DTrace Internals: Digging into DTrace

Arun Thomas
<arun.thomas@acm.org>
@arunthomas

BSDCan 2017

On Systems Programming

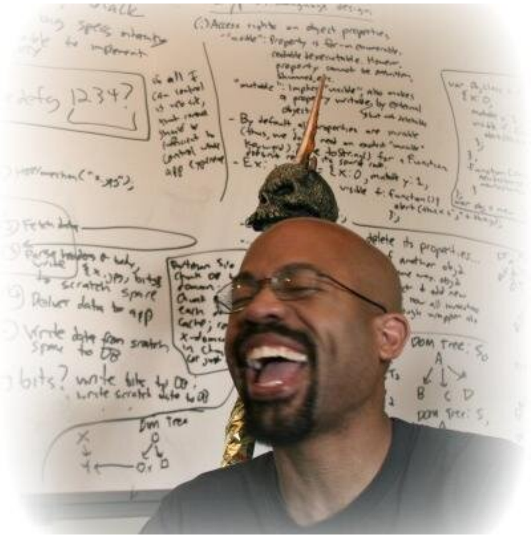
James Mickens, The Night Watch:

> *[A] systems programmer has seen the terrors of the world and understood the intrinsic horror of existence*

On Kernel Debugging

James Mickens, The Night Watch:

> *When you debug ... an OS kernel, you do it Texas-style. You gather some mean, stoic people, people who have seen things die, and you get some* ***primitive tools****, like a compass and a rucksack and a stick that's pointed on one end, and you walk into the wilderness and you look for trouble, possibly while using chewing tobacco.*



DTrace: An Advanced Debugging Tool

It's like packing ***military-grade GPS and a lightsaber*** *in your rucksack before heading into the wilderness*

And looking for ***trouble***

Talk Overview

- DTrace 101
- DTrace Architecture
- DTrace on FreeBSD (With Code)

What is DTrace?

- **Safe Dynamic Trace-ing of production systems**
- **D language drives instrumentation and reporting**
  - Inspired by C and AWK
- **Unified tracing of full software stack**
  - Applications, Runtimes, Libraries, Kernel
- **Single framework supports variety of instrumentation sources**
  - FBT, SDT, syscall, pid providers
- **No overhead when not enabled**
  - When **enabled**, overhead depends on provider/probes
- **Available on FreeBSD, NetBSD, macOS, and OpenBSD (in progress)**

What Can DTrace Trace?

- Function calls (Userspace and Kernel)
- Function arguments and return values
- Stack traces (Userspace and Kernel)
- Programmer-defined tracepoints
- Data structures (Userspace and Kernel)
- Much, much more

Syscall Tracing (syscall Provider)

```
# dtrace -qn 'syscall:::entry { printf("%s %s\n", execname, probefunc); }'

sshd read
sshd getpid
sshd select
sshd sigprocmask
sshd sigprocmask
sshd write
sshd select
ntpd sigprocmask
ntpd sigreturn
ntpd select
```

read() Sizes (syscall Provider)

```
# dtrace -qn 'syscall::read:return /execname == "sshd"/ { @ = quantize(arg0); }'

           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@                                    23
               1 |@@@@@@@@@@                               41
               2 |                                         1
               4 |@@@@                                     19
               8 |@                                        4
              16 |                                         2
              32 |@                                        6
              64 |@@                                       9
             128 |@@@                                      12
             256 |@@@@@                                    20
             512 |@@                                       9
            1024 |@@@                                      12
            2048 |@                                        4
            4096 |@                                        4
            8192 |@                                        3
           16384 |                                         0
```

Kernel Function Tracing (FBT Provider)

```
# dtrace -n 'fbt::malloc:entry /pid != $pid/ { printf("%s %s\n", execname, arg0); }'
dtrace: description 'fbt::malloc:entry ' matched 1 probe
CPU     ID                    FUNCTION:NAME
  1  32051                     malloc:entry   sendmail 44
  0  32051                     malloc:entry   syncer 496
  2  32051                     malloc:entry   ntpd   16
```

Userland Tracing (pid Provider)

```
# dtrace -qn 'pid$target:libc.so.7::entry {  @[probefunc] = count(); }' -c /bin/ls
  ...
  memset                                                           69
  mbrtowc                                                          76
  wcrtomb                                                          76
  __free                                                          104
  free                                                            104
  memcpy                                                          119
  strncmp                                                         252
```

proc Provider (SDT)

