

# 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

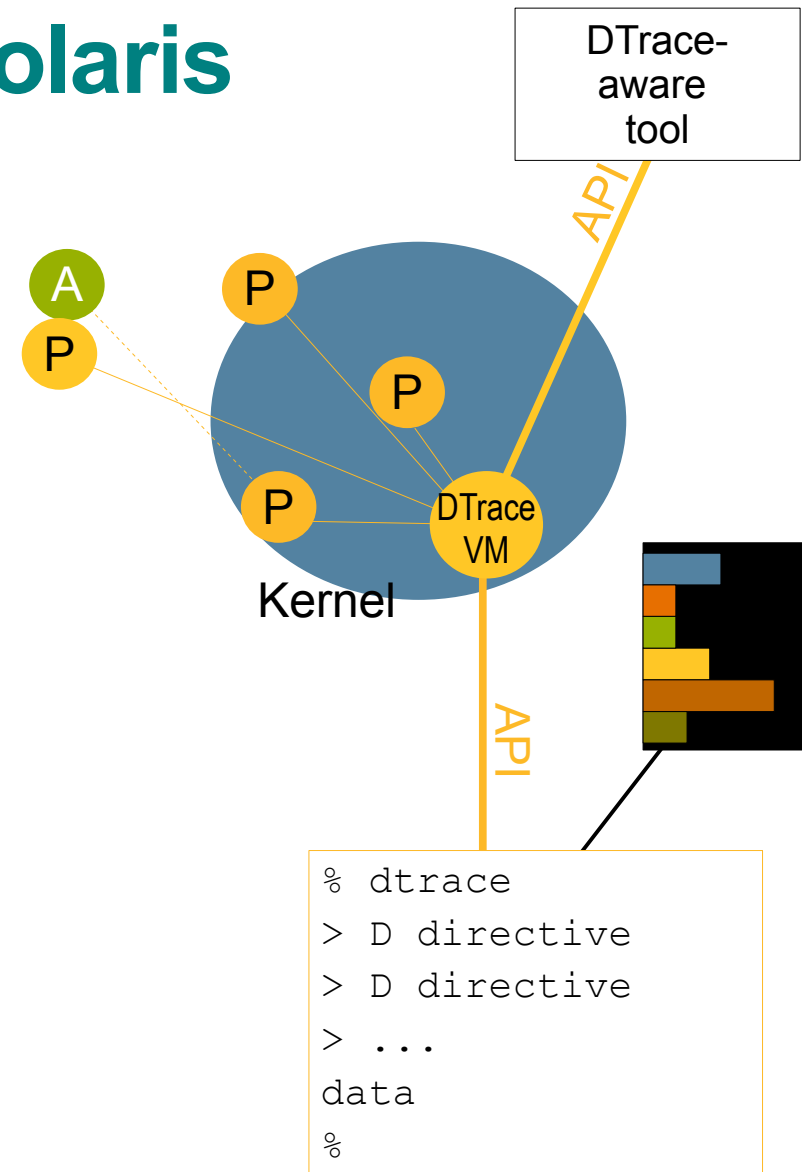
## System Stats

- 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

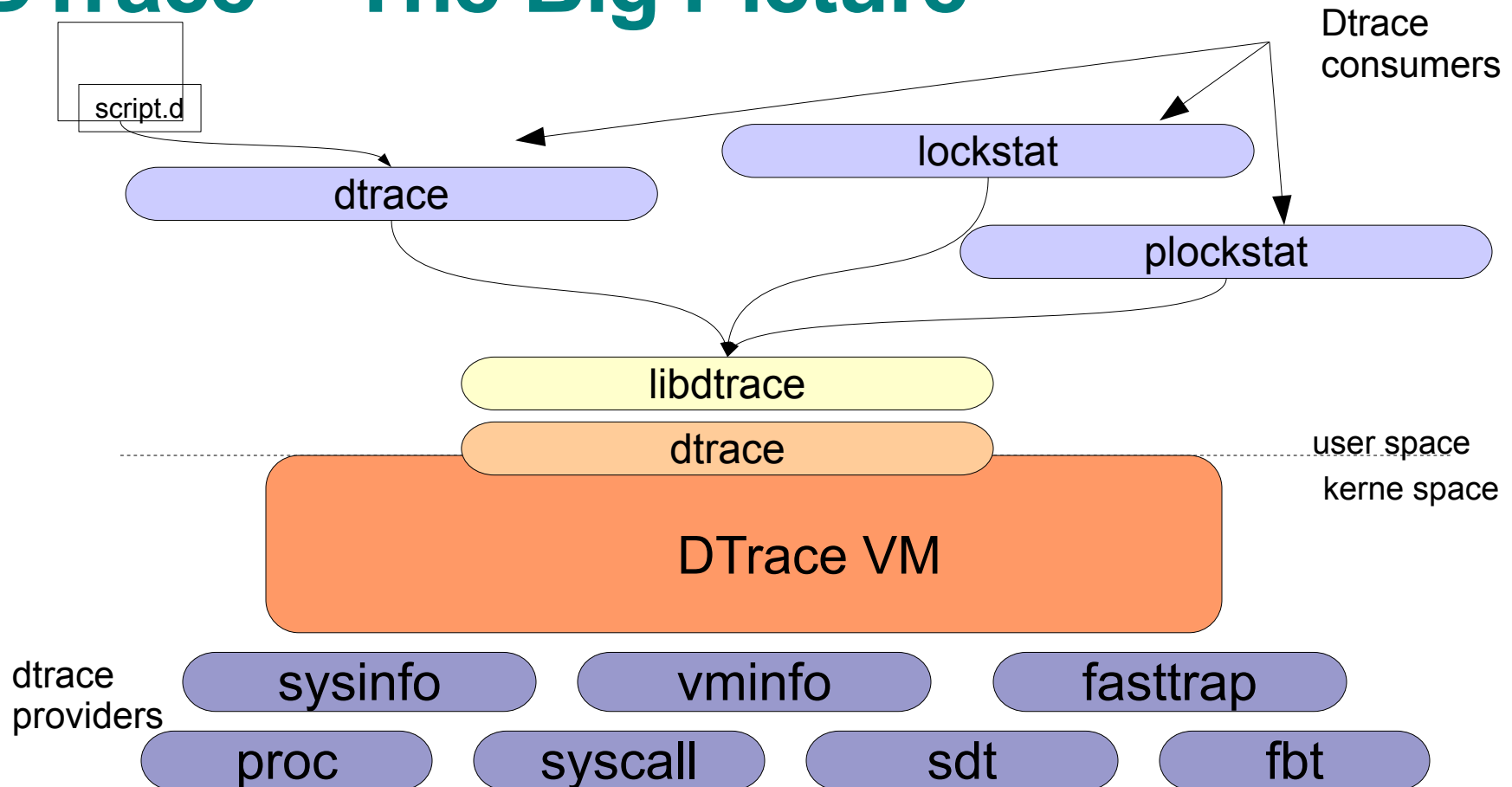
