# DTrace

Crash Dump Analysis 2014/2015

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

# DTrace

- **Dynamic Tracing**
  - Production systems observability
    - Safety
    - Ideally zero overhead if inactive
    - Minimal overhead if active
    - No special debugging builds required
  - Total observability
    - