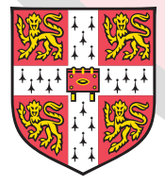
Adding DTrace probes to your favourite kernel subsystem

Robert N. M. Watson
University of Cambridge
Computer Laboratory

6 May 2009



UNIVERSITY OF CAMBRIDGE

The plan

- Quick introduction to DTrace
- Why not just use fbt?
- Implementing static probes
  - Examples: callouts
- Implementing dynamic probes
  - Example: NFSv23 client

What is DTrace?

DTrace is a comprehensive dynamic tracing framework for ~~the Solaris Operating Environment~~ FreeBSD.

DTrace provides a powerful infrastructure to permit administrators, developers, and service personnel to concisely answer arbitrary questions about the behavior of the operating system and user programs.

How DTrace works

- "D" scripts describe actions to be taken when *probes* fire in kernel or applications
- Scripts execute in both userspace, kernel
- "D" scripts have access to C types, global and probe-specific variables
- Aggregate types allow SMP-efficient data storage and manipulation

Where do probes come from?

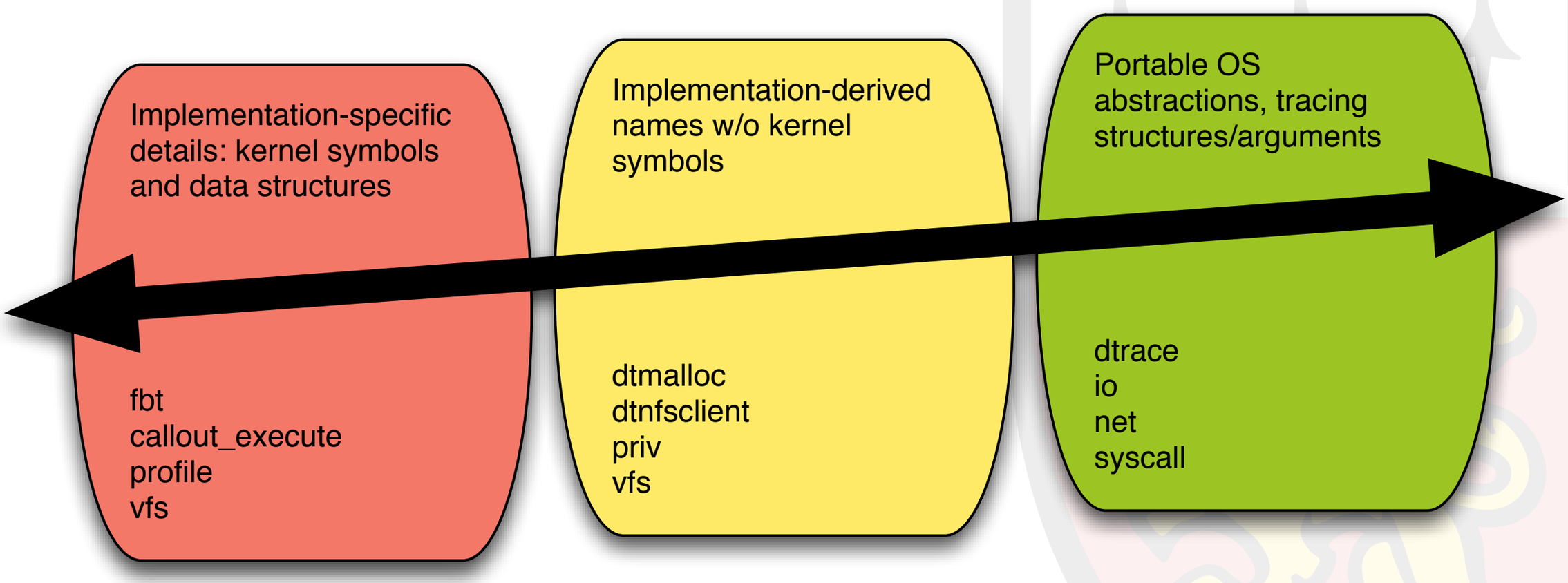
- *Providers* implement sets of related probes
- FBT - Function Boundary Tracing
  - Probes in all function prologues/epilogues
- SDT - Statically Defined Tracing
  - Explicit developer-defined probes
- And many others...