# Dynamic Tracing for Solaris

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

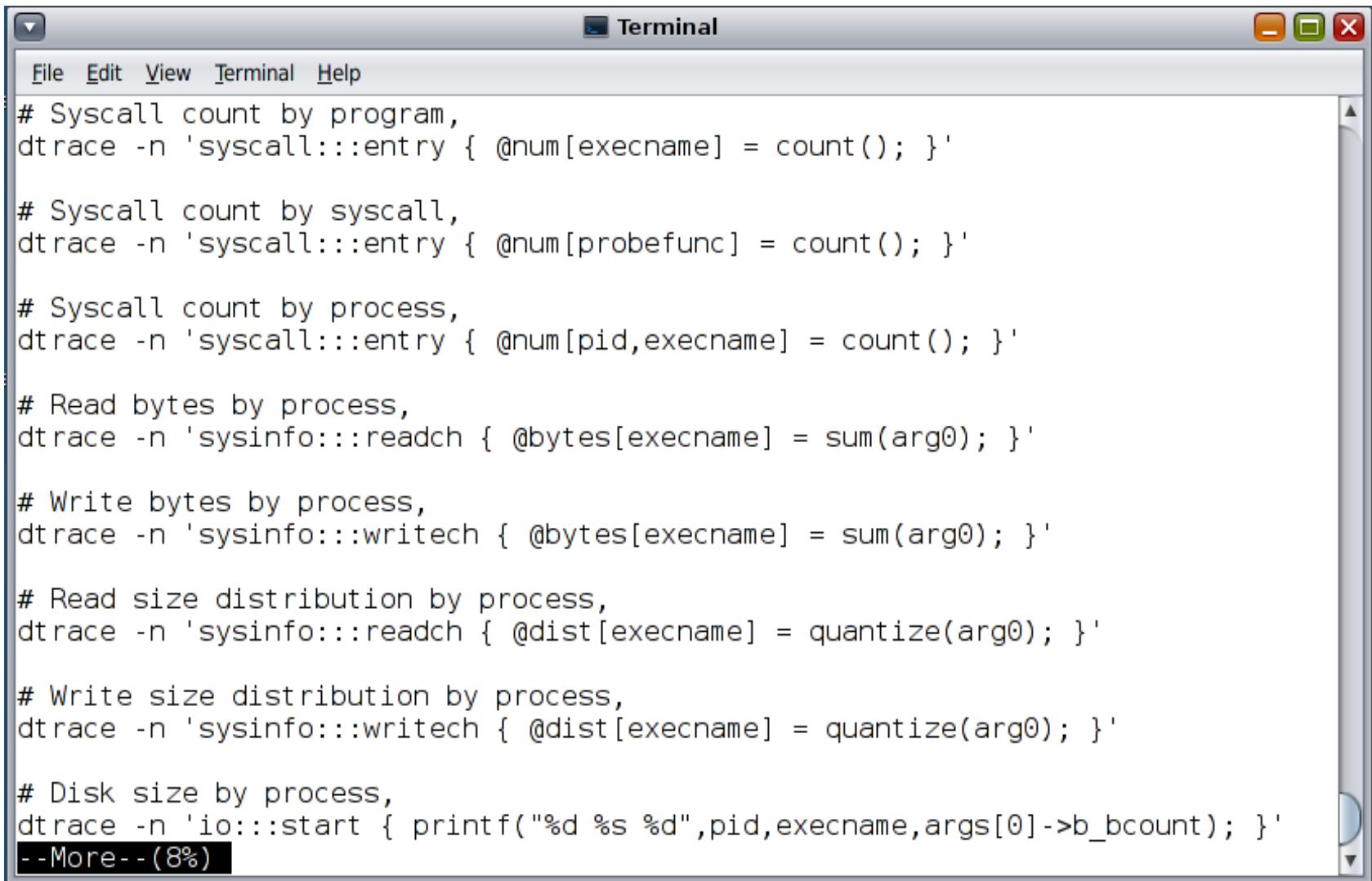
- Basic structure of a D script:

```
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

