OS Observability Tools

- "Classic" tools and their limitations
- DTrace (Solaris)
- SystemTAP (Linux)

Where we're going with this...

- Know **about** OS observation tools
  - See some **examples**
  - how to **use existing examples**
    - not our goal to become fluent/experts
  - how to **interpret** results
- **Future Homework:**
  - Stress an OS with CPU-, I/O-, Memory-, or thread-intensive programs...
  - ...use OS observation tools to *discover* and *describe* what's happening

Observing OS Activity

"All truths are easy to understand once they are discovered; the point is to discover them."

-- Galileo Galilei (1564 - 1642)

General Approaches

- Start with existing tools **common** to Linux and UNIX
  - `vmstat/mpstat/iostat (if installed for OS)`
  - `pmap/pstack`
  - `top`
- Understand some tools unique to **Solaris**:
  - `mpstat, top, prstat (like top), intrstat, truss, pfiles, pldd, ptree, dtrace`
  - `plockstat` (DTrace consumer)
  - `pstack` (for Java)
  - `pfiles` (now shows file names)
- Understand some tools unique to **Linux**:
  - `latencytop, stap` (SystemTap) (if installed)
- Can use **GUI** tools when/if they are available for each OS

Solaris Performance and Tracing Tools

Process stats

- cputrack - per-processor hw counters
- pargs – process arguments
- pflags – process flags
- pcred – process credentials
- pldd – process's library dependencies
- psig – process signal disposition
- pstack – process stack dump
- pmap – process memory map
- pfiles – open files and names
- prstat – process statistics
- ptree – process tree
- ptime – process microstate times
- pwdx – process working directory

Process control

- pgrep – grep for processes
- pkill – kill processes list
- pstop – stop processes
- prun – start processes
- prctl – view/set process resources
- pwait – wait for process
- preap – reap a zombie process

Process Tracing/ debugging

- abitrace – trace ABI interfaces
- **dtrace – trace the world**
- mdb – debug/control processes
- truss – trace functions and system calls

Kernel Tracing/ debugging

- ***dtrace*** – trace and monitor kernel
- lockstat – monitor locking statistics
- lockstat -k – profile kernel
- mdb – debug live and kernel cores

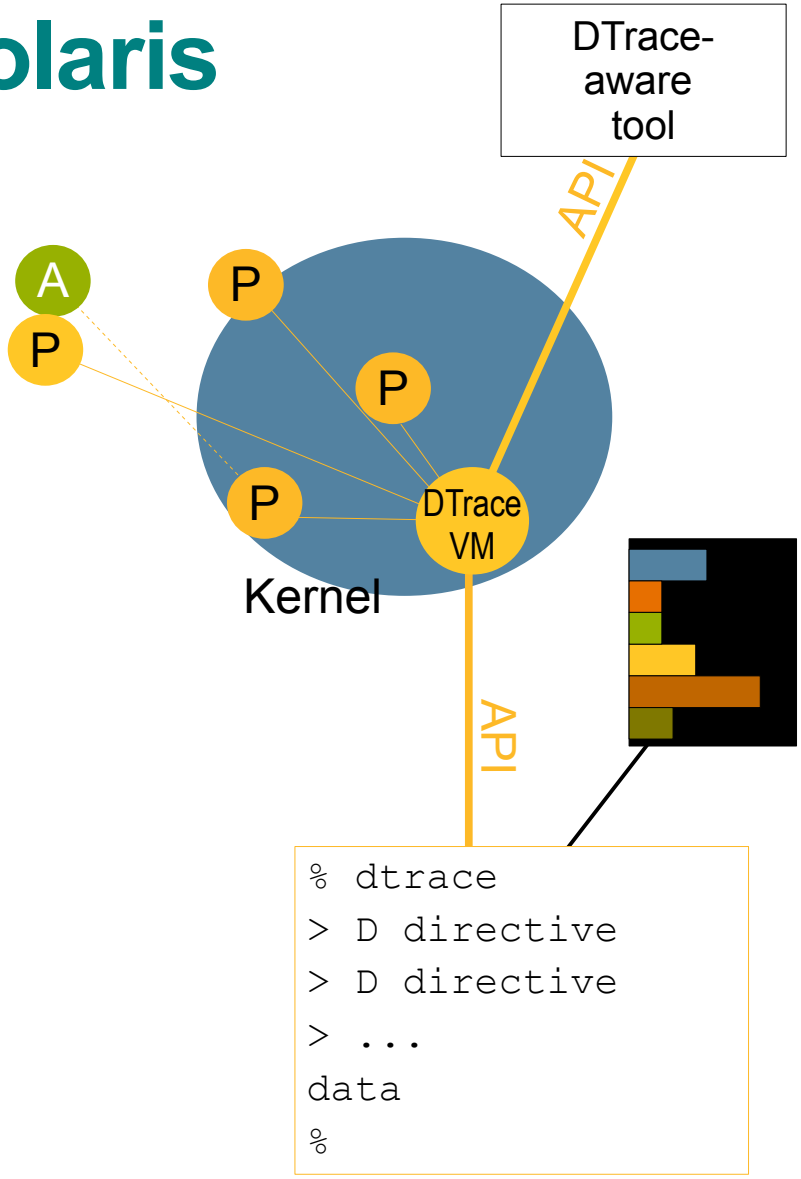
System Stats

- acctcom – process accounting
- busstat – Bus hardware counters
- cpustat – CPU hardware counters
- iostat – IO & NFS statistics
- kstat – display kernel statistics
- mpstat – processor statistics
- netstat – network statistics
- nfsstat – nfs server stats
- sar – kitchen sink utility
- vmstat – virtual memory stats