Why not just use fbt?

- Sub-function probe granularity
- Inlining and compiler quirks
- Abstract across multiple implementations
- Data-centric probes
- Portable/maintainable interfaces

Stability spectrum

Developer-centric **User-centric**



Example: timing callouts

```
#pragma D option quiet
callout_execute:::callout_start
{
        self->cstart = timestamp;
}

callout_execute:::callout_end
{
        @callouts[((struct callout *)arg0)->c_func] = sum(timestamp -
            self->cstart);
}

tick-1sec
{
        printa("%40a %10@d\n", @callouts);
        clear(@callouts);
        printf("\n");
}

BEGIN
{
        printf("%40s | %s\n", "function", "nanoseconds per second");
}
```

self allows per-thread, per-script variables to be used in DTrace

@callouts is an aggregate data structure, indexed by the *c_func* function pointer in the kernel's *struct callout*.

sum() adds sampled time deltas in an SMP-friendly manner

DTrace automatically splits script execution between userspace and the kernel

Example: timing callouts

```
cinnamon-freebsd# dtrace -s callout.d
                                function | nanoseconds per second
                          kernel`logtimeout       7242
                          kernel`pffasttimo       8847
                       kernel`tcp_isn_tick      10796
                                0xc89f3ec0      14107
                          kernel`scrn_timer      18264
                       kernel`dcons_timeout      31084
                          kernel`pfslowtimo      32020
                       kernel`atkbd_timeout      51523
                      kernel`sleepq_timeout   37685360
^C
```

Example: callout time distribution

```
#pragma D option quiet
callout_execute:::callout_start
{
        self->cstart = timestamp;
}

callout_execute:::callout_end
{
        @callouts = quantize(timestamp - self->cstart);
}
```

@callouts is an aggregate data structure, indexed by the *c_func* function pointer in the kernel's *struct callout*.

sum() adds sampled time deltas in an SMP-friendly manner

Example: callout time distribution

```
cinnamon-freebsd# dtrace -s callout_distribution.d
^C
           value  ------------- Distribution ------------- count
             512 |                                         0
            1024 |                                         20
            2048 |@@@@@@@@@@@@@@@@@@@@@@@@                 1257
            4096 |@@@@@@@@                                 466
            8192 |@@                                       143
           16384 |@@@                                      209
           32768 |@@@@                                     222
           65536 |@                                        43
          131072 |                                         1
          262144 |                                         1
          524288 |                                         0
         1048576 |                                         0
         2097152 |                                         1
         4194304 |                                         1
         8388608 |                                         3
        16777216 |                                         26
        33554432 |                                         0
```

Static probes

- Similar to KTR
- Declare named probes, argument types
- Invoke probes at strategic moments

Static probes: kern_timeout.c

```
59 SDT_PROVIDER_DEFINE(callout_execute);
60 SDT_PROBE_DEFINE(callout_execute, kernel, , callout_start);
61 SDT_PROBE_ARGTYPE(callout_execute, kernel, , callout_start, 0,
62     "struct callout *");
63 SDT_PROBE_DEFINE(callout_execute, kernel, , callout_end);
64 SDT_PROBE_ARGTYPE(callout_execute, kernel, , callout_end, 0,
65     "struct callout *");

...

408                                 THREAD_NO_SLEEPING();
409                                 SDT_PROBE(callout_execute, kernel, ,
410                                     callout_start, c, 0, 0, 0, 0);
411                                 c_func(c_arg);
412                                 SDT_PROBE(callout_execute, kernel, ,
413                                     callout_end, c, 0, 0, 0, 0);
414                                 THREAD_SLEEPING_OK();
```

DTrace will capture additional context, such as timestamps or stack traces, only if required by the script.