```

Terminal
File Edit View Terminal Help
hfoxwell@opsol134b:~/Downloads# dtrace -n 'syscall:::entry { @num[execname] = count(); }'
dtrace: description 'syscall:::entry' matched 236 probes
dtrace: aggregation size lowered to 2m
^C

nwamd 1
rad 1
iiimd 2
ssh-agent 4
dhcpgent 6
gnome-power-mana 6
devfsadm 8
sendmail 11
nscd 13
updatemanagernot 14
intrd 16
metacity 20
xscreensaver 26
firefox-bin 35
gam_server 42
gvfs-hal-volume- 43
gvfsd-trash 43
thunderbird-bin 55
gnome-panel 62
nautilus 64
clock-applet 66
gnome-settings-d 70
mixer_applet2 92
dlmgtmd 112
gnome-terminal 149
java 181
acroread 202
Xorg 500
gnome-netstatus- 570
dtrace 2116
hfoxwell@opsol134b:~/Downloads#

```

## Some Examples:DTrace

```

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 |
  1 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 3
  2 |
  3 |
  4 |
  5 |
  6 |
  7 |
  8 |
  9 |
 10 |
 11 |
 12 |
 13 |
 14 |
 15 |
 16 |
 17 |
 18 |
 19 |
 20 |
 21 |
 22 |
 23 |
 24 |
 25 |
 26 |
 27 |
 28 |
 29 |
 30 |
 31 |
 32 |
 33 |
 34 |
 35 |
 36 |
 37 |
 38 |
 39 |
 40 |
 41 |
 42 |
 43 |
 44 |
 45 |
 46 |
 47 |
 48 |
 49 |
 50 |
 51 |
 52 |
 53 |
 54 |
 55 |
 56 |
 57 |
 58 |
 59 |
 60 |
 61 |
 62 |
 63 |
 64 |
 65 |
 66 |
 67 |
 68 |
 69 |
 70 |
 71 |
 72 |
 73 |
 74 |
 75 |
 76 |
 77 |
 78 |
 79 |
 80 |
 81 |
 82 |
 83 |
 84 |
 85 |
 86 |
 87 |
 88 |
 89 |
 90 |
 91 |
 92 |
 93 |
 94 |
 95 |
 96 |
 97 |
 98 |
 99 |
100 |
101 |
102 |
103 |
104 |
105 |
106 |
107 |
108 |
109 |
110 |
111 |
112 |
113 |
114 |
115 |
116 |
117 |
118 |
119 |
120 |
121 |
122 |
123 |
124 |
125 |
126 |
127 |
128 |
129 |
130 |
131 |
132 |
133 |
134 |
135 |
136 |
137 |
138 |
139 |
140 |
141 |
142 |
143 |
144 |
145 |
146 |
147 |
148 |
149 |
150 |
151 |
152 |
153 |
154 |
155 |
156 |
157 |
158 |
159 |
160 |
161 |
162 |
163 |
164 |
165 |
166 |
167 |
168 |
169 |
170 |
171 |
172 |
173 |
174 |
175 |
176 |
177 |
178 |
179 |
180 |
181 |
182 |
183 |
184 |
185 |
186 |
187 |
188 |
189 |
190 |
191 |
192 |
193 |
194 |
195 |
196 |
197 |
198 |
199 |
200 |
201 |
202 |
203 |
204 |
205 |
206 |
207 |
208 |
209 |
210 |
211 |
212 |
213 |
214 |
215 |
216 |
217 |
218 |
219 |
220 |
221 |
222 |
223 |
224 |
225 |
226 |
227 |
228 |
229 |
230 |
231 |
232 |
233 |
234 |
235 |
236 |
237 |
238 |
239 |
240 |
241 |
242 |
243 |
244 |
245 |
246 |
247 |
248 |
249 |
250 |
251 |
252 |
253 |
254 |
255 |
256 |
257 |
258 |
259 |
260 |
261 |
262 |
263 |
264 |
265 |
266 |
267 |
268 |
269 |
270 |
271 |
272 |
273 |
274 |
275 |
276 |
277 |
278 |
279 |
280 |
281 |
282 |
283 |
284 |
285 |
286 |
287 |
288 |
289 |
290 |
291 |
292 |
293 |
294 |
295 |
296 |
297 |
298 |
299 |
300 |
301 |
302 |
303 |
304 |
305 |
306 |
307 |
308 |
309 |
310 |
311 |
312 |
313 |
314 |
315 |
316 |
317 |
318 |
319 |
320 |
321 |
322 |
323 |
324 |
325 |
326 |
327 |
328 |
329 |
330 |
331 |
332 |
333 |
334 |