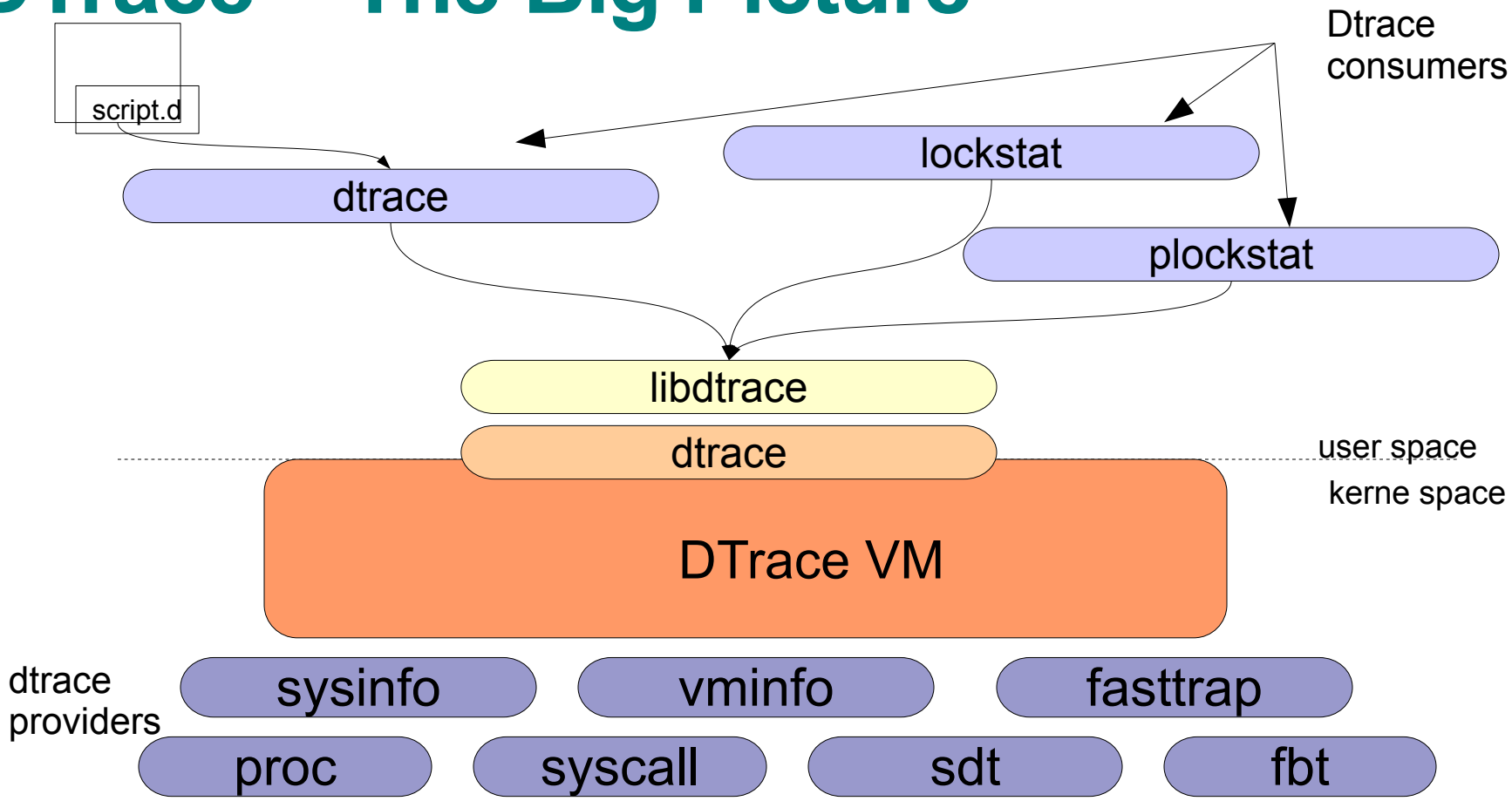
Dynamic Tracing for Solaris

Also for Apple OS X, Free BSD, ...
Designed for Production Systems

- Safe; always there
  - > No performance hit
  - > No app or OS changes
  - > No OS halt
  - > No looping
- Views system as a whole
  - > Comprehensive
  - > Extensible; scriptable
- Debug, analyze, optimize in real time



DTrace – The Big Picture



DTrace Components

- **Probes**
  - A point of instrumentation
  - Has a *name* (string), and a unique probe ID (integer)
- **Providers**
  - DTrace-specific facilities for managing probes, and the interaction of collected data with consumers
- **Consumers**
  - A process that interacts with dtrace
  - typically `dtrace`(1)
- **Using dtrace**
  - Command line – `dtrace`(1)
  - Scripts written in the 'D' language

Dtrace Components

- **Probes** (syscall::ioctl:entry) (empty fields are wildcards)
- **Providers** (syscall, fbt)
- **Consumers** (dtrace, lockstat)
- **Action** – what to do when a probe is activated
- D – scripting **language** similar to Perl and awk
- **Predicates** – conditional control for the D language
- **Aggregations** – helps identify patterns
- see **/usr/demo/dtrace** examples on Solaris systems