What happens when we "man man"?

```
# dtrace -n 'proc:::exec-success { printf("%s", curpsinfo->pr_psargs); }'
dtrace: description 'proc:::exec-success ' matched 1 probe
CPU     ID                    FUNCTION:NAME
  1     14                none:exec-success /bin/sh /usr/bin/man man
  1     14                none:exec-success /sbin/sysctl -n hw.machine_arch
  2     14                none:exec-success /sbin/sysctl -n hw.machine
  2     14                none:exec-success /usr/bin/locale
  2     14                none:exec-success /usr/bin/zcat /usr/share/man/man1/man.1.gz
  0     14                none:exec-success head -1
  2     14                none:exec-success /usr/bin/zcat /usr/share/man/man1/man.1.gz
  1     14                none:exec-success mandoc -Tlint -Wunsupp
  2     14                none:exec-success /usr/bin/zcat /usr/share/man/man1/man.1.gz
  1     14                none:exec-success mandoc
  0     14                none:exec-success less
```

DTrace Solaris History

- **Created by Bryan Cantrill, Mike Shapiro, and Adam Leventhal at Sun Microsystems**
- **Development begins (2001)**
- **Solaris integration (2003)**

```
From bmc Wed Sep  3 10:27:51 2003
Subject: Houston, Tranquility Base here...
To: dtrace-interest@kiowa

DTrace has landed.  Twenty-three months after we set out, the first cut
of DTrace has integrated into Solaris 10.
```

- **USENIX ATC paper (2004)**
- **Solaris 10 release (2005)**

DTrace Solaris History (2)

DTrace merge stats from Bryan's announcement:

```
  Number of source files in usr/src/uts                                  1,757
  Rank, in lines of code, of dtrace.c among these                           12
  Rank, in number of assertions, of dtrace.c among these                     7
  Lines in new .c and .h files created by DTrace project                54,644
  Lines of code (lines with trailing semicolon) in same                 17,088
  Lines of comments in same                                              6,713
```

```
  Age, in months, of DTrace project                                         23
  Number of engineers on DTrace project                                      3
  Number of above neither married nor engaged at beginning of project        3
  Number of above either married or engaged at end of project                2
```

```
  Subscribers to dtrace-interest@kiowa                                     181
  Number of people who asked to unsubscribe from dtrace-interest@kiowa       1
```

DTrace BSD history

- Mac OS X 10.5 release with DTrace (2007)
- Merged into FreeBSD (2008)
- DTrace enabled by default in FreeBSD (2010)
- Merged into NetBSD (2010)
- DTrace enabled by default in NetBSD (2016)
- DTrace on OpenBSD effort begins (2016)

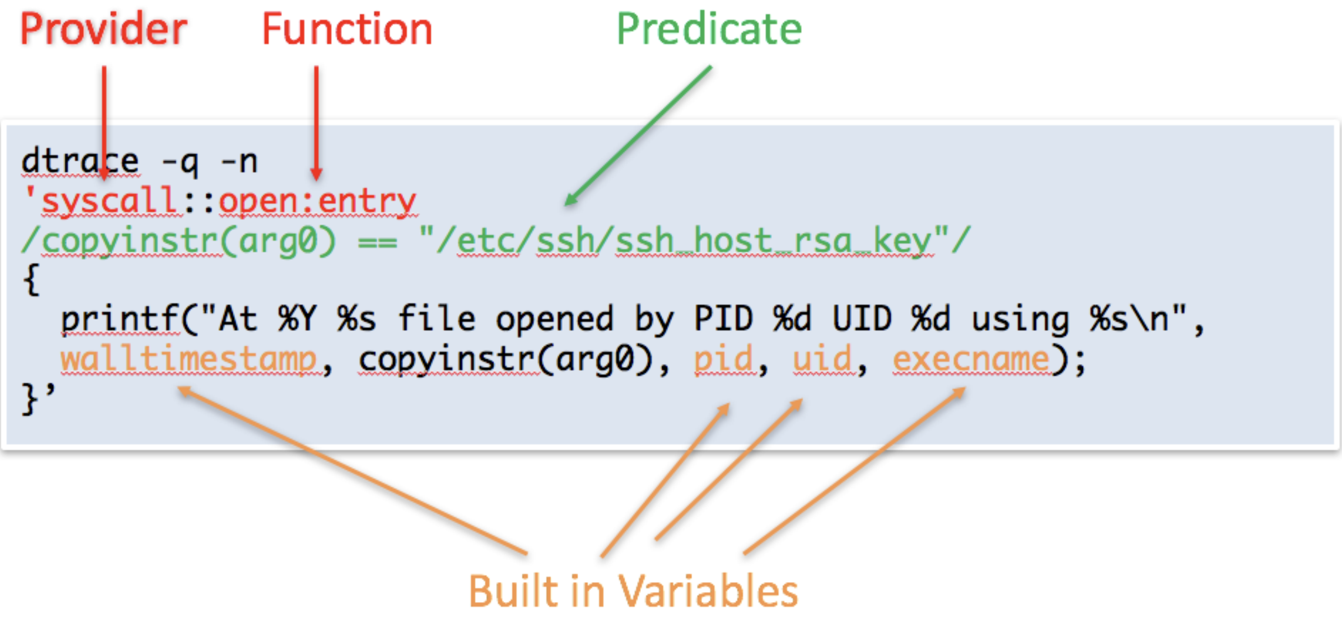
D Language

- **A powerful, safe subset of C with elements from AWK**
- **And some new things:**
  - Aggregations
  - Predicates
  - Associative Arrays