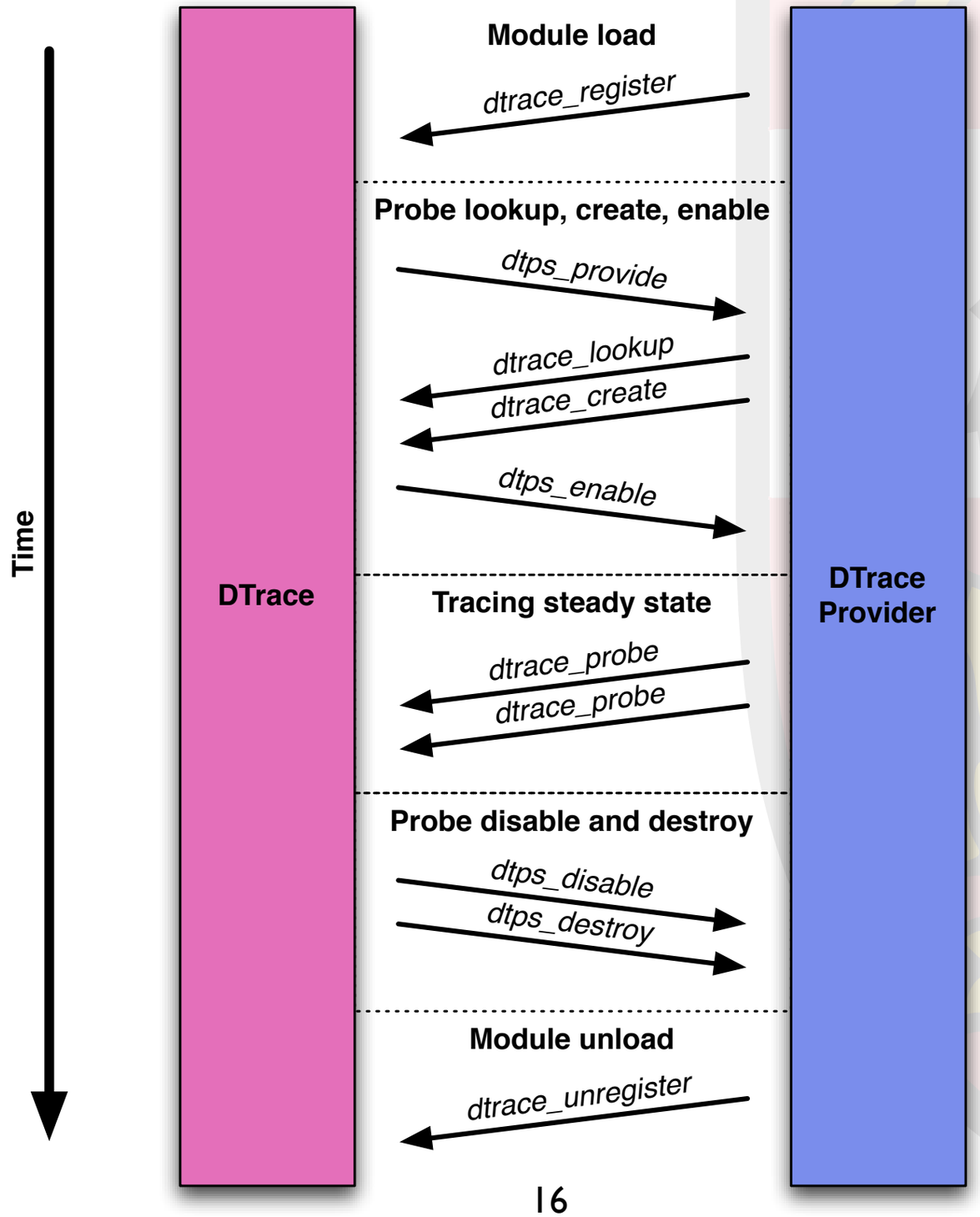
Dynamic probes

- Implement new providers in C code
- Dynamically extend probe set at run-time
- Many possible probes for one point in code
  - E.g., dtmalloc adds probes for new types
- DTrace queries available probes, enable/disable probes, argument types and values