Providers

- DTrace has quite a few providers, e.g.:
  - The function boundary tracing (FBT) provider can dynamically instrument every function entry and return in the kernel.
  - The `syscall` provider can dynamically instrument the system call table
  - The `lockstat` provider can dynamically instrument the kernel synchronization primitives
  - The `profile` provider can add a configurable- rate profile interrupt to the system

Providers, continued

- DTrace has quite a few providers, e.g.:
  - The `vminfo` *provider can dynamically instrument the kernel "vm" statistics, used by commands such as vmstat*
  - The `sysinfo` provider can dynamically instrument the kernel "sys" statistics, used by commands such as mpstat
  - The `pid` provider can dynamically instrument application code, such as any function entry and return point (actually any instruction!)
  - The `io` provider can dynamically instrument disk I/O events (iostat)
  - And more!
- some community developers are rewriting `vmstat`, `iostat`, etc in DTrace to get more/better info.

The D language

- D is a C-like language specific to DTrace, with some constructs similar to awk(1).
- Complete access to kernel C types, complete support for ANSI-C operators.
- Rich set of built-in variables
- Anonymous arrays
- Complete access to statics and globals.
- Support for strings as first-class citizen.

D scripts, continued

- Basic structure of a D script:

```
probe description (provider:module:function:name)
/ predicate /
{
        action statements
}
```

- For example, a script to trace the executable name upon entry of each system call:

```
#!/usr/sbin/dtrace -s
syscall:::entry
{
        trace(execname);
}
```