DTrace Terminology

- **Probe: What to trace**
  - **Provider**: DTrace-defined module that defines a kind of instrumentation
  - **Module**: Software module (e.g., kernel, driver, library)
  - **Function**: A function in a module (e.g., `write()`)
  - **Name**: The specific probe (e.g., entry)
- **Action: D language statement carried out when a probe fires**
- **Predicate: Filters which probe will fire at run time**

DTrace syntax



```
dtrace -q -n
'syscall::open:entry
/copyinstr(arg0) == "/etc/ssh/ssh_host_rsa_key"/
{
  printf("At %Y %s file opened by PID %d UID %d using %s\n",
  walltimestamp, copyinstr(arg0), pid, uid, execname);
}'
```

DTrace Providers

- **fbt: Function Boundary Tracing**
- **syscall: System Calls**
- **pid: User space processes**
- **proc: Process Operations**
- **sched: Scheduler**
- **Network Protocols**
  - IP, UDP, TCP
- **lock: Kernel locking points**
- **io: I/O calls**
- **vfs: Filesystem Routines**
- **profile: Timing source**
- **mac_framework: Mandatory Access Control framework**
- **dtaudit: Audit framework**

DTrace Components

- **Userland**
    - Consumers (e.g., `dtrace(1)`)
    - `libdtrace` - API for consumers, talks to kernel DTrace via `ioctl`
- **Kernel**
    - Generic instrumentation framework
    - Providers for specific instrumentation (e.g., FBT, pid, SDT)
    - Kernel hooks
- **Toolchain**
    - D language compiler
    - Build support for CTF generation

DTrace Core Framework

- **Interfaces to userspace through `ioctl()` to `/dev/dtrace/*`**
- **Majority of the code is generic framework code**
    - Does not do any instrumentation
    - Delegates instrumentation to providers
- **Implements a virtual machine for the `D Intermediate Format (DIF)`**
    - Ensures safety of D scripts
    - Analogous to Java bytecode

D Intermediate Format (DIF)

- **D compiler:**
  - Compiles D scripts to DIF
  - Assembles DIF into DIFOs (for predicate and actions)
  - Merges DIFOs into DOF (D Object Format)
- **DIF has a RISC-like instruction set that supports D language constructs**
  - Accessing D variables (`DT_VAR`) such as `execname`

```
# dtrace -S -n 'syscall:freebsd:open:entry {printf("%s %s\n", execname, copyinstr(arg0)); }'

DIFO 0x0x8049c7150 returns string (unknown) by ref (size 256)
OFF OPCODE      INSTRUCTION
00: 29011801    ldgs DT_VAR(280), %r1       ! DT_VAR(280) = "execname"
01: 23000001    ret  %r1
...
```

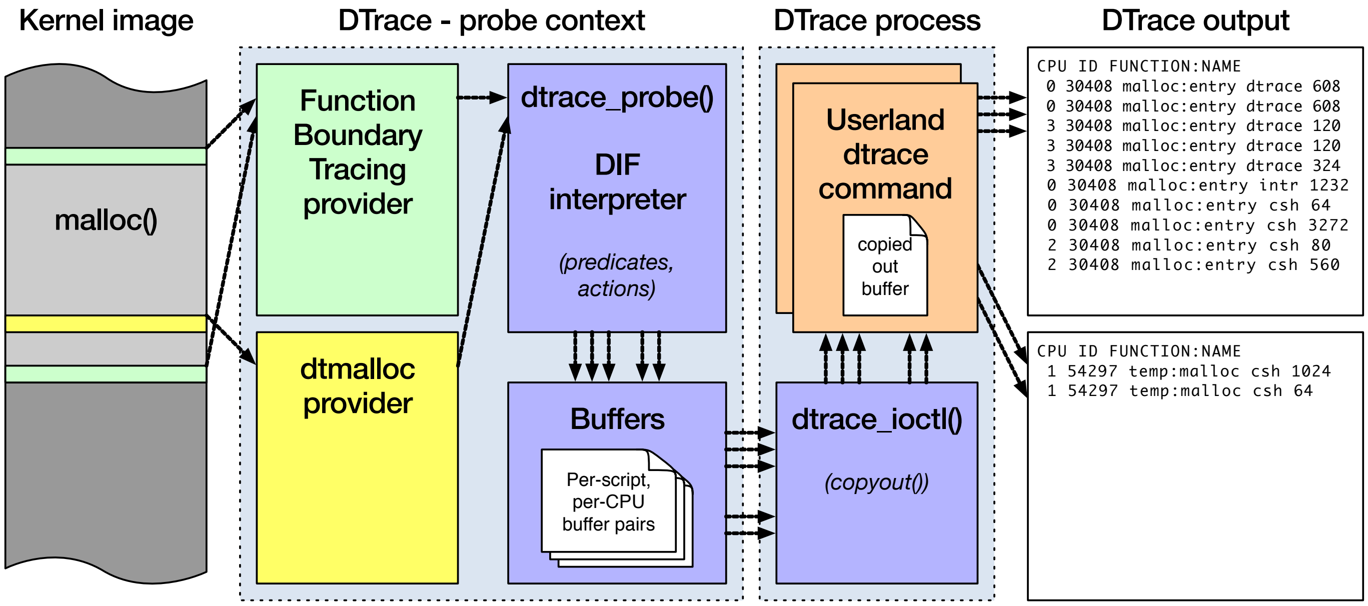
- **Kernel DTrace framework will interpret the DIF to perform actions defined in D script**

