Linux**
 - Complete framework for tracing, collecting data, and reporting
 - Flexible language lets *you* define the collection
 - Safely probe even on production systems
 - Little to no overhead when not in use
- **Free and open source software (GPL)**
- **Contributions by Red Hat, Hitachi, IBM, Intel, and others**

Target Users

- **Usage model is flexible**
- **Applicable to many different type users, including:**
 - System Administrators
 - Application Developers
 - Kernel Developers
 - Technical Support
 - Software Maintenance
 - Students and End Users

How to use SystemTap

- **Run pre-existing SystemTap scripts**
 - See <http://sources.redhat.com/systemtap/wiki/ScriptsTools>
 - See <http://sources.redhat.com/systemtap/wiki/WarStories>
- **Write your own SystemTap scripts**
- **Within existing shell scripts**
 - Helps to know shell scripting, and procedure languages
- **Use SystemTapGUI (IDE for SystemTap)**
 - See <http://stapgui.sourceforge.net/>

Getting Started

- **On Fedora, install using yum:**
 - yum install systemtap gcc make
 - yum install kernel-devel yum-utils
 - debuginfo-install kernel
- **On Ubuntu, install apt-get:**
 - apt-get install systemtap
 - apt-get install linux-headers-generic gcc make
 - apt-get install linux-image-debug-generic
 - In `-s /boot/vmlinux-debug-`uname -r` \`
`/lib/modules/`uname -r`/vmlinux`
- **For others, see <http://sourceware.org/systemtap/wiki/>**

SystemTap Scripting Language

- Scripting language resembles **awk**
- Two main outermost constructs:
 - Probes and functions
- Within these outer constructs are:
 - Statements and expressions in C-like syntax

Scripting Language Basics

- **global VAR1[=<value>], VAR2[=<value>]**
 - declares variables that are accessible anywhere
- **probe PROBE { HANDLER }**
 - defines a probe location and its handler
- **function FUNC (ARG1, ARG2, ...) { BODY }**
 - defines global functions for reusable code
- **# comment to the end of the line**
// comment to the end of the line
/* C-style enclosed comment */

Datatypes and Operators

■ Numeric type **'long'**

- 64-bit signed integer

- Supports C-type arithmetic operators:

* / % + - >> << & ^ | && || = *= /= %= += -= >>= <<=
&= ^= |= < > <= >= == !=

■ String type **'string'**

- Zero-terminated string in a fixed-length buffer

- Supports concatenation and comparison:

. .= < > <= >= == !=

■ Associative arrays (global only)

- Mapping between long/string indexes to long/string/stat value

■ Statistics (global only)

- Accumulates longs with <<< operator

Statements

- **If (EXP) STMT1 [else STMT2]**
- **while (EXP) STMT**
- **for (EXP1; EXP2; EXP3) STMT**
- **foreach (VAR in ARRAY [limit EXP]) STMT**
 - Loop over each element of the named global array, assigning current key to VAR
- **foreach ([VAR1, VAR2, ...] in ARRAY [limit EXP]) STMT**
 - Same as above, used when array is indexed with a tuple of keys
- **break, continue (while or for or foreach)**
 - Exit, or iterate the innermost nesting loop
- **return EXP**
 - Return EXP value from enclosing function

Script Arguments

■ Command-line arguments

- Use `$1 .. $N` to access numeric arguments
- Use `@1 .. @N` to access string arguments
- Use `$#` for the number of arguments (or `@#` as a string)
- Missing arguments from the user will trigger a compile-time error

■ Access C-style command-line arguments

- Use `argc` for the number of arguments
- Index `argv[]` for each argument, limited to 32 arguments

Writing Probes

- Main construct of the language
- General syntax:
 - `probe PROBE1 [, PROBE2] { [STMT ...] }`
- “Target variables” are presented as variables prefixed with “\$”
- Probe points may be defined using “aliases”:
 - `probe syscall.read = kernel.function(“sys_read”) {
 fildes = $fd # target variable
 if (execname == “init”) next # skip rest of probe
}`
- An alias is used just like a built-in probe type:
 - `probe syscall.read {
 printf(“reading fd=%d\n”, fildes)
}`

Writing Functions

- **Scripts may define subroutines to factor out common work**
- **General syntax:**
 - `function FUNC[:type] (ARG1[:type], ARG2[:type], ...) { BODY }`
- **Type declarations are inferred is absent**
- **Example function:**
 - `function thisfn (arg1, arg2) {
 return arg1 + arg2
}`
 - `function thisfn:string (arg1:long, arg2) {
 return sprintf(arg1) . Arg2
}`
- **Functions may call others or themselves recursively, up to a fixed nesting limit of 10**

Example #1: Obligatory first script

- `% stap -e 'probe begin { println("Hello World!") }'`
`Hello World!`

Example #2: Fibonacci

- **# fib1.stp**
by Josh Stone from Intel
probe begin { println(fib(\$1)); exit() }
function fib(n) {
 if (n < 0) return 0
 if (n == 1) return 1
 return fib(n - 1) + fib(n - 2)
}
- **% stap fib1.stp 4**
3

Example #3: More fibonacci

- **# fib2.stp**
by Josh Stone from Intel
probe begin { println(fib(\$1)); exit() }
global fibdata
function fib(n) {
 fibdata[0] = 0
 fibdata[1] = 1;
 for (i=2; i<=n; ++i)
 fibdata[i] = fibdata[i-1] + fibdata[i-2]
 return fibdata[n]
}
- **% stap fib2.stp 10**
55

Example #4: Trace when process executes

- ```
#!/usr/bin/env stap
execve.stp
probe syscall.exec* {
 printf("exec %s %s\n", execname(), argstr)
}
```
- ```
$ stap execve.stp
exec hald-runner /usr/lib/hal/scripts/hal-system-killswitch-get-power
exec hal-system-kill /usr/bin/hal-is-caller-privileged -udi \
    /org/freedesktop/Hal/devices/ipw_wlan_switch -action \
    org.freedesktop.hal.killswitch.wl
exec hal-system-kill /bin/basename /usr/lib/hal/scripts/hal- \
    system-killswitch-get-power
exec hal-system-kill /usr/libexec/hal-ipw-killswitch-linux getrfkill
^C
```

Example #5: Top process I/O

- **# pid-iotop.stp**
Based on a script by Mike Grundy and Mike Mason from IBM
global reads, writes
probe vfs.read { reads[pid()] += bytes_to_read }
probe vfs.write { writes[pid()] += bytes_to_write }
print top 5 IO users by pid every 5 seconds
probe timer.s(5) {
 printf ("\n%-10s\t%10s\t%15s\n", "Process ID",
 "KB Read", "KB Written")
 foreach (id in reads- limit 5)
 printf("%-10d\t%10d\t%15d\n", id,
 reads[id]/1024, writes[id]/1024)
 delete reads
 delete writes
}

Example #5: Top process I/O (cont.)

- **% stap pid-iotop.stp**

Process ID KB Read KB Written

25553 3216 0

4272 24 0

4253 16 0

4048 16 0

4230 16 0

Process ID KB Read KB Written

25553 3328 0

19033 16 0

4253 16 0

4246 12 0

2132 8 0

Example #6: Call graph tracing

- `# para-callgraph.stp`

```
function trace(entry_p) {
    if(tid() in trace)
        printf("%s%s%s\n",thread_indent(entry_p),
                (entry_p>0?"->":"<-"),
                probefunc())
}
global trace
probe kernel.function(@1).call {
    if (pid() == stp_pid()) next # skip our own helper process
    trace[tid()] = 1
    trace(1)
}
probe kernel.function(@1).return {
    trace(-1)
    delete trace[tid()]
}
probe kernel.function(@2).call { trace(1) }
probe kernel.function(@2).return { trace(-1) }
```

Example #6: Call graph tracing (cont.)

- **% stap para-callgraph.stp sys_read '*@fs/*.c'**
 - 0 clock-applet(4325):->sys_read
 - 9 clock-applet(4325): ->fget_light
 - 13 clock-applet(4325): <-fget_light
 - 18 clock-applet(4325): ->vfs_read
 - 24 clock-applet(4325): ->rw_verify_area
 - 29 clock-applet(4325): <-rw_verify_area
 - 36 clock-applet(4325): ->do_sync_read
 - 42 clock-applet(4325): <-do_sync_read
 - 46 clock-applet(4325): <-vfs_read
 - 50 clock-applet(4325):<-sys_read

Example #7: Wake-ups due to timeouts

- Monitor timers that may cause a process to wake up
- Probe points: **poll, select, epoll, “real time” interval timer, schedule_timeout**