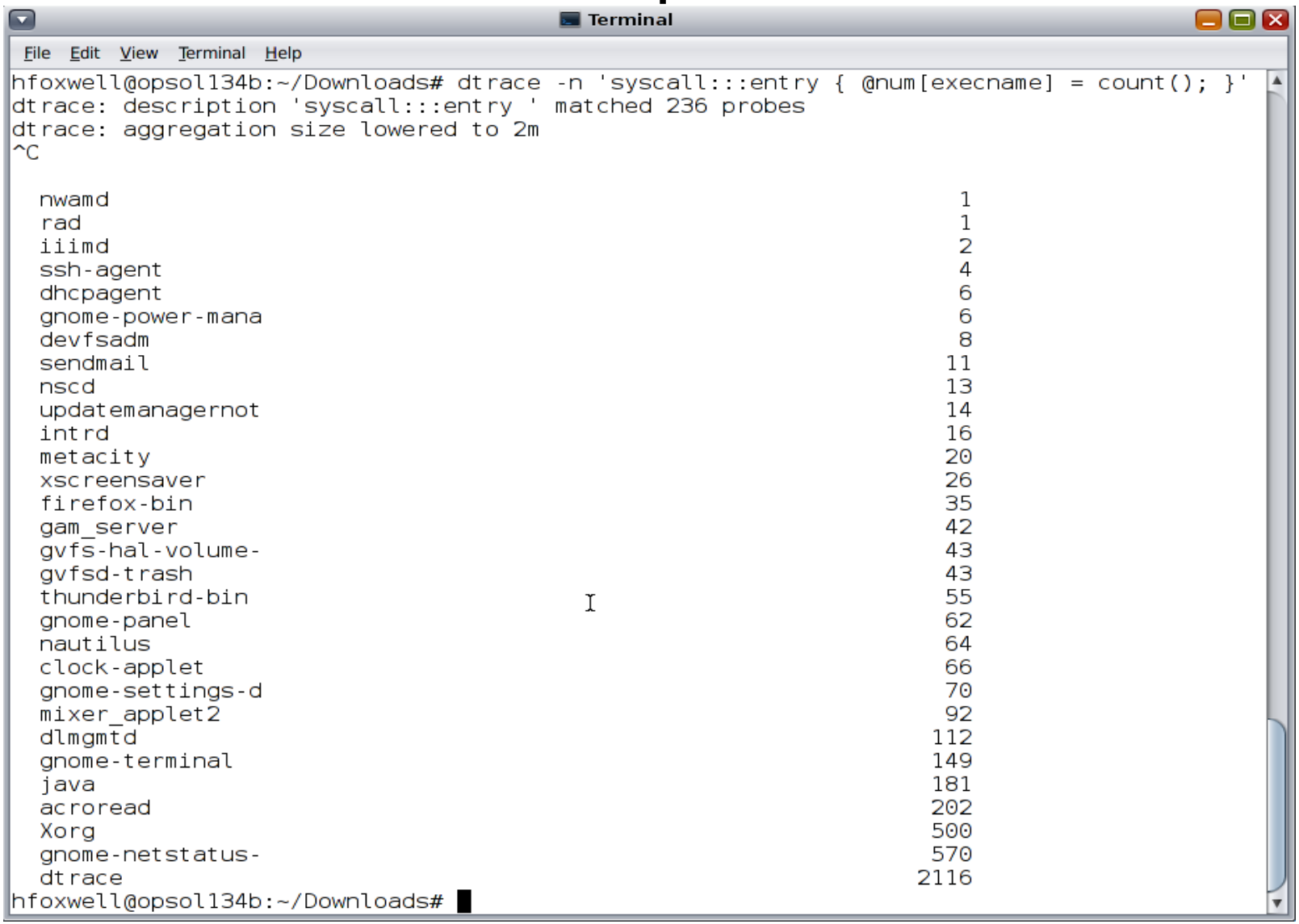
Some Examples: DTrace

Terminal

File Edit View Terminal Help

```
# Syscall count by program,
dtrace -n 'syscall:::entry { @num[execname] = count(); }'

# Syscall count by syscall,
dtrace -n 'syscall:::entry { @num[probefunc] = count(); }'

# Syscall count by process,
dtrace -n 'syscall:::entry { @num[pid,execname] = count(); }'

# Read bytes by process,
dtrace -n 'sysinfo:::readch { @bytes[execname] = sum(arg0); }'

# Write bytes by process,
dtrace -n 'sysinfo:::writech { @bytes[execname] = sum(arg0); }'

# Read size distribution by process,
dtrace -n 'sysinfo:::readch { @dist[execname] = quantize(arg0); }'

# Write size distribution by process,
dtrace -n 'sysinfo:::writech { @dist[execname] = quantize(arg0); }'

# Disk size by process,
dtrace -n 'io:::start { printf("%d %s %d",pid,execname,args[0]->b_bcount); }'
--More--(8%)
```

Some Examples: DTrace



Some Examples:DTrace

Terminal

File Edit View Terminal Help

```
hfoxwell@opsol134b:~/Downloads# dtrace -n 'sysinfo:::readch { @dist[execname] = quantize(arg0); }'
dtrace: description 'sysinfo:::readch ' matched 4 probes
^C

  firefox-bin
           value  ------------- Distribution ------------- count
               0 |                                         0
               1 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 3
               2 |                                         0

  gnome-settings-d
           value  ------------- Distribution ------------- count
              16 |                                         0
              32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
              64 |                                         0

  gnome-power-mana
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@                  1
               1 |                                         0
               2 |                                         0
               4 |                                         0
               8 |                                         0
              16 |                                         0
              32 |@@@@@@@@@@@@@@@@@@@@@@@                  1
              64 |                                         0

  thunderbird-bin