The Role of Providers

- Holds the knowledge about specific instrumentation techniques
- Enables probes when consumer says so
- Process is specific to each provider
- Goal: Transfer control to DTrace framework by calling `dtrace_probe(probe_id)` somehow (e.g., runtime patching the executable)
- Then, DTrace framework can performs actions associated with probe

DTrace Architecture

```
dtrace -n 'fbt::malloc:entry { trace(execname); trace(arg0); }'
```



```
CPU ID FUNCTION:NAME
 0 30408 malloc:entry dtrace 608
 0 30408 malloc:entry dtrace 608
 3 30408 malloc:entry dtrace 120
 3 30408 malloc:entry dtrace 120
 3 30408 malloc:entry dtrace 324
 0 30408 malloc:entry intr 1232
 0 30408 malloc:entry csh 64
 0 30408 malloc:entry csh 3272
 2 30408 malloc:entry csh 80
 2 30408 malloc:entry csh 560
```

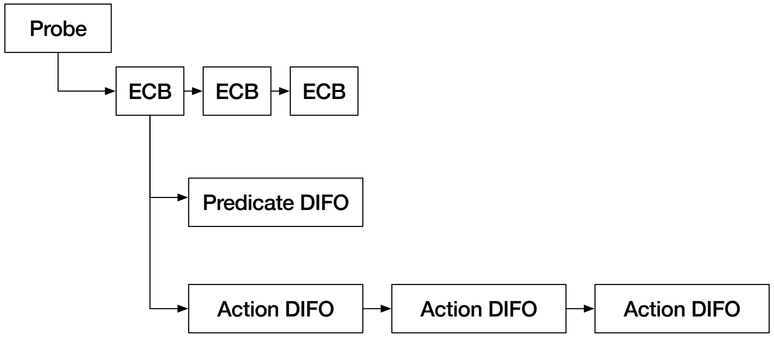
```
CPU ID FUNCTION:NAME
 1 54297 temp:malloc csh 1024
 1 54297 temp:malloc csh 64
```

```
dtrace -n 'dtmalloc::temp:malloc /execname=“csh”/ { trace(execname); trace(arg3); }'
```

Diagram shamelessly stolen from TeachBSD

Enabling Control Blocks (ECBs)

- **Kernel data structure that represents an enabled probe**
- **Each ECB contains DIFO for predicates and actions associated with probe enabling**
- **ECBs can be chained together**
    - If multiple consumers are interested in a given probe