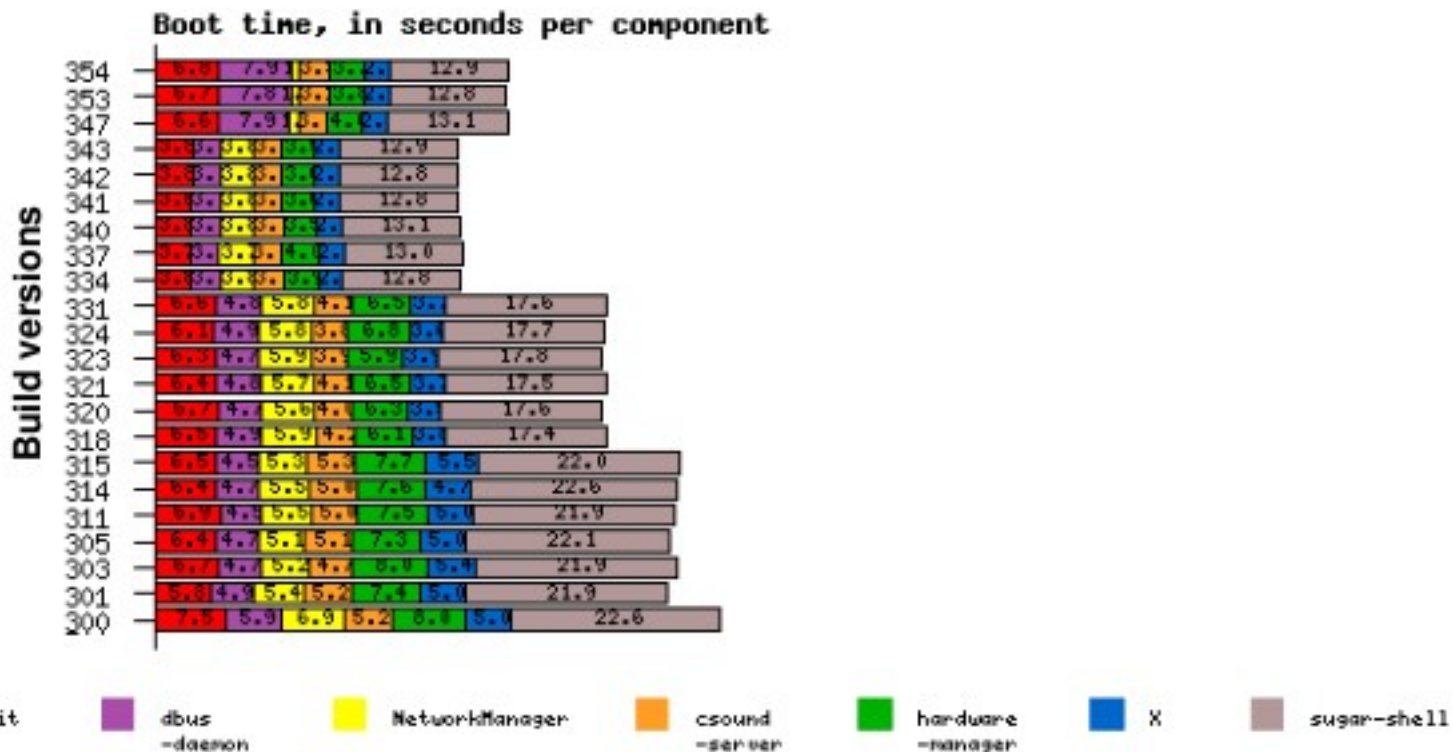
- **% pstimeouts -u**

```
uid    pid | poll  select  epoll itimer other| process
500    3463 |  116    0     0     0     0| gnome...
500    3467 |    0     0     0     0     0| scim...
500    3482 |  220    0     0     2     0| metacity
[...]
```

- See <http://dev.laptop.org/ticket/110> for the script

Example #8: Boot up time

- Measuring boot time, in seconds per component
- Components include init, dbus-daemon, NetworkManager, csound-server, hardware-manager, X, and sugar-shell
- See OLPC Tinkerbox <http://learn.laptop.org/tinderbox/>





Explore SystemTap

- This is just an introduction
- There's much, much more...
- Learn from **man stap, stapprobes, stapfuncs, stapex**
- Learn from existing scripts **testsuite/systemtap.examples/**
- Website: **<http://sources.redhat.com/systemtap/>**
- Wiki: **<http://sources.redhat.com/systemtap/wiki/>**
- War Stories: **<http://sources.redhat.com/systemtap/wiki/WarStories>**
- Join our mailing list: **systemtap@sources.redhat.com**
- And hang out at **[#systemtap](http://irc.freenode.net)** on **irc.freenode.net**



Writing SystemTap Scripts

红帽亚太区

张汉辉

eugeneteo@kernel.sg

GNOME.Asia 2008 峰会