What providers do

- Attach using *dtrace_register*
- Create and manage probes using *dtrace_probe_create*, *dtrace_probe_lookup*
- Handle *provide*, *destroy*, *enable*, *disable*, *getargdesc*, requests from DTrace
- Invoke *dtrace_probe* for enabled probes
- Detach using *dtrace_unregister*

Life cycle of a provider



Dynamic probes: NFSv23 client RPCs

- Modules: nfs2, nfs3, attrcache, accesscache
- src/sys/nfsclient/nfs_kdtrace.[ch]
- A single instrumentation point in the RPC layer triggers probes for NFSv2 and NFSv3
- NFS client contains explicit state and instrumentation to assist dtnfsclient
- We look at only RPC portion of provider

State in the NFS client

```
77 #ifdef KDTRACE_HOOKS
78 #include <sys/dtrace_bsd.h>
79
80 dtrace_nfsclient_nfs23_start_probe_func_t
81     dtrace_nfsclient_nfs23_start_probe;
82
83 dtrace_nfsclient_nfs23_done_probe_func_t
84     dtrace_nfsclient_nfs23_done_probe;
85
86 /*
87  * Registered probes by RPC type.
88  */
89 uint32_t        nfsclient_nfs2_start_probes[NFS_NPROCS];
90 uint32_t        nfsclient_nfs2_done_probes[NFS_NPROCS];
91
92 uint32_t        nfsclient_nfs3_start_probes[NFS_NPROCS];
93 uint32_t        nfsclient_nfs3_done_probes[NFS_NPROCS];
94 #endif
```

Start and done probe function pointers will point to functions in nfs_kdtrace.c when in use.

Arrays of probe IDs indexed by RPC procedure number have non-0 values when a probe is active, and allow NFS to tell DTrace which event is occuring

Provider definition

```
57 static void     dtnfsclient_getargdesc(void *, dtrace_id_t, void *,
58                     dtrace_argdesc_t *);
59 static void     dtnfsclient_provide(void *, dtrace_probedesc_t *);
60 static void     dtnfsclient_destroy(void *, dtrace_id_t, void *);
61 static void     dtnfsclient_enable(void *, dtrace_id_t, void *);
62 static void     dtnfsclient_disable(void *, dtrace_id_t, void *);

....

140 static dtrace_pops_t dtnfsclient_pops = {
141          dtnfsclient_provide,
142          NULL,
143          dtnfsclient_enable,
144          dtnfsclient_disable,
145          NULL,
146          NULL,
147          dtnfsclient_getargdesc,
148          NULL,
149          NULL,
150          dtnfsclient_destroy
151 };
152
153 static dtrace_provider_id_t     dtnfsclient_id;
```

Some provider entry points are optional, but *dtrace_register* will return EINVAL if an unsupported combination is requested.

dtrace_register

```
484 static void
485 dtnfsclient_load(void *dummy)
486 {
487
488         if (dtrace_register("nfsclient", &dtnfsclient_attr,
489             DTRACE_PRIV_USER, NULL, &dtnfsclient_pops, NULL,
490             &dtnfsclient_id) != 0)
491                 return;
492
493         dtrace_nfsclient_nfs23_start_probe =
494             (dtrace_nfsclient_nfs23_start_probe_func_t)dtrace_probe;
495         dtrace_nfsclient_nfs23_done_probe =
496             (dtrace_nfsclient_nfs23_done_probe_func_t)dtrace_probe;
497 }

...

537 SYSINIT(dtnfsclient_load, SI_SUB_DTRACE_PROVIDER, SI_ORDER_ANY,
538     dtnfsclient_load, NULL);
```

Probe function pointers are typedefs of *dtrace_probe*, to allow argument type checking