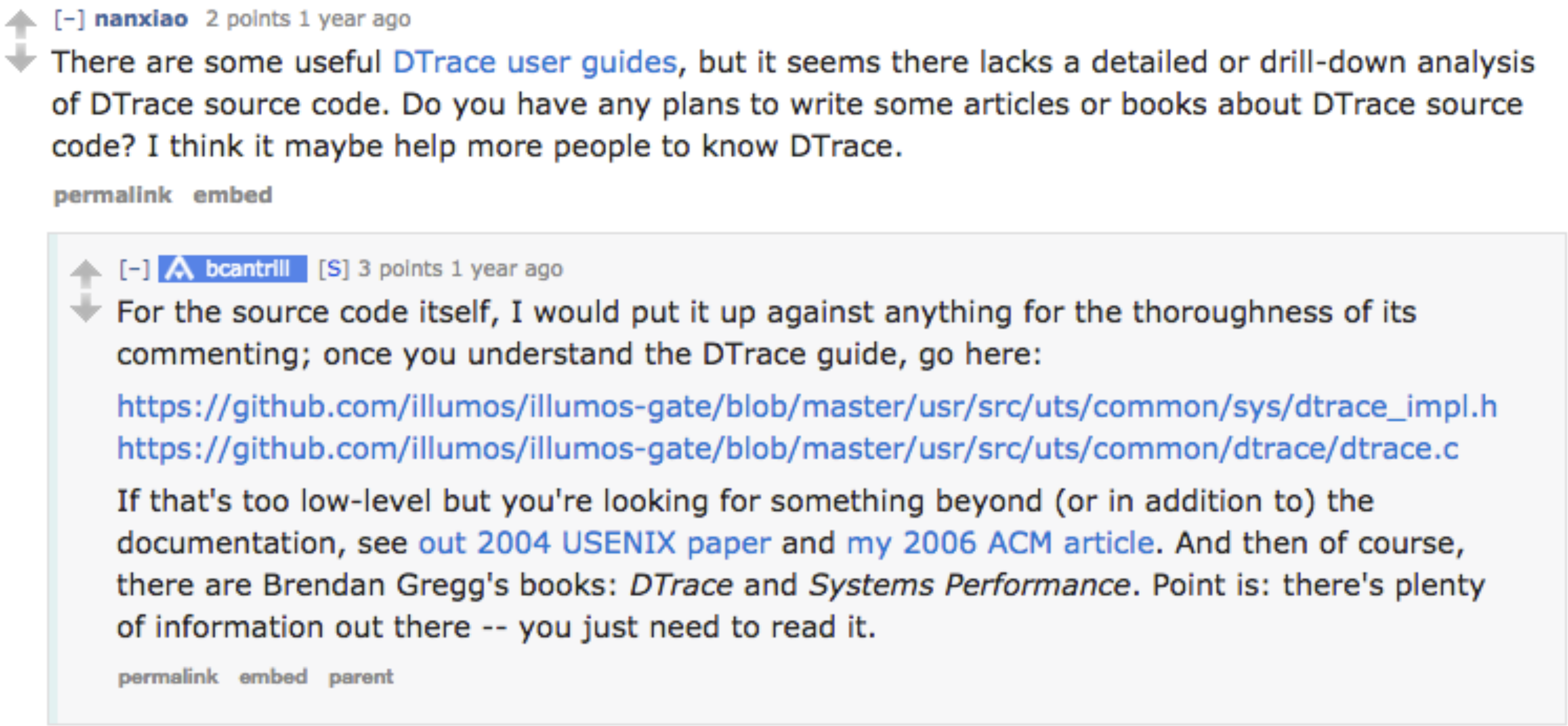


ECBs in Action

- **When probe is first enabled, provider will rewrite code to enter DTrace framework in "probe context"**
- **When enabled probe fires, ECBs iterated over and DIFO is interpreted.**
  - As DIFO is interpreted, data is placed in buffers
  - Consumers will periodically read the buffers
- **ECBs are removed when probe consumer terminates**
  - If no ECBs left for the probe, rewrite code to restore original instruction

Digging into DTrace Source Code

Bryan Cantrill's Reddit AMA illuminates our path:



[–] nanxiao 2 points 1 year ago

There are some useful DTrace user guides, but it seems there lacks a detailed or drill-down analysis of DTrace source code. Do you have any plans to write some articles or books about DTrace source code? I think it maybe help more people to know DTrace.

permalink embed

[–] bcantrill [S] 3 points 1 year ago

For the source code itself, I would put it up against anything for the thoroughness of its commenting; once you understand the DTrace guide, go here:

https://github.com/illumos/illumos-gate/blob/master/usr/src/uts/common/sys/dtrace_impl.h
https://github.com/illumos/illumos-gate/blob/master/usr/src/uts/common/dtrace/dtrace.c

If that's too low-level but you're looking for something beyond (or in addition to) the documentation, see out 2004 USENIX paper and my 2006 ACM article. And then of course, there are Brendan Gregg's books: *DTrace* and *Systems Performance*. Point is: there's plenty of information out there -- you just need to read it.

permalink embed parent

DTrace Frontends (aka Consumers)

dtrace(1)

```
NAME
       dtrace - generic front-end to the DTrace facility
```

lockstat(1)

```
NAME
       lockstat - report kernel lock and profiling statistics
```

plockstat(1)

```
NAME
     plockstat - Trace pthread lock statistics using DTrace
```

Writing a Consumer

- **A DTrace consumer is a userland program that drives instrumentation**
  - Instructs providers to create/enable one or more probes
  - Ingests trace data from buffers
- **Using libdtrace APIs, you can write your own DTrace client (aka consumer)**
  - In the language of your choice (e.g., Python, Go, Rust)
- **To see what a consumer is requesting of the kernel:**

```
sysctl debug.dtrace.verbose_ioctl=1
```

```
dtrace_ioctl(411): DTRACEIOC_ENABLE
dtrace_ioctl(584): DTRACEIOC_GO
dtrace_ioctl(388): DTRACEIOC_DOFGET
dtrace_ioctl(388): DTRACEIOC_DOFGET
dtrace_ioctl(778): DTRACEIOC_STATUS
dtrace_ioctl(237): DTRACEIOC_BUFSNAP curcpu 2 cpu 0
dtrace_ioctl(338): copyout the buffer snapshot
dtrace_ioctl(357): copyout buffer desc: size 0 drops 0 errors 0
```

