

Tools

	<i>DTrace</i>	<i>SystemTap</i>
Tool	dtrace(1M)	stap(1)
List probes	# dtrace -l # dtrace -l -P io	# stap -l 'ioblock.*' # stap -L 'ioblock.*'
One-liner	# dtrace -n ' syscall::read:entry { trace(arg1); } '	# stap -e ' probe syscall.read { println(fd); } '
Script	# dtrace -s script.d (optionally add -C for preprocessor, -q for quiet mode)	# stap script.stp
Custom probe	# dtrace -P io -n start	-
Integer arguments	# dtrace -n ' syscall::read:entry / cpu == \$1 / ' 0	# stap -e ' probe syscall.read { if(cpu() != \$1) next; println(fd); } ' 0
String arguments	# dtrace -n ' syscall::read:entry / execname == \$1 / ' "'cat"'	# stap -e ' probe syscall.read { if(execname() == @1) println(fd); } ' cat
Guru/destructive mode (!)	# dtrace -w ...	# stap -g ...
Redirect to file	# dtrace -o FILE ... (appends)	# stap -o FILE ... (rewrites)
Tracing process	# dtrace -n ' syscall::read:entry / pid == \$target / { ... }' -c 'cat /etc/motd' (or -p PID)	# stap -e ' probe syscall.read { if(pid() == target()) ... }' -c 'cat /etc/motd' (or -x PID)

Probe names

	<i>DTrace</i>	<i>SystemTap</i>
Begin/end	dtrace:::BEGIN, dtrace:::END	begin, end
foo() entry	fbt::foo:entry	kernel.function("foo") module("mod").function("foo")
foo() return	fbt::foo:return	kernel.function("foo").return
Wildcards	fbt::foo*:entry	kernel.function("foo*")
Static probe mark	sdt:::mark	kernel.trace("mark")
System call	syscall::read:entry	syscall.read
Timer once per second	tick-1s	timer.s(1)
Profiling	profile-997hz	timer.profile(), perf.*
read() from libc	pid\$target:libc:read:entry Traces process with pid == \$target	process("/lib64/libc.so.6").function("read") Traces any process that loads libc

In DTrace parts of probe name may be omitted: fbt::foo:entry -> foo:entry

Units for timer probes: ns, us, ms, s, hz, jiffies (SystemTap), m, h, d (all three - DTrace)

Printing

	<i>DTrace</i>	<i>SystemTap</i>
Value	<code>trace(v)</code>	<code>print(v)</code>
Value + newline	-	<code>println(v)</code>
Delimited values	-	<code>printf(", ", v1, v2)</code> <code>printdln(", ", v1, v2)</code>
Memory dump	<code>tracemem(ptr, 16)</code>	<code>printf("%16M", ptr)</code>
Formatted	<code>printf("%s", str)</code>	
Backtrace	<code>ustack(n)</code> <code>ustack()</code>	<code>print_ubacktrace()</code> <code>print_ustack(ubacktrace())</code>
Symbol	<code>usym(addr)</code> <code>ufunc(addr)</code> <code>uaddr(addr)</code>	<code>print(usymname(addr))</code> <code>print(usymdata(addr))</code>

If *u* prefix is specified, userspace symbols and backtraces are printed, if not -- kernel symbols are used

String operations

<i>Operation</i>	<i>DTrace</i>	SystemTap_
Get from kernel	<code>stringof(expr)</code> <code>(string) expr</code>	<code>kernel_string*()</code>
Convert scalar		<code>sprint()</code> and <code>sprintf()</code>
Copy from user	<code>copyinstr()</code>	<code>user_string*()</code>
Compare	<code>==, !=, >, >=, ,</code>	
Concat	<code>strjoin(str1, str2)</code>	<code>str1 . str2</code>
Get length	<code>strlen(str)</code>	
Check for substring	<code>strstr(haystack, needle)</code>	<code>isinstr(haystack, needle)</code>

Context variables

<i>Description</i>	<i>DTrace</i>	<i>SystemTap</i>
Thread	<code>curthread</code>	<code>task_current()</code>
Thread ID	<code>tid</code>	<code>tid()</code>
PID	<code>pid</code>	<code>pid()</code>
Parent PID	<code>ppid</code>	<code>ppid()</code>
User/group ID	<code>uid/gid</code>	<code>uid()/gid()</code> <code>eid()/egid()</code>
Executable name	<code>execname</code> <code>curpsinfo-></code> <code>ps_fname</code>	<code>execname()</code>
Command line	<code>curpsinfo-></code> <code>ps_psargs</code>	<code>cmdline_*</code>
CPU number	<code>cpu</code>	<code>cpu()</code>
Probe names	<code>probeprov</code> <code>probemod</code> <code>probefunc</code> <code>probename</code>	<code>pp()</code> <code>pn()</code> <code>ppfunc()</code> <code>probefunc()</code> <code>probemod()</code>

Time

<i>Time source</i>	<i>DTrace</i>	<i>SystemTap</i>
System timer	<code>`lbolt</code> <code>`lbolt64</code>	<code>jiffies()</code>
CPU cycles	-	<code>get_cycles()</code>
Monotonic time	<code>timestamp</code>	<code>local_clock_unit()</code> <code>cpu_clock_unit(cpu)</code>
CPU time of thread	<code>vtimestamp</code>	-
Real time	<code>walltimestamp</code>	<code>gettimeofday_unit()</code>

Where *unit* is one of s, ms, us, ns

Aggregations

<i>Time source</i>	<i>DTrace</i>	<i>SystemTap</i>
Add value	<code>@aggr[keys] = func(value);</code>	<code>aggr[keys]</code>
Printing	<code>printa(@aggr);</code> <code>printa("format string", @aggr);</code>	<code>foreach([keys] in aggr) {</code> <code> print(keys, @func(aggr[keys]));</code> <code>}</code>
Clear	<code>clear(@aggr);</code> or <code>trunc(@aggr);</code>	<code>delete aggr;</code>
Normalization by 1000	<code>normalize(@aggr, 1000);</code> <code>denormalize(@aggr);</code>	<code>@func(aggr) / 1000</code> in printing
Select 20 values	<code>trunc(@aggr, 20);</code>	<code>foreach([keys] in aggr limit 20) {</code> <code> print(keys, @func(aggr[keys]));</code> <code>}</code>
Histograms (linear in [10;100] with step 5 and logarithmical)	<code>@lin = lquantize(value, 10, 100, 5);</code> <code>@log = quantize(value);</code> <code>...</code> <code>printa(@lin); printa(@log);</code>	<code>aggr</code>

Where *func* is one of count, sum, min, max, avg, stddev

Process management

SystemTap

Getting task_struct pointers:

- task_current() – current task_struct
- task_parent(t) – parent of task t
- pid2task(pid) – task_struct by pid

Working with task_struct pointers:

- task_pid(t) task_tid(t)
- task_state(t) – 0 (running), 1-2 (blocked)
- task_execname(t)

DTrace

kthread_t* curthread fields:

- t_tid, t_pri, t_start, t_pctcpu

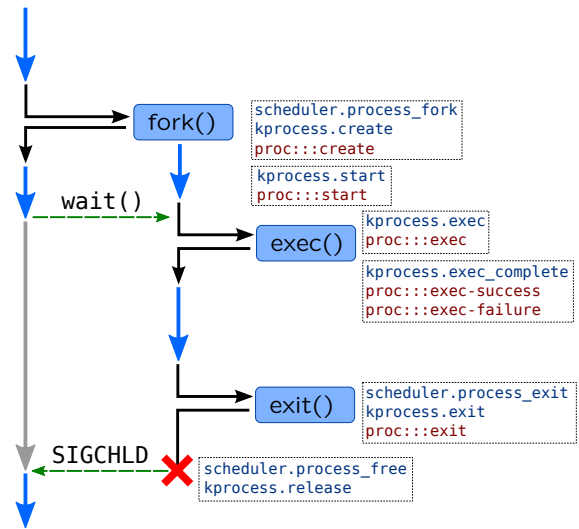
psinfo_t* curpsinfo fields:

- pr_pid, pr_uid, pr_gid, pr_fname, pr_psargs, pr_start

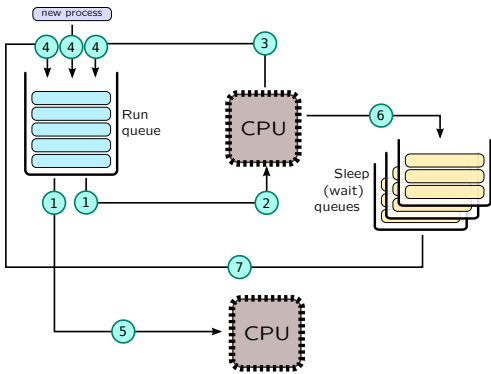
lwpsinfo_t* curlwpsinfo fields:

- pr_lwpid, pr_state/pr_sname

psinfo_t* and lwpsinfo_t* are passed to some proc::: probes



Scheduler



	<i>DTrace</i>	<i>SystemTap</i>
1	sched:::dequeue	kernel.function("dequeue_task")
2	sched:::on-cpu	scheduler.cpu_on
3	sched:::off-cpu	scheduler.cpu_off
4	sched:::enqueue	kernel.function("enqueue_task")
5	-	scheduler.migrate
6	sched:::sleep	-
7	sched:::wakeup	scheduler.wakeup

Virtual memory

Probes

SystemTap

- vm.brk – allocating heap
- vm.mmap – allocating anon memory
- vm.munmap – freeing anon memory