           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@                               3
               1 |@@@@@@@@@@@@@@@@@@@@@@@@@@               7
               2 |                                         0
               4 |                                         0
               8 |                                         0
```

Some Examples: DTrace

Terminal

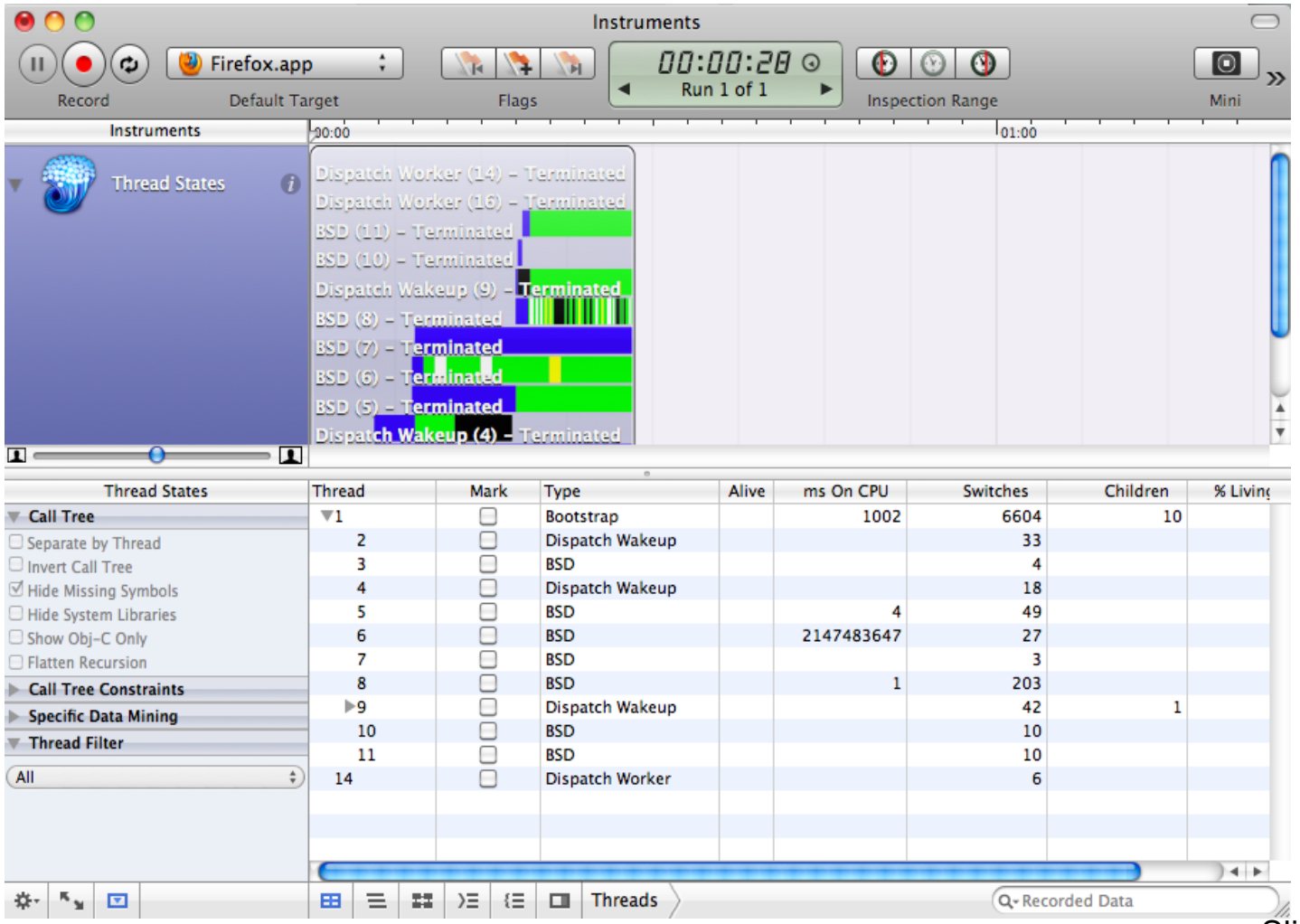
File Edit View Terminal Help

```
hfoxwell@opsol134b:~/Downloads/DTraceToolkit-0.99/Cpu# ls ..
Apps          dtruss       hotkernel    Java         Misc         procsystime  Snippits
Bin           dvmstat      hotuser      JavaScript   Net          Python       statsnoop
Code          errinfo      Include      Kernel       Notes        README       System
Cpu           Examples     install      License      opensnoop    Ruby         Tcl
dexplorer     execsnoop    iopattern    Locks        Perl         rwsnoop      User
Disk          FS           iosnoop      Man          Php          rwtop        Version
Docs          Guide        iotop        Mem          Proc         Shell        Zones
hfoxwell@opsol134b:~/Downloads/DTraceToolkit-0.99/Cpu# ls
cputypes.d     dispqlen.d    intoncpu.d     loads.d        runocc.d
cpuwalk.d      intbycpu.d    inttimes.d     Readme         xcallsbypid.d
hfoxwell@opsol134b:~/Downloads/DTraceToolkit-0.99/Cpu# ./dispqlen.d
Sampling... Hit Ctrl-C to end.
^C
 CPU 1
           value  ------------- Distribution ------------- count
             < 0 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 7465
               1 |@@                                       316
               2 |                                         88
               3 |                                         0

 CPU 0
           value  ------------- Distribution ------------- count
             < 0 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 7372
               1 |@@                                       481
               2 |                                         13
               3 |                                         3
               4 |                                         0


hfoxwell@opsol134b:~/Downloads/DTraceToolkit-0.99/Cpu#
```