`dtrace(1)`

- **The canonical DTrace consumer**
- **Source code for the `dtrace` command is here:**
  - *cddl/contrib/opensolaris/cmd/dtrace/dtrace.c*
- **We will discuss `plockstat(1)` as it's simpler**

plockstat(1)

cddl/contrib/opensolaris/cmd/plockstat/plockstat.c

```
    if ((g_dtp = dtrace_open(DTRACE_VERSION, 0, &err)) == NULL)
        fatal("failed to initialize dtrace: %s\n",
            dtrace_errmsg(NULL, err));
...
    if ((prog = dtrace_program_strcompile(g_dtp, g_prog,
        DTRACE_PROBESPEC_NAME, 0, 0, NULL)) == NULL)
        dfatal("failed to compile program");

    if (dtrace_program_exec(g_dtp, prog, &info) == -1)
        dfatal("failed to enable probes");
...
    if (dtrace_go(g_dtp) != 0)
        dfatal("dtrace_go()");
...
    do {
        ...
        switch (dtrace_work(g_dtp, stdout, NULL, chewrec, NULL)) {
        case DTRACE_WORKSTATUS_DONE:
            done = 1;
            break;
        ...
    } while (!done);
```

libdtrace Source

- **Key paths:**
  - *cddl/contrib/opensolaris/lib/libdtrace*
  - *cddl/contrib/opensolaris/lib/libdtrace/common*
  - *cddl/contrib/opensolaris/lib/libdtrace/{aarch64, i386, riscv}*
- **Key headers:**
  - *cddl/contrib/opensolaris/lib/libdtrace/common/dtrace.h*
  - *cddl/contrib/opensolaris/lib/libdtrace/common/dt_impl.h*
- **Key source files:**
  - *cddl/contrib/opensolaris/lib/libdtrace/common/dt_*.c*

dt_open.c

cddl/contrib/opensolaris/lib/libdtrace/common/dt_open.c:

Which D language constructs are supported?

```
static const dt_ident_t _dtrace_globals[] = {
...
{ "execname", DT_IDENT_SCALAR, 0, DIF_VAR_EXECNAME,
    DT_ATTR_STABCMN, DT_VERS_1_0, &dt_idops_type, "string" },
{ "exit", DT_IDENT_ACTFUNC, 0, DT_ACT_EXIT, DT_ATTR_STABCMN, DT_VERS_1_0,
    &dt_idops_func, "void(int)" },
...
{ "strlen", DT_IDENT_FUNC, 0, DIF_SUBR_STRLEN, DT_ATTR_STABCMN, DT_VERS_1_0,
    &dt_idops_func, "size_t(const char *)" },
...
};
```

DIF compilation

- **See *cddl/contrib/opensolaris/lib/libdtrace***
- **Start with *dt_cc.c*, the compiler driver**
- **Files of interest:**
  - *dt_lex.l* - lex scanner
  - *dt_grammar.y* - yacc grammar
  - *dt_parser.c* - parse tree creation and semantic checking
  - *dt_cc.c* - compiler driver and `dtrace_prog_t` construction
  - *dt_cg.c* - DIF code generator
  - *dt_as.c* - DIF assembler
  - *dt_dof.c* - `dtrace_prog_t` -> DOF conversion

Kernel Source Paths

Framework paths:

- ***sys/cddl/contrib/opensolaris/uts/common/dtrace***
- ***sys/cddl/contrib/opensolaris/uts/common/sys***
- ***sys/cddl/dev/dtrace/***

Provider paths:

- ***sys/cddl/dev/dtmalloc***
- ***sys/cddl/dev/fbt***
- ***sys/cddl/dev/profile***
- ***sys/cddl/dev/sdt***
- ***sys/cddl/dev/systrace***

Key Kernel Source Files

- ***sys/cddl/contrib/opensolaris/uts/common/sys/dtrace.h***
- ***sys/cddl/contrib/opensolaris/uts/common/sys/dtrace_impl.h***
- ***sys/cddl/contrib/opensolaris/uts/common/dtrace/dtrace.c***

From *sys/cddl/contrib/opensolaris/uts/common/dtrace/dtrace.c*

```
/*
 * DTrace - Dynamic Tracing for Solaris
 *
 * This is the implementation of the Solaris Dynamic Tracing framework
 * (DTrace).  The user-visible interface to DTrace is described at length in
 * the "Solaris Dynamic Tracing Guide".  The interfaces between the libdtrace
 * library, the in-kernel DTrace framework, and the DTrace providers are
 * described in the block comments in the <sys/dtrace.h> header file.  The
 * internal architecture of DTrace is described in the block comments in the
 * <sys/dtrace_impl.h> header file.  The comments contained within the DTrace
 * implementation very much assume mastery of all of these sources; if one has
 * an unanswered question about the implementation, one should consult them
 * first.
```

dtrace_probe()