Per-RPC probe state

```
73 /*
74  * Description of NFSv3 and (optional) NFSv2 probes for a procedure.
75  */
76 struct dtnfsclient_rpc {
77         char            *nr_v3_name;
78         char            *nr_v2_name;    /* Or NULL if none. */
79
80         /*
81          * IDs for the start and done cases, for both NFSv2 and NFSv3.
82          */
83         uint32_t        nr_v2_id_start, nr_v2_id_done;
84         uint32_t        nr_v3_id_start, nr_v3_id_done;
85 };
86
87 /*
88  * This table is indexed by NFSv3 procedure number, but also used for NFSv2
89  * procedure names.
90  */
91 static struct dtnfsclient_rpc   dtnfsclient_rpcs[NFS_NPROCS] = {
92         { "null", "null" },
93         { "getattr", "getattr" },
94         { "setattr", "setattr" },
95         { "lookup", "lookup" },
```

dtps_provide

```
308 static void
309 dtnfsclient_provide(void *arg, dtrace_probedesc_t *desc)
310 {
311         int i;
312
313         if (desc != NULL)
314                 return;
```

When *dpts_provide* is invoked, look up our probes and, if not found, create them.

This approach is lazy but not unusual: we don't have to implement *dtps_destroy*.

```
...
401         /*
402          * Register NFSv3 RPC procedures.
403          */
404         for (i = 0; i < NFS_NPROCS; i++) {
405                 if (dtrace_probe_lookup(dtnfsclient_id, dtnfsclient_nfs3_str,
406                     dtnfsclient_rpcs[i].nr_v3_name, dtnfsclient_start_str) ==
407                     0) {
408                         dtnfsclient_rpcs[i].nr_v3_id_start =
409                             dtrace_probe_create(dtnfsclient_id,
410                             dtnfsclient_nfs3_str,
411                             dtnfsclient_rpcs[i].nr_v3_name,
412                             dtnfsclient_start_str, 0,
413                             &nfsclient_nfs3_start_probes[i]);
414                 }
```

Pointer to NFS client probes array

dtps_enable, dtps_disable

```
433 static void
434 dtnfsclient_enable(void *arg, dtrace_id_t id, void *parg)
435 {
436         uint32_t *p = parg;

...

456                 *p = id;
457 }


459 static void
460 dtnfsclient_disable(void *arg, dtrace_id_t id, void *parg)
461 {
462         uint32_t *p = parg;
...
480         else
481                 *p = 0;
482 }
```

enable and *disable* are implemented by pushing a non-zero probe ID into the NFS client's per-RPC arrays so that the NFS client will start invoking the probe function for it

NFS client invoking the probes

```
491 #ifdef KDTRACE_HOOKS
492         if (dtrace_nfsclient_nfs23_start_probe != NULL) {
493                 uint32_t probe_id;
494                 int probe_procnum;
495
496                 if (nmp->nm_flag & NFSMNT_NFSV3) {
497                         probe_id = nfsclient_nfs3_start_probes[procnum];
498                         probe_procnum = procnum;
499                 } else {
500                         probe_id = nfsclient_nfs2_start_probes[procnum];
501                         probe_procnum = nfsv2_procid[procnum];
502                 }
503                 if (probe_id != 0)
504                         (dtrace_nfsclient_nfs23_start_probe)(probe_id, vp,
505                             mreq, cred, probe_procnum);
506         }
507 #endif
```

dtrace_nfsclient_nfs23_start_probe is simply an alias for *dtrace_probe*. Provider can do more interesting things, such as map kernel-internal structures to more portable representations.

dtps_destroy

Because we don't allocate any probe state, we have nothing to free here--some modules will need to do so, however.

DTrace requires that *dtps_destroy* always be defined, even if empty.

```
428 static void
429 dtnfsclient_destroy(void *arg, dtrace_id_t id, void *parg)
430 {
431 }
```

dtrace_unregister

```
500 static int
501 dtnfsclient_unload()
502 {
503         int error = 0;
504
505         dtrace_nfsclient_nfs23_start_probe = NULL;
506         dtrace_nfsclient_nfs23_done_probe = NULL;
507
508         if ((error = dtrace_unregister(dtnfsclient_id)) != 0)
509                 return (error);
510
511         return (error);
512 }
```

Clear the NFS client probe pointers to prevent further calls into the provider.