335 |
336 |
337 |
338 |
339 |
340 |
341 |
342 |
343 |
344 |
345 |
346 |
347 |
348 |
349 |
350 |
351 |
352 |
353 |
354 |
355 |
356 |
357 |
358 |
359 |
360 |
361 |
362 |
363 |
364 |
365 |
366 |
367 |
368 |
369 |
370 |
371 |
372 |
373 |
374 |
375 |
376 |
377 |
378 |
379 |
380 |
381 |
382 |
383 |
384 |
385 |
386 |
387 |
388 |
389 |
390 |
391 |
392 |
393 |
394 |
395 |
396 |
397 |
398 |
399 |
400 |
401 |
402 |
403 |
404 |
405 |
406 |
407 |
408 |
409 |
410 |
411 |
412 |
413 |
414 |
415 |
416 |
417 |
418 |
419 |
420 |
421 |
422 |
423 |
424 |
425 |
426 |
427 |
428 |
429 |
430 |
431 |
432 |
433 |
434 |
435 |
436 |
437 |
438 |
439 |
440 |
441 |
442 |
443 |
444 |
445 |
446 |
447 |
448 |
449 |
450 |
451 |
452 |
453 |
454 |
455 |
456 |
457 |
458 |
459 |
460 |
461 |
462 |
463 |
464 |
465 |
466 |
467 |
468 |
469 |
470 |
471 |
472 |
473 |
474 |
475 |
476 |
477 |
478 |
479 |
480 |
481 |
482 |
483 |
484 |
485 |
486 |
487 |
488 |
489 |
490 |
491 |
492 |
493 |
494 |
495 |
496 |
497 |
498 |
499 |
500 |
501 |
502 |
503 |
504 |
505 |
506 |
507 |
508 |
509 |
510 |
511 |
512 |
513 |
514 |
515 |
516 |
517 |
518 |
519 |
520 |
521 |
522 |
523 |
524 |
525 |
526 |
527 |
528 |
529 |
530 |
531 |
532 |
533 |
534 |
535 |
536 |
537 |
538 |
539 |
540 |
541 |
542 |
543 |
544 |
545 |
546 |
547 |
548 |
549 |
550 |
551 |
552 |
553 |
554 |
555 |
556 |
557 |
558 |
559 |
560 |
561 |
562 |
563 |
564 |
565 |
566 |
567 |
568 |
569 |
570 |
571 |
572 |
573 |
574 |
575 |
576 |
577 |
578 |
579 |
580 |
581 |
582 |
583 |
584 |
585 |
586 |
587 |
588 |
589 |
590 |
591 |
592 |
593 |
594 |
595 |
596 |
597 |
598 |
599 |
600 |
601 |
602 |
603 |
604 |
605 |
606 |
607 |
608 |
609 |
610 |
611 |
612 |
613 |
614 |
615 |
616 |
617 |
618 |
619 |
620 |
621 |
622 |
623 |
624 |
625 |
626 |
627 |
628 |
629 |
630 |
631 |
632 |
633 |
634 |
635 |
636 |
637 |
638 |
639 |
640 |
641 |
642 |
643 |
644 |
645 |
646 |
647 |
648 |
649 |
650 |
651 |
652 |
653 |
654 |
655 |
656 |
657 |
658 |
659 |
660 |
661 |
662 |
663 |
664 |
665 |
666 |
667 |
668 |
669 |
670 |
671 |
672 |
673 |
674 |
675 |
676 |
677 |
678 |
679 |
680 |
681 |
682 |
683 |
684 |
685 |
686 |
687 |
688 |
689 |
690 |
691 |
692 |
693 |
694 |
695 |
696 |
697 |
698 |
699 |
700 |
701 |
702 |
703 |
704 |
705 |
706 |
707 |
708 |
709 |
710 |
711 |
712 |
713 |
714 |
715 |
716 |
717 |
718 |
719 |
720 |
721 |
722 |
723 |
724 |
725 |
726 |
727 |
728 |
729 |
730 |
731 |
732 |
733 |
734 |
735 |
736 |
737 |
738 |
739 |
740 |
741 |
742 |
743 |
744 |
745 |
746 |
747 |
748 |
749 |
750 |
751 |
752 |
753 |
754 |
755 |
756 |
757 |
758 |
759 |
760 |
761 |
762 |
763 |
764 |
765 |
766 |
767 |
768 |
769 |
770 |
771 |
772 |
773 |
774 |
775 |
776 |
777 |
778 |
779 |
780 |
781 |
782 |
783 |
784 |
785 |
786 |
787 |
788 |
789 |
790 |
791 |
792 |
793 |
794 |
795 |
796 |
797 |
798 |
799
```



# Some Examples: DTrace