- "The epicenter of DTrace"
- Able to probe virtually any context
- Implements actions by processing enabling controlled blocks (ECBs) and linked DIF Code
- In *sys/cddl/contrib/opensolaris/uts/common/dtrace/dtrace.c*:

```
/*
 * If you're looking for the epicenter of DTrace, you just found it.  This
 * is the function called by the provider to fire a probe -- from which all
 * subsequent probe-context DTrace activity emanates.
 */
void
dtrace_probe(dtrace_id_t id, uintptr_t arg0, uintptr_t arg1,
    uintptr_t arg2, uintptr_t arg3, uintptr_t arg4)
```

`dtrace_probe()` - probe context

- **Runs in specialized context known as probe context**
  - Interrupts are disabled for the CPU executing the probe
  - Does no memory allocation and takes no locks
- **More details in *sys/cddl/contrib/opensolaris/uts/common/sys/dtrace.h*:**

```
 *   dtrace_probe() may be called in virtually any context:  kernel, user,
 *   interrupt, high-level interrupt, with arbitrary adaptive locks held, with
 *   dispatcher locks held, with interrupts disabled, etc.  The only latitude
 *   that must be afforded to DTrace is the ability to make calls within
 *   itself (and to its in-kernel subroutines) and the ability to access
 *   arbitrary (but mapped) memory.  On some platforms, this constrains
 *   context.  For example, on UltraSPARC, dtrace_probe() cannot be called
 *   from any context in which TL is greater than zero.  dtrace_probe() may
 *   also not be called from any routine which may be called by dtrace_probe()
 *   -- which includes functions in the DTrace framework and some in-kernel
 *   DTrace subroutines.  All such functions "dtrace_"; providers that
 *   instrument the kernel arbitrarily should be sure to not instrument these
 *   routines.
```

DTrace-ing DTrace

- Generally not recommended
- In sys/cddl/dev/fbt/fbt.c:

```
static void
fbt_provide_module(void *arg, modctl_t *lf)
{
    ...
    /*
     * Employees of dtrace and their families are ineligible.  Void
     * where prohibited.
     */
    if (strcmp(modname, "dtrace") == 0)
        return;
```

Writing a Provider

- Providers are kernel modules that register a set of callbacks with DTrace framework
- See `prototype` for a skeleton provider, *sys/cddl/dev/prototype.c*
- See `dtmalloc` for a simple real provider, *sys/cddl/dev/dtmalloc/dtmalloc.c*

```
static dtrace_pops_t dtmalloc_pops = {
        dtmalloc_provide,
        NULL,
        dtmalloc_enable,
        dtmalloc_disable,
        ...
};
```

```
    if (dtrace_register("dtmalloc", &dtmalloc_attr, DTRACE_PRIV_USER,
        NULL, &dtmalloc_pops, NULL, &dtmalloc_id) != 0)
        return;
```

Providers We'll Discuss

- **FBT (Function Boundary Tracing) - Kernel function tracing**
- **pid - Userspace function tracing**
- **SDT (Statically Defined Tracing)**
  - USDT (User SDT) exists, but we won't discuss it
- **FBT and pid use similar runtime patching techniques**
  - No disabled probe overhead

FBT and CTF (Compact C Type Format)

- Like DWARF, but simpler
- How DTrace gets debugging info (e.g., function arguments)
- Used to create function entry/return probes (FBT)
- CTF is generated during the build process
  - `ctfconvert`: converts DWARF to CTF (`.SUNW_ctf` section)
  - `ctfmerge`: merges multiple CTF sections

```
--- clock.o ---
ctfconvert -L VERSION -g clock.o
...
--- buildkernel ---
ctfmerge -L VERSION -g -o kernel.full ...
```

```
# readelf  -S /boot/kernel/kernel|grep -A 1 ctf
  ...
  [42] .SUNW_ctf         PROGBITS         0000000000000000  018aaa38
       00000000000d86bc  0000000000000000          43     0     4
```

FBT Probe Insertion Observed via kgdb

- **sys_getpid, no DTrace**

```
<sys_getpid>          push   %rbp
<sys_getpid+1>        mov    %rsp,%rbp
<sys_getpid+4>        mov    0x8(%rdi),%rax
<sys_getpid+8>        movslq 0xbc(%rax),%rax
```

- **sys_getpid, while running**
  `dtrace -n 'fbt:kernel:sys_getpid:entry'`
  - `int3` replaces first `sys_getpid` instruction
  - `push` instruction is emulated in the kernel
  - Original `push` instruction is restored after dtrace(1) exits

```
<sys_getpid>          int3
<sys_getpid+1>        mov    %rsp,%rbp
<sys_getpid+4>        mov    0x8(%rdi),%rax
<sys_getpid+8>        movslq 0xbc(%rax),%rax
```