And for our OS X brethren...

- Instruments
  - installed with ADC Dev Tools (**XCode**)
  - much of probe/monitoring based on OS X implementation of DTrace

And for our OS X brethren...



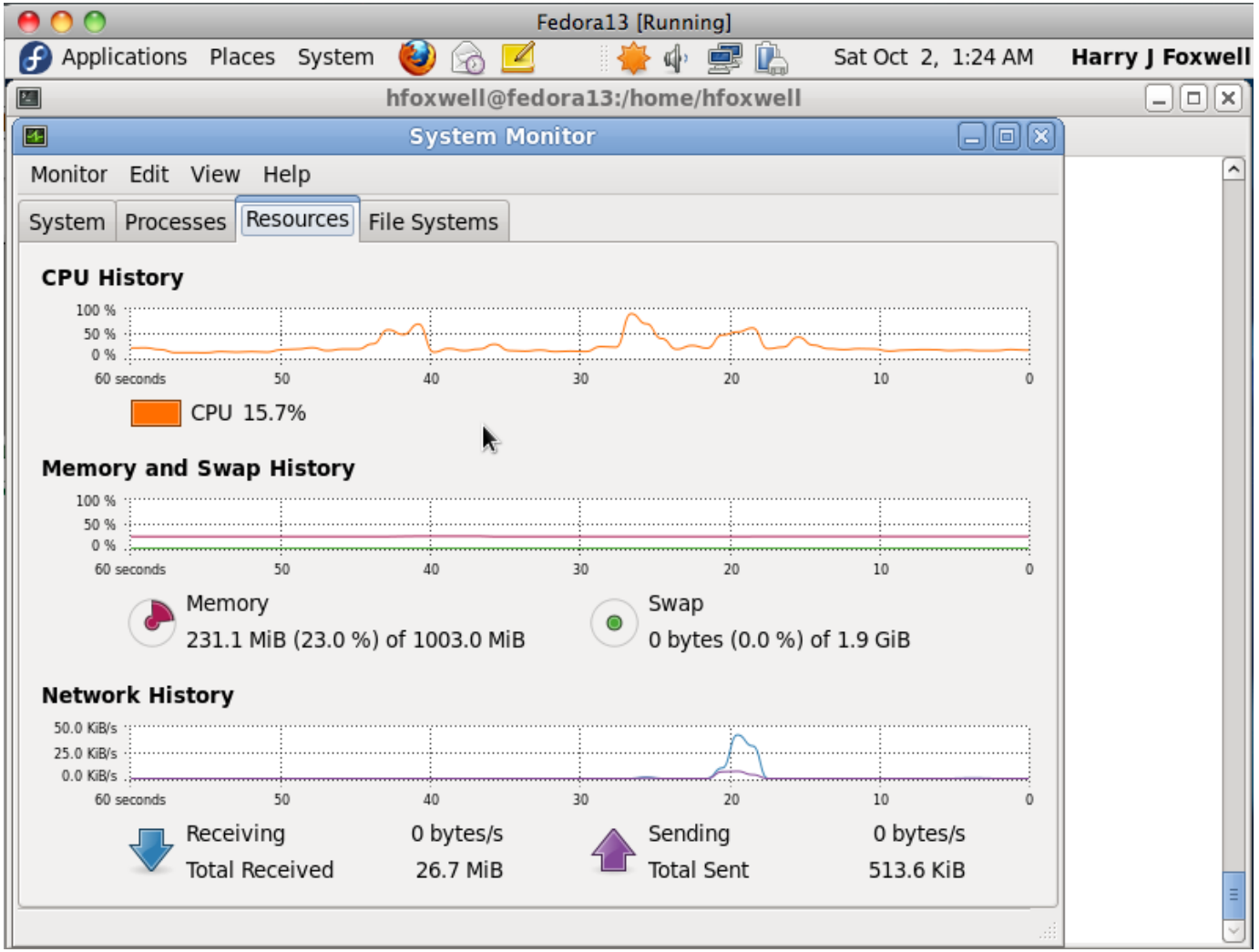
Linux Observability Tools

- `strace`
  - process-oriented observation tool
    like Solaris 'truss'
  - strace progname
  - strace -o outfile progname
  - strace -e trace=syscallname progname
  - strace -c progname
  - delivered with Linux kernel
  - output can (must) be post-processed
  - use **in addition** to traditional tools
    top, vmstat, ps, /proc

Linux Observability Tools

- KProbes
  - IBM-supported open source contribution to Linux 2.6 kernel
  - inserts *breakpoints* into running kernel at specified address
  - can modify registers and global data
  - install files, patch kernel to accept printk requests
  - uses C syntax
  - get address of desired kernel inspection point, write and register probe (in C), write data handler
  - no safety checking, needs view of instrumented codepath, can't see local variables, ...

GNOME System Monitor (Linux AND Solaris)



SystemTAP

- Modeled after DTrace!
  - **GPL** tool developed by IBM, Intel, Red Hat, and Oracle for Linux dynamic tracing
  - Primarily for kernel development & tuning
    - limited application / user space instrumentation
  - Basically a safety wrapper around kprobes
    - has default (safe) and “guru” mode (can change data)
    - some protection: no div by 0, no bad memory refs, limited recursion, no infinite loops.
  - /usr/bin/stap myscript.stp
    - CLI/script like C, stapgui.sourceforge.net

SystemTap for Linux

- safety wrapper around KProbes
- can probe kernel and user space
  - but no specialties for PHP, Java, or other dynamic environments
- generates C code, compiles into kernel module, loads & runs
- designed for low/no overhead

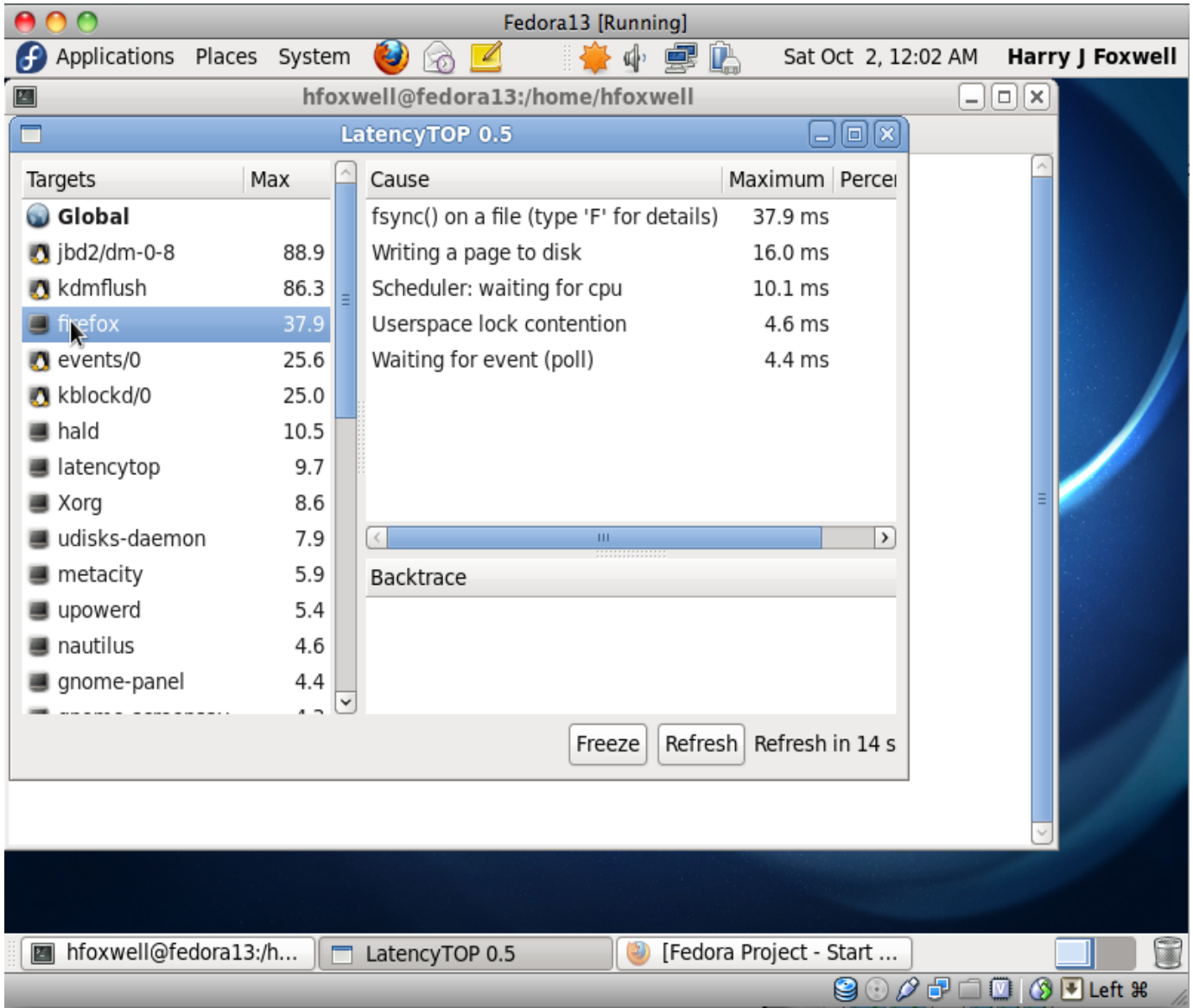
SystemTAP

- edit/create a script (myscript.stp)
  - transformed to C, compiled to loadable kernel module, runs & collects data, upon ^C sends data to sdtout
- To install & verify on RHEL5 (included with RHEL 5.4)
  - yum install systemtap-testsuite
  - cd /usr/share/systemtap/testsuite
  - make installcheck
- http://sourceware.org/systemtap/wiki
- http://sourceware.org/systemtap/langref/

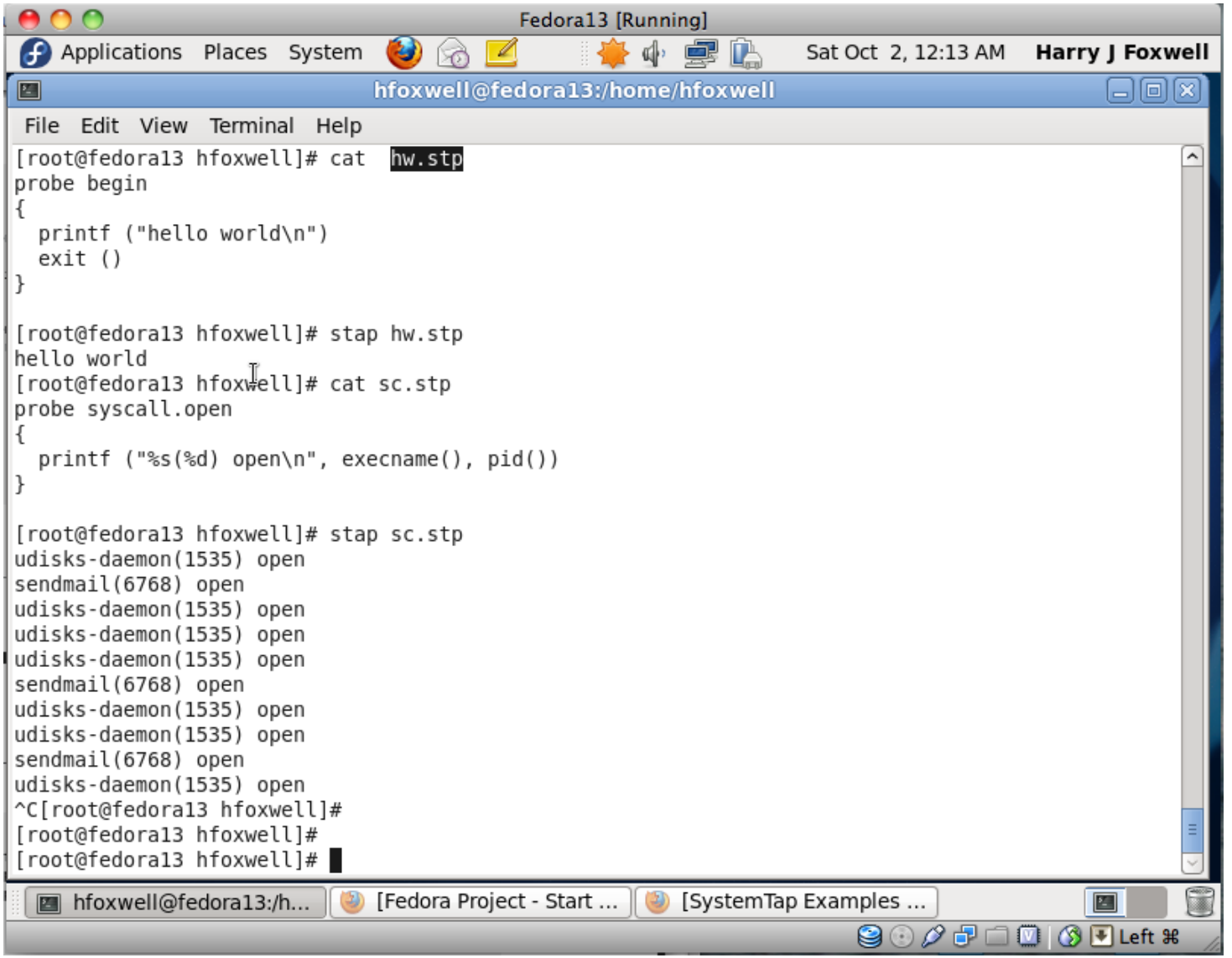
SystemTap for Linux

- See DTrace/SystemTAP comparison
- http://sources.redhat.com/systemtap/wiki/SystemtapDtraceComparison
- C/awk probe language
  - specifies a probe, and a probe handler
  - when probe is 'hit', handler suspends monitored thread, executes handler instructions
  - very similar to DTrace, but...
    - must be careful not to loop in handler, block, or grab & keep locks
    - can write anywhere in kernel memory, directly call any kernel subroutine

Some Examples: Linux `latencytop`



Some Examples: Linux `systemtap`



Some Examples: Linux `systemtap`

How does it work?