```

Terminal
File Edit View Terminal Help
hfoxwell@opsol134b:~/Downloads/DTraceToolkit-0.99/Cpu# ls ..
Apps      dtmstat      hotkernel    Java         Misc         procsystime  Snippets
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          rwttop       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

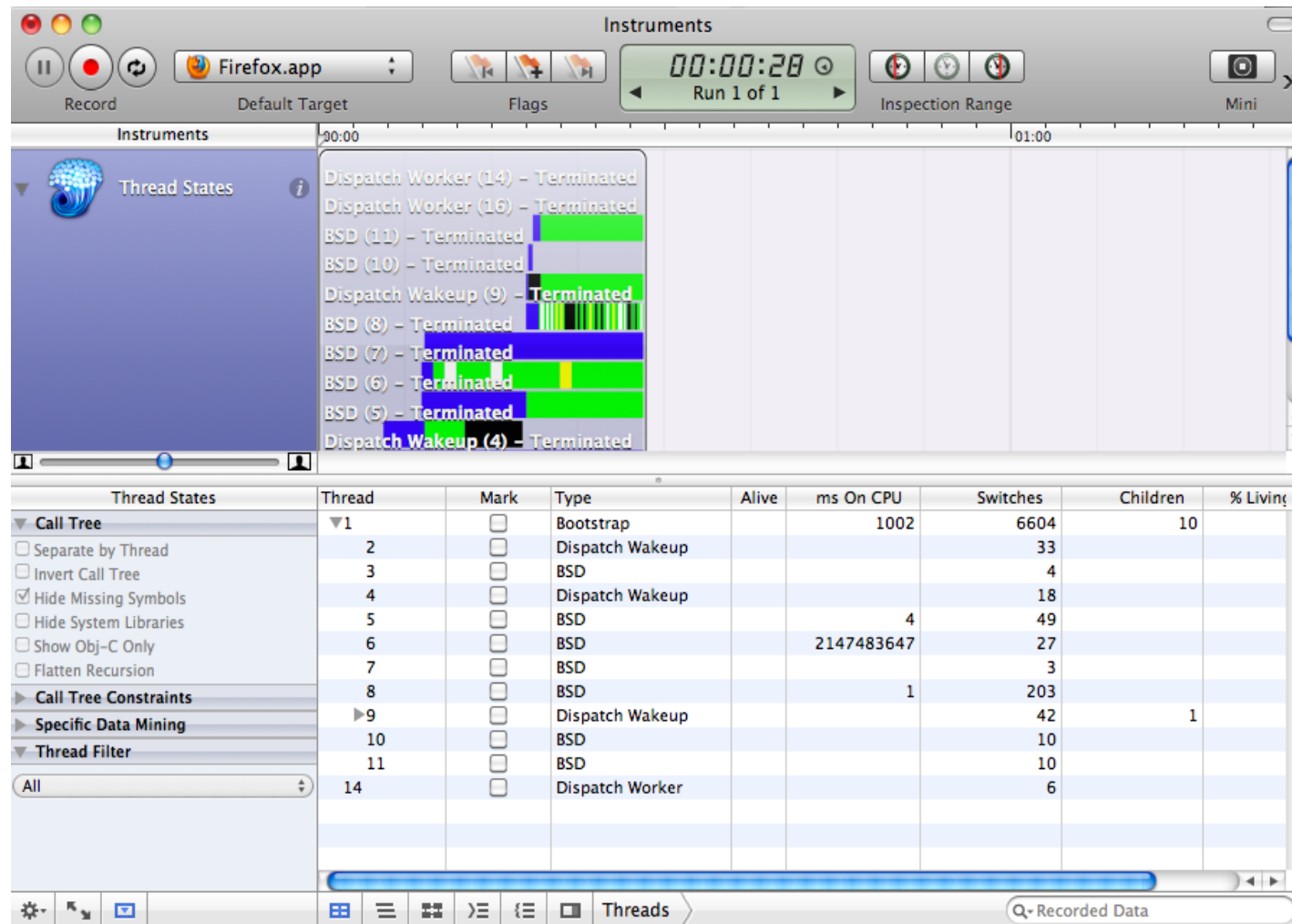
hfoxwell@opsol134b:~/Downloads/DTraceToolkit-0.99/Cpu#

```

# 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...



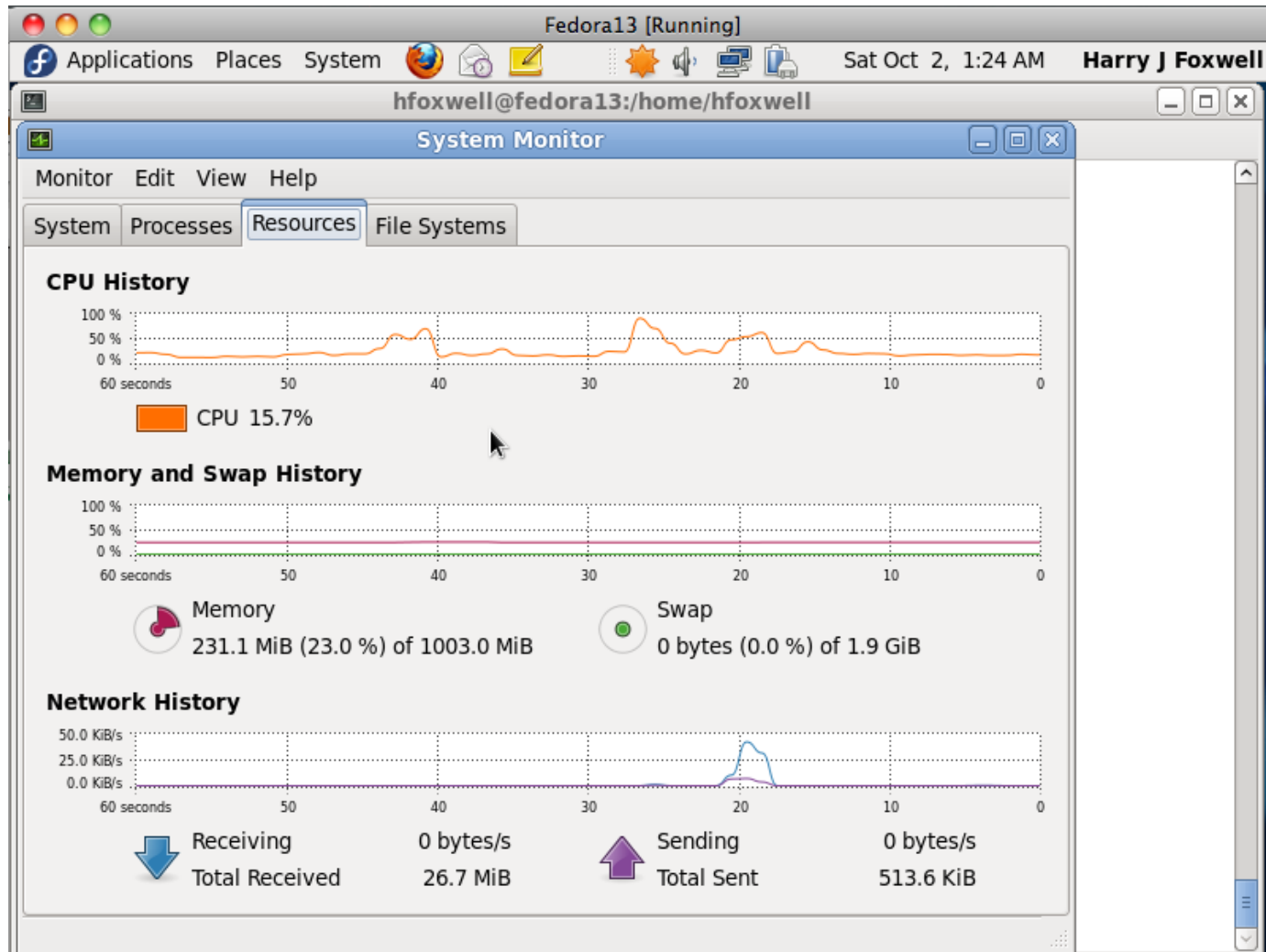
# Linux Observability Tools

- `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](http://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



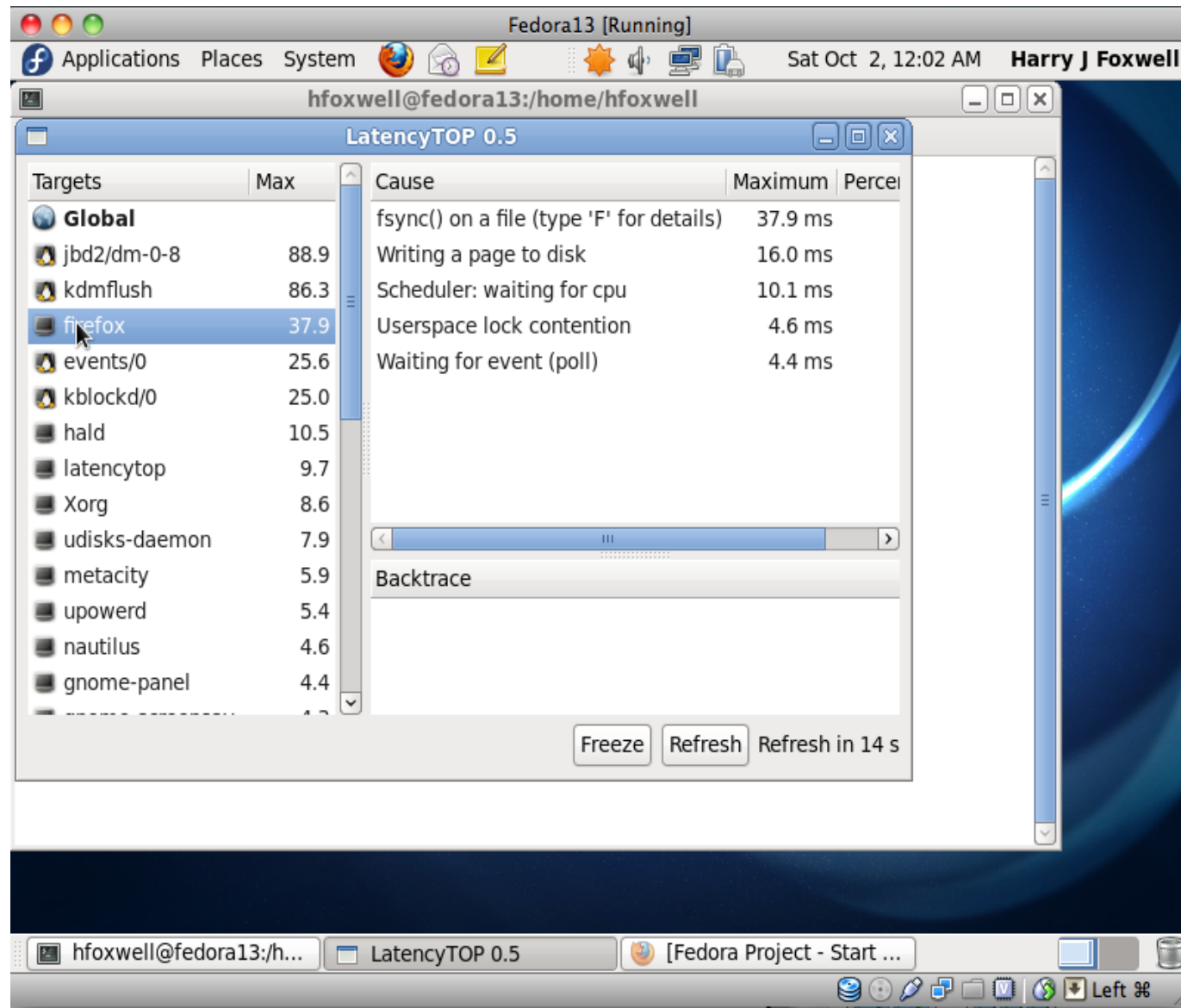
# SystemTAP

- edit/create a script (myscript.stp)