FBT Patching Value

- x86-64: `int $3` in *sys/cddl/dev/fbt/x86/fbt_isa.c*

```
#define  FBT_PATCHVAL            0xcc
```

- ARMv8 (64-bit): `brk 0x40d` in *sys/cddl/dev/fbt/aarch64/fbt_isa.c*

```
#define      FBT_PATCHVAL            (AARCH64_BRK | AARCH64_BRK_IMM16_VAL)
```

- RISC-V: `sbreak` in *sys/cddl/dev/fbt/riscv/fbt_isa.c*

```
#define        FBT_PATCHVAL            (RISCV_INSN_BREAK)
```

FBT Trap Handling on ARMv8

- **Exception handler eventually calls `dtrace_probe()`**
  - `do_el1h_sync` -> `dtrace_invop()` -> `fbt_invop()` -> `dtrace_probe()`
- **DTrace framework takes control once `dtrace_probe()` is called**
- **Similar process on x86-64 and RISC-V**

pid Provider

- Uses `int $0x3` on x86-64 (like FBT). No ARMv8 or RISC-V support yet.
- In *sys/cddl/contrib/opensolaris/uts/intel/sys/fasttrap_isa.h*:

```
#define FASTTRAP_INSTR                  0xcc
```

- In *sys/cddl/contrib/opensolaris/uts/intel/dtrace/fasttrap_isa.c*:

```
int
fasttrap_tracepoint_install(proc_t *p, fasttrap_tracepoint_t *tp)
{
    fasttrap_instr_t instr = FASTTRAP_INSTR;

    if (uwrite(p, &instr, 1, tp->ftt_pc) != 0)
        return (-1);
```

Defining Static Tracepoints (aka SDT)

- **Example: Tracing successful `exec()` calls using `proc` provider**

```
# dtrace -n 'proc:::exec-success { printf("%s", curpsinfo->pr_psargs); }'
```

- **`exec-success` SDT probe is defined in *sys/kern/kern_exec.c*:**

```
SDT_PROVIDER_DECLARE(proc);
...
SDT_PROBE_DEFINE1(proc, , , exec__success, "char *");
...
/*
 * In-kernel implementation of execve().  All arguments are assumed to be
 * userspace pointers from the passed thread.
 */
static int
do_execve(td, args, mac_p)
        ...
        SDT_PROBE1(proc, , , exec__success, args->fname);
```

SDT: Transfer to DTrace Framework

- **On Solaris and macOS, SDTs are nops that can be rewritten to eventually enter `dtrace_probe()`**
  - Like FBT probes, SDT probes will trap into the kernel when enabled
- **On FreeBSD currently, SDT probes are function pointers that can be set to `dtrace_probe()`**
  - `dtrace_probe()` installed in *sys/cddl/dev/sdt/sdt.c*

```
static void
sdt_load()
{
...
    sdt_probe_func = dtrace_probe;
```

- **`SDT_PROBE` calls `sdt_probe_func()` (aka `dtrace_probe()` if enabled) in *sys/sys/sdt.h***

```
#define SDT_PROBE(prov, mod, func, name, arg0, arg1, arg2, arg3, arg4)  do {    \
        ...
        (*sdt_probe_func)(sdt_##prov##_##mod##_##func##_##name->id, \
            (uintptr_t) arg0, (uintptr_t) arg1, (uintptr_t) arg2,   \
            (uintptr_t) arg3, (uintptr_t) arg4);             \
        ...
```

Misc. DTrace Source

- **D translators:**
  - *cddl/lib/libdtrace*
- **D scripts:**
  - *share/dtrace*
- **DTrace testsuite:**
  - *cddl/usr.sbin/dtrace/tests*
  - *cddl/contrib/opensolaris/cmd/dtrace/test*

Resources

- DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD
- Solaris Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris
- The Design and Implementation of the FreeBSD Operating System, Second Edition
- Dynamic Tracing Guide
- FreeBSD Handbook
- Dynamic Instrumentation of Production Systems (USENIX 2004)
- Hidden in Plain Sight (ACM Queue, 2006)
- DTrace for BSD (BSDCan 2008)
- The DTrace backend on Solaris for x86/x64
- DTrace developer blogs
- awesome-dtrace.com

Thanks

- George Neville-Neil
- Robert Watson
- Mark Johnston
- Ruslan Bukin
- Andrew Turner
- Samuel Lepetit
- Everyone who's hacked DTrace on any platform

Summary

- **DTrace is a powerful tool**
- **We've seen how it works at a high level**
  - DTrace 101
  - DTrace Architecture
  - DTrace on FreeBSD (With Code)
- **Lots more code to dig through**
  - Exercise for the reader