DTrace

David Evans

February 24, 2009

What I'll discuss

1. What's DTrace all about
2. DTrace mechanisms
3. Instrumenting SBus
4. The end, and after

What I won’t discuss

- The D language in detail
- Strategies for tracing things
- Useful recipes

1. What's DTrace all about

What does DTrace do?

- Provides probe points

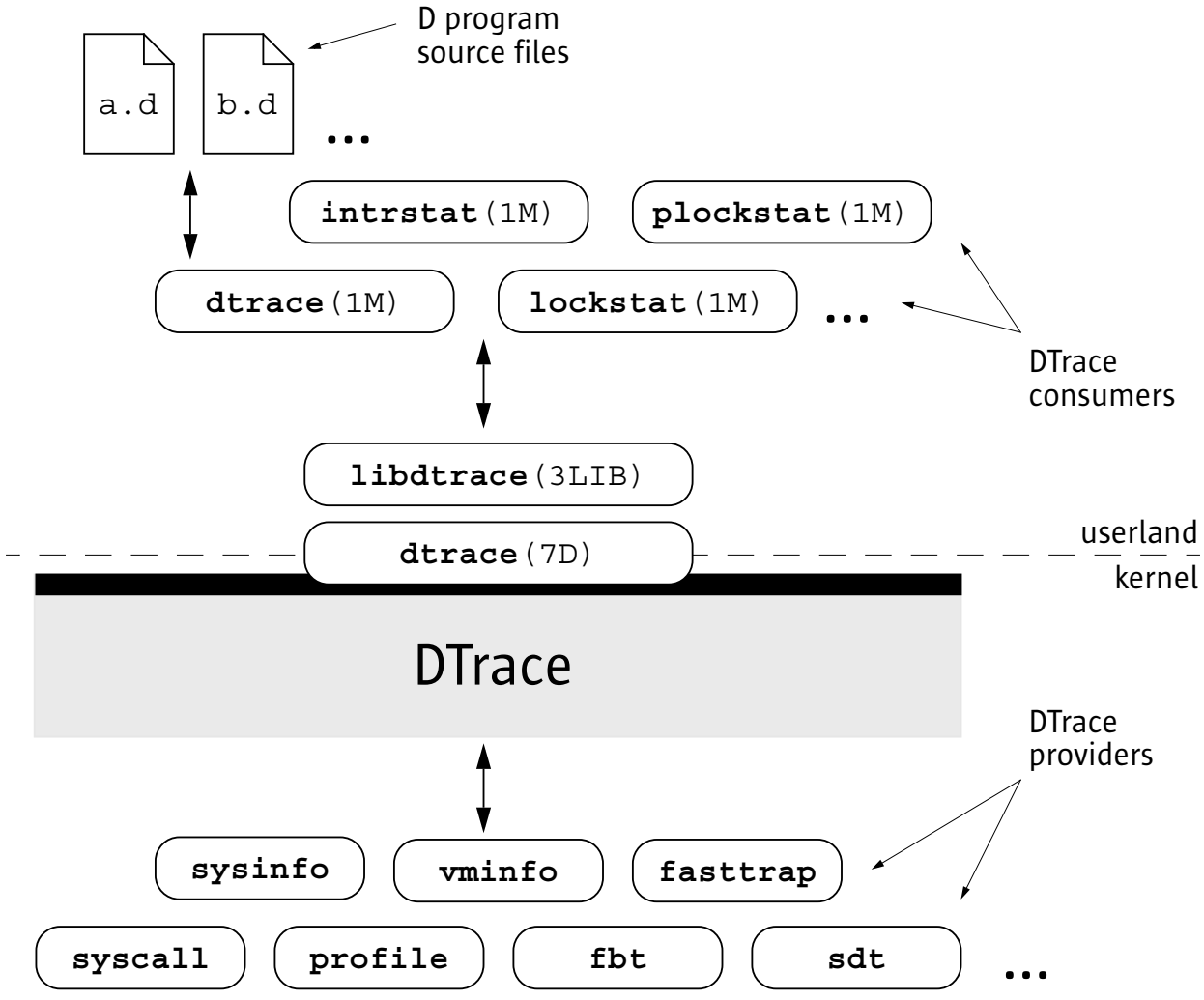
```
# dtrace -l | wc -l
   24009
```

- Allows straightforward event-driven measurement

Sun propaganda

If you have ever wanted to understand the behavior of your system, DTrace is the tool for you. DTrace is a comprehensive dynamic tracing facility that is built into Solaris. The DTrace facility can be used to examine the behavior of user programs. The DTrace facility can also be used to examine the behavior of the operating system. DTrace can be used by system administrators or application developers, and is suitable for use with live production systems. DTrace will allow you to explore your system to understand how it works, track down performance problems across many layers of software, or locate the cause of aberrant behavior.

The DTrace architecture



2. DTrace mechanisms

Probes

```
provider:module:function:name
```

Providers

```
# dtrace -l | awk '{print $2}' | ... |
   sort | uniq
...
dtrace
fbt
io