  - transformed to C, compiled to loadable kernel module, runs & collects data, upon ^C sends data to stdout
- 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

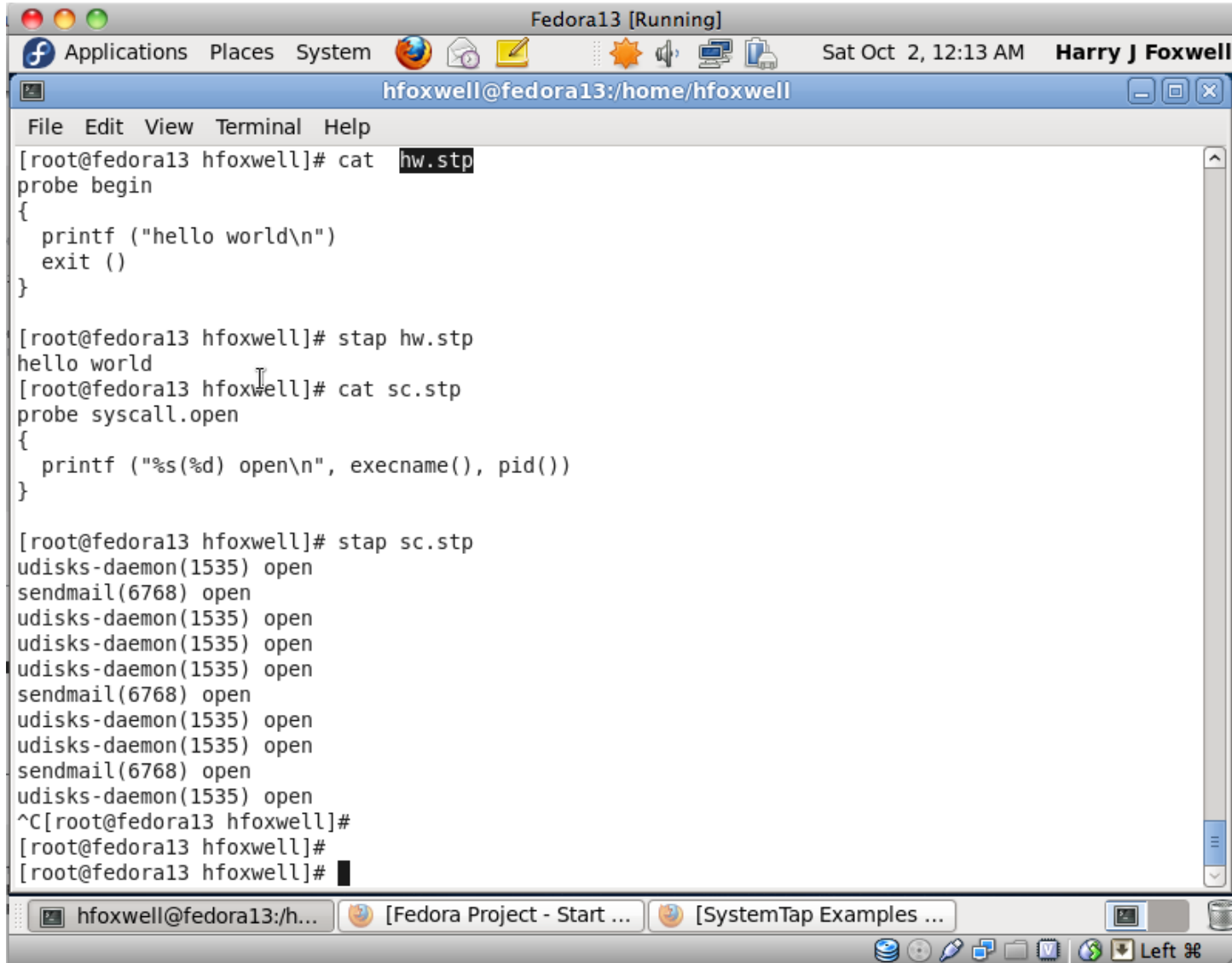


The screenshot shows a Fedora 13 desktop environment. The LatencyTOP 0.5 application is running, displaying a list of system targets and their maximum latency values. The 'firefox' target is selected, showing a maximum latency of 37.9 ms. The cause of the latency is 'Userspace lock contention'.

Targets	Max	Cause	Maximum	Percentage
<b>Global</b>		fsync() on a file (type 'F' for details)	37.9 ms	
jbd2/dm-0-8	88.9	Writing a page to disk	16.0 ms	
kdmflush	86.3	Scheduler: waiting for cpu	10.1 ms	
<b>firefox</b>	<b>37.9</b>	Userspace lock contention	4.6 ms	
events/0	25.6	Waiting for event (poll)	4.4 ms	
kblockd/0	25.0			
hald	10.5			
