The following library of DTrace one-liners were last tested on FreeBSD 10.0.

Listing Probes

```
# List probes and search for string "foo":
dtrace -l | grep foo

# Summarize probes by providers:
dtrace -l | awk '{ print $2 }' | sort | uniq -c | sort -n
```

Syscalls

```
# Trace file opens with process and filename:
dtrace -n 'syscall::open*:entry { printf("%s %s", execname, copyinstr(arg0)); }'

# Count system calls by program name:
dtrace -n 'syscall:::entry { @[execname] = count(); }'

# Count system calls by syscall:
dtrace -n 'syscall:::entry { @[probefunc] = count(); }'

# Count system calls by syscall, for PID 123 only:
dtrace -n 'syscall:::entry /pid == 123/ { @[probefunc] = count(); }'

# Count system calls by syscall, for all processes with a specific program name ("nginx"):
dtrace -n 'syscall:::entry /execname == "nginx"/ { @[probefunc] = count(); }'

# Count system calls by PID and program name:
dtrace -n 'syscall:::entry { @[pid, execname] = count(); }'

# Summarize requested read() sizes by program name, as power-of-2 distributions (bytes):
dtrace -n 'syscall::read:entry { @[execname] = quantize(arg2); }'

# Summarize returned read() sizes by program name, as power-of-2 distributions (bytes or error):
dtrace -n 'syscall::read:return { @[execname] = quantize(arg1); }'

# Summarize read() latency as a power-of-2 distribution by program name (ns):
dtrace -n 'syscall::read:entry { self->ts = timestamp; } syscall::read:return /self->ts/ {
    @[execname, "ns"] = quantize(timestamp - self->ts); self->ts = 0; }'

# Summarize read() latency as a linear distribution (0 to 1000, step 5) by program name (ms):
dtrace -n 'syscall::read:entry { self->ts = timestamp; } syscall::read:return /self->ts/ {
    @[execname, "ms"] = lquantize((timestamp - self->ts) / 1000000, 0, 1000, 5); self->ts = 0; }'

# Summarize read() on-CPU duration as a power-of-2 distribution by program name (ns):
dtrace -n 'syscall::read:entry { self->ts = vtimestamp; } syscall::read:return /self->ts/ {
    @[execname, "ns"] = quantize(vtimestamp - self->ts); self->ts = 0; }'

# Count read() variants that "nginx" is using (if previous one-liners didn't work):
dtrace -n 'syscall::*read*:entry /execname == "nginx"/ { @[probefunc] = count(); }'

# Summarize returned pread() sizes for "nginx" as distributions (bytes or error):
dtrace -n 'syscall::pread:return /execname == "nginx"/ { @ = quantize(arg1); }'

# Count socket accept() variants by process name:
dtrace -n 'syscall::*accept*:return { @[execname] = count(); }'

# Count socket connect() variants by process name:
dtrace -n 'syscall::*connect*:return { @[execname] = count(); }'

# Summarize returned pread() sizes for "nginx"... and label the output:
dtrace -n 'syscall::pread:return /execname == "nginx"/ { @["rval (bytes)"] = quantize(arg1); }'
```

Process Tracing

```
# Trace new processes showing program name (and args if available):
dtrace -n 'proc:::exec-success { trace(curpsinfo->pr_psargs); }'

# Count process-level events:
dtrace -n 'proc::: { @[probename] = count(); }'
```

Profiling

```
# Count sampled thread names on-CPU at 997 Hertz:
dtrace -n 'profile-997 { @[stringof(curthread->td_name)] = count(); }'

# Count sampled non-idle thread names on-CPU at 997 Hertz:
dtrace -n 'profile-997 /!(curthread->td_flags & 0x20)/ { @[stringof(curthread->td_name)] = count(); }'

# Count sampled on-CPU kernel stacks at 99 Hertz:
dtrace -n 'profile-99 /arg0/ { @[stack()] = count(); }'

# Count sampled process names and on-CP user stacks at 99 Hertz:
dtrace -n 'profile-99 /arg1/ { @[execname, ustack()] = count(); }'
```

Storage I/O

```
# Count kernel stacks leading to block device I/O:
dtrace -n 'io:::start { @[stack()] = count(); }'
```

Scheduler

```
# Count kernel stacks leading to a context-switch off-CPU:
dtrace -n 'sched:::off-cpu { @[stack()] = count(); }'
```

IP

```
# Count IP-level events:
dtrace -n 'ip::: { @[probename] = count(); }'
```

UDP

```
# Count UDP-level events:
dtrace -n 'udp::: { @[probename] = count(); }'
```

TCP

```
# Count TCP-level events:
dtrace -n 'tcp::: { @[probename] = count(); }'

# Trace TCP accepted connections by remote IP address:
dtrace -n 'tcp:::accept-established { trace(args[3]->tcps_raddr); }'

# Count TCP passive opens by remote IP address:
dtrace -n 'tcp:::accept-established { @[args[3]->tcps_raddr] = count(); }'

# Count TCP active opens by remote IP address:
dtrace -n 'tcp:::connect-established { @[args[3]->tcps_raddr] = count(); }'

# Count TCP sent messages by remote IP address:
dtrace -n 'tcp:::send { @[args[2]->ip_daddr] = count(); }'

# Count TCP received messages by remote IP address:
dtrace -n 'tcp:::receive { @[args[2]->ip_saddr] = count(); }'

# Summarize TCP sent messages by IP payload size, as a power-of-2 distribution:
dtrace -n 'tcp:::send { @[args[2]->ip_daddr] = quantize(args[2]->ip_plength); }'
```

Kernel Locks

```
# Sum kernel adaptive lock block time by process name (ns):
dtrace -n 'lockstat:::adaptive-block { @[execname] = sum(arg1); }'

# Summarize adaptive lock block time distribution by process name (ns):
dtrace -n 'lockstat:::adaptive-block { @[execname] = quantize(arg1); }'

# Sum kernel adaptive lock block time by kernel stack trace (ns):
dtrace -n 'lockstat:::adaptive-block { @[stack()] = sum(arg1); }'

# Sum kernel adaptive lock block time by lock name (ns):
dtrace -n 'lockstat:::adaptive-block { @[arg0] = sum(arg1); } END { printa("%40a %@16d ns\n", @); }'

# Sum kernel adaptive lock block time by calling function (ns):
dtrace -n 'lockstat:::adaptive-block { @[caller] = sum(arg1); } END { printa("%40a %@16d ns\n", @);
```

```
}'
```

Raw Kernel Tracing

```
# Count kernel slab memory allocation by function:
dtrace -n 'fbt::kmem*:entry { @[probefunc] = count(); }'

# Count kernel slab memory allocation by calling function:
dtrace -n 'fbt::kmem*:entry { @[caller] = count(); } END { printa("%40a %@16d\n", @); }'

# Count kernel malloc() by calling function:
dtrace -n 'fbt::malloc:entry { @[caller] = count(); } END { printa("%40a %@16d\n", @); }'

# Count kernel malloc() by kernel stack trace:
dtrace -n 'fbt::malloc:entry { @[stack()] = count(); }'

# Summarize vmem_alloc()s by arena name and size distribution:
dtrace -n 'fbt::vmem_alloc:entry { @[args[0]->vm_name] = quantize(arg1); }'

# Summarize TCP life span in seconds:
dtrace -n 'fbt::tcp_close:entry { @["TCP life span (seconds):"] =
    quantize((uint32_t)(`ticks - args[0]->t_starttime) / `hz); }'
```

CategoryDTrace