lockstat
mach_trap
mds32
proc
profile
syscall
vminfo
```

Actions

```
syscall::entry
{
  @c[execname] = count();
}
```

Actions with predicates

```
syscall::::return
/ arg0 == -1 /
{
  @c[execname, probefunc] = count();
}
```

D language features

- Variables: lots of types
- Aggregations

$$f(f(x_0) \cup f(x_1) \cup \ldots \cup f(x_n)) = f(x_0 \cup x_1 \cup \ldots \cup x_n)$$

- Thread local- (`self->`) and clause local- (`this->`) variables
- Structs, unions, C preprocessor, ...

Strings and address spaces

- DTrace scripts run in the kernel
- Getting at data in user space requires `copyin()`
- `copyinstr()` is a favourite

Speculative tracing

```
#pragma D option nspec=100

syscall::stat64:entry
{
  self->spec = speculation();
  speculate(self->spec);
  printf("path of failed %s by pid %d is %s",
      probefunc, pid,
      stringof(copyinstr(arg0)));
}
```

Speculative tracing

```
syscall::stat64:return
/ self->spec /
{
  speculate(self->spec);
  printf("errno is %d", errno);
}
```

Speculative tracing

```
syscall::stat64:return
/ self->spec && errno != 0 /
{
  commit(self->spec);
  self->spec = 0;
}
```

Speculative tracing

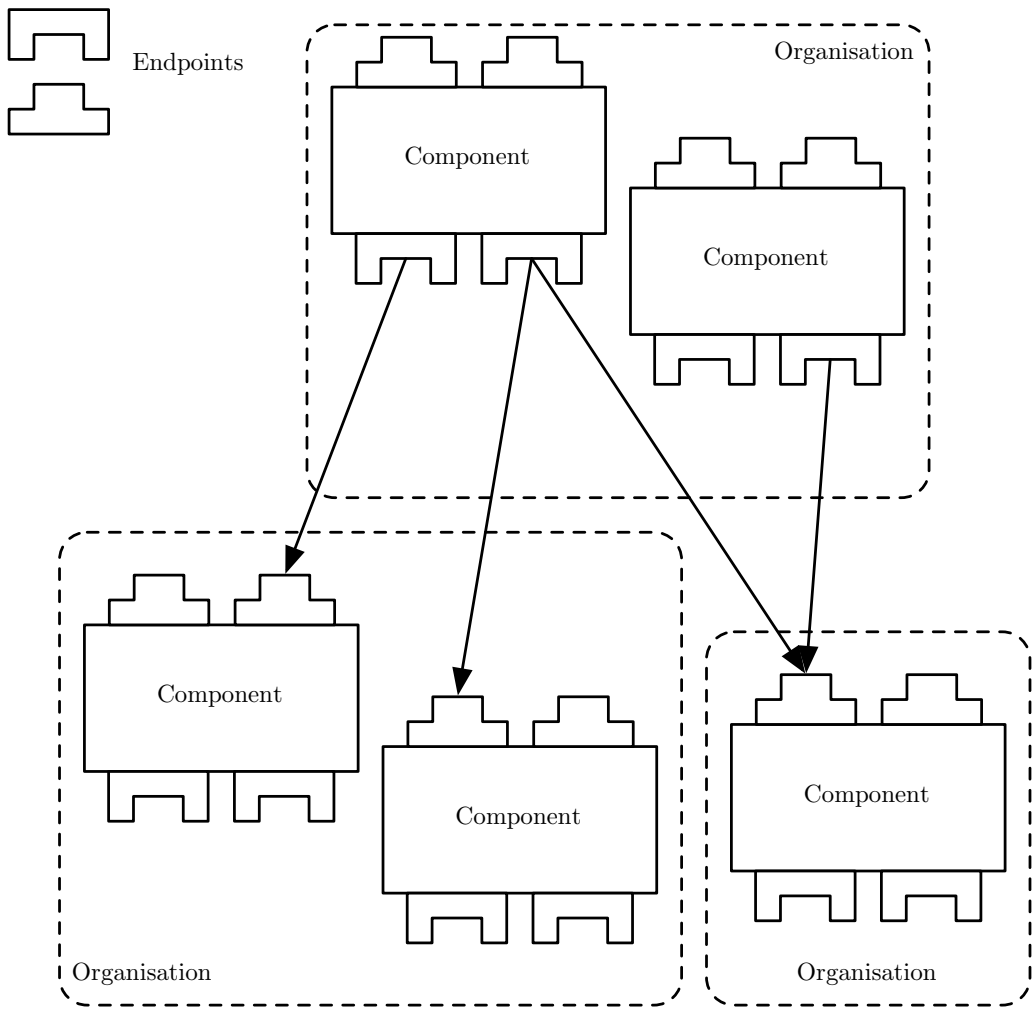
```
syscall::stat64:return
/ self->spec && errno == 0 /
{
  discard(self->spec);
  self->spec = 0;
}
```