latencytop	9.7			
Xorg	8.6			
udisks-daemon	7.9			
metacity	5.9			
upowerd	5.4			
nautilus	4.6			
gnome-panel	4.4			

Buttons: Freeze, Refresh, Refresh in 14 s

# Some Examples: Linux systemtap

A screenshot of a Fedora 13 terminal window. The window title is "Fedora13 [Running]". The top bar shows "Applications Places System" and the date "Sat Oct 2, 12:13 AM" with the user "Harry J Foxwell". The terminal prompt is "hfoxwell@fedora13:/home/hfoxwell". The terminal shows the following commands and output:

```
[root@fedora13 hfoxwell]# cat hw.stp
probe begin
{
    printf ("hello world\n")
    exit ()
}

[root@fedora13 hfoxwell]# stap hw.stp
hello world
[root@fedora13 hfoxwell]# cat sc.stp
probe syscall.open
{
    printf ("%s(%d) open\n", execname(), pid())
}

[root@fedora13 hfoxwell]# stap sc.stp
udisks-daemon(1535) open
sendmail(6768) open
udisks-daemon(1535) open
udisks-daemon(1535) open
udisks-daemon(1535) open
sendmail(6768) open
udisks-daemon(1535) open
udisks-daemon(1535) open
sendmail(6768) open
udisks-daemon(1535) open
^C[root@fedora13 hfoxwell]#
[root@fedora13 hfoxwell]#
[root@fedora13 hfoxwell]#
```

The terminal window has a menu bar with "File Edit View Terminal Help". The bottom of the window shows a taskbar with icons for "hfoxwell@fedora13:/h...", "[Fedora Project - Start ...]", "[SystemTap Examples ...]", and a "Left" button.

# 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 step 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

`probfunc()` 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](mailto: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>