1. write or choose a script describing what you want to observe
2. stap translates it into a kernel module
3. stap loads the module and communicates with it
4. just wait for your data

Some Examples: Linux `systemtap`

The five stap passes

```
# stap -v test.stp
Pass 1: parsed user script and 38 library script(s) in
    150usr/20sys/183real ms.
Pass 2: analyzed script: 1 probe(s), 5 function(s), 14
    embed(s), 0 global(s) in 110usr/110sys/242real ms.
Pass 3: translated to C into
    "/tmp/stapEjEd0T/stap_6455011c477a19ec8c7bbd5ac12a9cd0_13
     in 0usr/0sys/0real ms.
Pass 4: compiled C into
    "stap_6455011c477a19ec8c7bbd5ac12a9cd0_13608.ko" in
    1250usr/240sys/1685real ms.
Pass 5: starting run.
[...script output goes here...]
Pass 5: run completed in 20usr/30sys/4204real ms.
```

Some Examples: Linux `systemtap`

SystemTap probe points examples

SystemTap is all about executing certain actions when hitting certain probe points.

- `syscall.read`
  - when entering read() system call
- `syscall.close.return`
  - when returning from the close() system call
- `module("floppy").function("*")`
  - when entering any function from the "floppy" module
- `kernel.function("*@net/socket.c").return`
  - when returning from any function in file `net/socket.c`
- `kernel.statement("*@kernel/sched.c:2917")`
  - when hitting line 2917 of file `kernel/sched.c`

Some Examples: Linux `systemtap`

More probe points examples

- `timer.ms(200)`
  - every 200 milliseconds
- `process("/bin/ls").function("*")`
  - when entering any function in /bin/ls (not its libraries or syscalls)
- `process("/lib/libc.so.6").function("*malloc*")`
  - when entering any glibc function which has "malloc" in its name
- `kernel.function("*init*")`, `kernel.function("*exit*").return`
  - when entering any kernel function which has "init" in its name or returning from any kernel function which has "exit" in its name

RTFM for more (`man stapprobes`).

Some Examples: Linux `systemtap`

SystemTap programming language

- mostly C-style syntax with a feeling of awk
- builtin associative arrays
- builtin aggregates of statistical data
  - very easy to collect data and do statistics on it (average, min, max, count,...)
- many helper functions (builtin and in tapsets)

RTFM: *SystemTap Language Reference* shipped with SystemTap (langref.pdf)

Some Examples: Linux `systemtap`

Some helper functions you'll see a lot

`pid()`	which process is this?
`uid()`	which user is running this?
`execname()`	what is the name of this process?
`tid()`	which thread is this?
`gettimeofday_s()`	epoch time in seconds
`probefunc()`	what function are we in?
`print_backtrace()`	figure out how we ended up here

There are many many more. RTFM (`man stapfuncs`) and explore `/usr/share/systemtap/tapset/`.

Some Examples: Linux `systemtap`

Example 1: trace processes execution

Listing 1: exec.stp

```
1 probe syscall.exec* {
2     printf("exec %s %s\n", execname(), argstr)
3 }
```

```
$ stap -L 'syscall.exec*'
syscall.execve name:string filename:string
   args:string argstr:string

# stap exec.stp
exec gnome-terminal /bin/bash
exec bash /usr/bin/id -gn
exec bash /usr/bin/id -un
exec bash /bin/uname -s
exec bash /bin/uname -r
```

Some Examples: Linux `systemtap`

References and questions

- SystemTap wiki: http://sourceware.org/systemtap/wiki
- lot of excellent documentation included:
  - `man {stap,stapprobes,stapfuncs,stapvars,...}`
  - file:///usr/share/doc/systemtap*
- there is probably already a script to do what you want: http://sourceware.org/systemtap/examples/
- systemtap@sources.redhat.com
- irc://chat.freenode.net/#systemtap

OS Resources

- **General**
  - http://bhami.com/rosetta.html
- **Linux**
  - Linux Performance and Tuning Guidelines
    - http://www.redbooks.ibm.com/abstracts/redp4285.html
    - http://www.latencytop.org/
  - SystemTap
    - http://www.ibm.com/developerworks/linux/library/l-systemtap/index.html
    - http://sourceware.org/systemtap/
- **Solaris**
  - Solaris Performance and Tools book by McDougall, Mauro, and Gregg
  - DTrace
    - http://hub.opensolaris.org/bin/view/Community+Group+dtrace/WebHome
    - http://blogs.sun.com/brendan
    - http://www.oracle.com/technetwork/server-storage/solaris/dtrace-tutorial-142317.html