3. Instrumenting SBus

What is SBus?

- Messaging middleware
- Component-based
- Designed with streams of messages in mind

Architecture



Metrics of interest

- RPC response time
- Component transit time

Modifications

```
smessage *sendpoint::rcv()
{
...
  if(SBUS_RCV_ENABLED()) {
    inc->tree->cookie = 0;
    SBUS_RCV(name, type, inc->tree->cookie);
  }
}
```

Modifications

```
void sendpoint::reply(smessage *query,
    snode *result, int exception,
    HashCode *hc)
{
  if(SBUS_REPLY_ENABLED())
    SBUS_REPLY(name, type, result->cookie);
...
}
```

The D side

```
#pragma D option mangled
#pragma D option quiet
/*
 * This is copied from component.h.  It is a
 * very good idea to keep them in sync.
 */
enum EndpointType {
  EndpointServer,
  EndpointClient,
  EndpointSource,
  EndpointSink
};
```

The D side

```
struct message_info {
  long cookie;
  uint64_t arrival;
  string ep_name;
};

struct message_info rpc_msgs[string];
```

The D side

```
sbus$target:::rcv
/ arg1 == EndpointServer /
{
  this->ep_name = copyinstr(arg0);
  rpc_msgs[this->ep_name].cookie = arg2;
  rpc_msgs[this->ep_name].ep_name =
      this->ep_name;
  rpc_msgs[this->ep_name].arrival = timestamp;
}
```

The D side

```
sbus$target:::reply
/ arg1 == EndpointServer /
{
  msg = rpc_msgs[copyinstr(arg0)];
  @rpc_time[msg.ep_name] = avg(timestamp -
      msg.arrival);
  @rpcs[msg.ep_name] = count();
}
```

Glue

```
provider sbus
{
  probe rcv(char *, int, long);
  probe reply(char *, int, int);
};
```

and

```
#include "sbusProvider.h"
```

4. The end, and after

What DTrace lets you do

- Measure things in the OS (find bugs in apps!)
- Prepare your apps for instrumentation fairly easily

Why you (probably) can't have it

- Only for Solaris, Opensolaris, and MacOS X
- Maybe for FreeBSD
- Linux people seem to mistrust it...

For further information

- *Solaris Dynamic Tracing Guide*
- Sun BigAdmin info
- Greg Miller, “Exploring Leopard with DTrace”
- Bryan Cantrill, “DTrace Review”
- DTrace toolkit