DTrace

- as_map:entry – allocating proc mem
- as_unmap:entry – freeing proc mem

Page faults

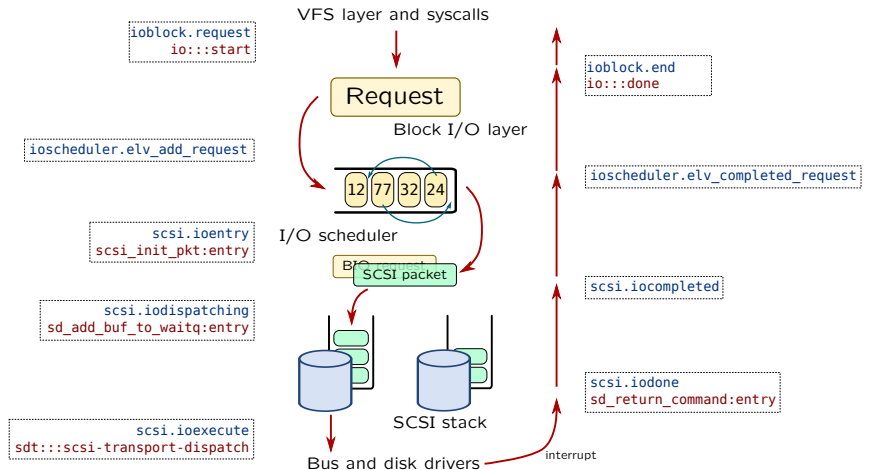
Type	<i>DTrace</i>	<i>SystemTap</i>
Any	vminfo:::as_fault	vm.pagefault vm.pagefault.return perf.sw.page_faults
Minor		perf.sw.page_faults_min
Major	vminfo:::maj_fault	perf.sw.page_faults_maj
CoW	vminfo:::cow_fault	
Protection	vminfo:::prot_fault	

Block Input-Output

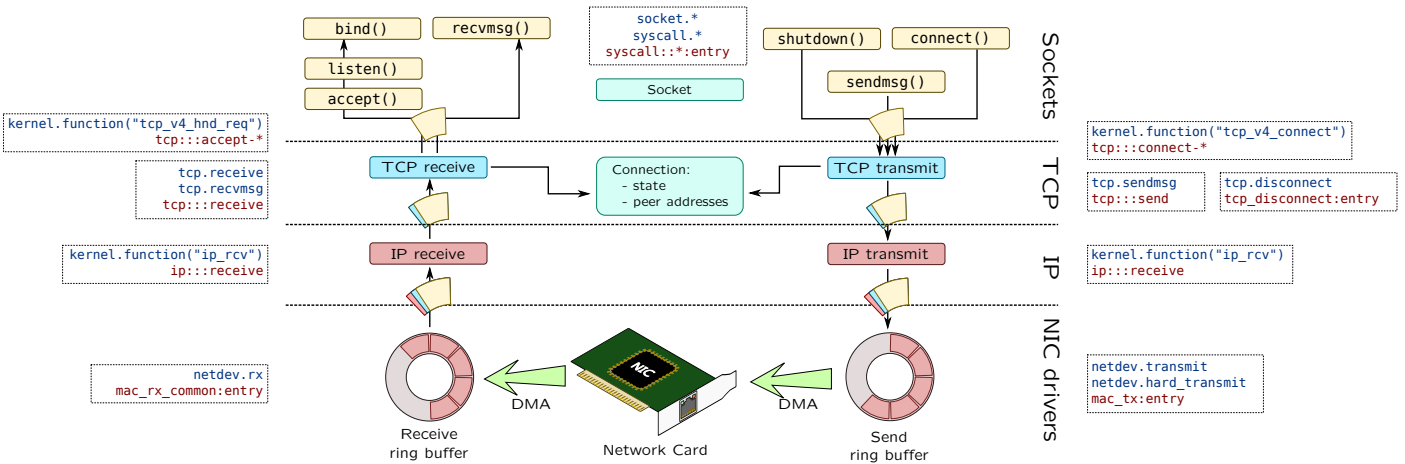
Block request structure fields:

Field	bufinfo_t struct buf	struct bio
Flags	b_flags	bi_flags
R/W	b_flags	bi_rw
Size	b_bcount	bi_size
Block	b_blkno b_lblkno	bi_sector
Callback	b_iodone	bi_end_io
Device	b_edev b_dip	bi_bdev

* flags B_WRITE, B_READ



Network stack



Non-native languages

Function call	DTrace	SystemTap
Java*	method-entry <ul style="list-style-type: none"> arg0 — internal JVM thread's identifier arg1:arg2 — class name arg3:arg4 — method name arg5:arg6 — method signature 	hotspot.method_entry <ul style="list-style-type: none"> thread_id — internal JVM thread's identifier class — class name method — method name sig — method signature
Perl	perl\$target:::sub-entry <ul style="list-style-type: none"> arg0 — subroutine name arg1 — source file name arg2 — line number 	process("...").mark("sub_entry") <ul style="list-style-type: none"> \$arg1 — subroutine name \$arg2 — source file name \$arg3 — line number
Python	python\$target:::function-entry <ul style="list-style-type: none"> arg0 — source file name arg1 — function name 	python.function.entry <ul style="list-style-type: none"> \$arg1 — source file name \$arg2 — function name
PHP	function-entry <ul style="list-style-type: none"> arg0 — function name arg1 — file name arg2 — line number arg3 — class name arg4 — scope operator :: 	process("...").mark("function_entry") <ul style="list-style-type: none"> \$arg1 — function name \$arg2 — file name \$arg3 — line number \$arg4 — class name \$arg5 — scope operator ::

*requires -XX:+DTraceMethodProbes