BUG: *dtrace_unregister* failure not handled!

dtps_getargdesc

```
201 static void
202 dtnfsclient_getargdesc(void *arg, dtrace_id_t id, void *parg,
203     dtrace_argdesc_t *desc)
204 {
205         const char *p = NULL;
206
...
279         } else {
280                 switch (desc->dtargd_ndx) {
281                 case 0:
282                         p = "struct vnode *";
283                         break;
284                 case 1:
285                         p = "struct mbuf *";
286                         break;
287                 case 2:
288                         p = "struct ucred *";
289                         break;
290                 case 3:
291                         p = "int";
292                         break;
293                 case 4:
294                         if (dtnfs23_isdoneprobe(id)) {
295                                 p = "int";
296                                 break;
297                         }
298                         /* FALLSTHROUGH */
299                 default:
300                         desc->dtargd_ndx = DTRACE_ARGNONE;
301                         break;
302                 }
303         }
304         if (p != NULL)
305                 strlcpy(desc->dtargd_native, p, sizeof(desc->dtargd_native));
306 }
```

Return C types of probe arguments as strings, or *DTRACE_ARGNONE*.

Notice that *dtnfsclient_getargdesc* returns different types for different probes.

Check that it's there

```
cinnamon-freebsd# dtrace -lP nfsclient
   ID   PROVIDER            MODULE                          FUNCTION NAME
  528  nfsclient       accesscache                          flush done
  529  nfsclient       accesscache                            get hit
  530  nfsclient       accesscache                           get miss
  531  nfsclient       accesscache                          load done
  532  nfsclient         attrcache                          flush done
  533  nfsclient         attrcache                            get hit
  534  nfsclient         attrcache                           get miss
  535  nfsclient         attrcache                          load done
  536  nfsclient              nfs2                          null start
  537  nfsclient              nfs2                          null done
  538  nfsclient              nfs2                       getattr start
  539  nfsclient              nfs2                       getattr done
  540  nfsclient              nfs2                       setattr start
  541  nfsclient              nfs2                       setattr done
  542  nfsclient              nfs2                        lookup start
  543  nfsclient              nfs2                        lookup done
  544  nfsclient              nfs2                      readlink start
  545  nfsclient              nfs2                      readlink done
  546  nfsclient              nfs2                        read start
  547  nfsclient              nfs2                        read done
  548  nfsclient              nfs2                       write start
....
```

Example: RPC time distribution

```
syscall:::entry
{
        self->count = 0;
}

nfsclient:nfs3::start
{
        self->timestamp = timestamp;
}

nfsclient:nfs3::done
{
        self->count += (timestamp - self->timestamp);
}

syscall:::return
/self->count != 0/
{
        @syscalls[probefunc] = quantize(self->count);
}
```

Example: RPC time distribution

```
cinnamon-freebsd# dtrace -s nfsclient.d
...
  open
           value  ------------- Distribution ------------- count
               8 |                                         0
              16 |@@@                                      19
              32 |                                         0
...
           16384 |                                         0
           32768 |                                         1
           65536 |                                         1
          131072 |                                         2
          262144 |@@@                                      20
          524288 |@@@@@@@@@@@@@@@@                         106
         1048576 |@@@@@@@@@@                               67
         2097152 |@                                        8
         4194304 |@                                        5
         8388608 |@@                                       15
        16777216 |@@                                       14
        33554432 |@@                                       15
        67108864 |@                                        7
       134217728 |                                         0
```

Where to learn more

- DTrace User Guide
- Solaris Dynamic Tracing Guide ← read this
- src/sys/cddl/contrib/opensolaris/uts/common/sys/dtrace.h

Conclusion

- We have learned about...
  - How DTrace can allow you to quickly analyze kernel behavior, including timing
  - Implement static probes
  - Implement a new provider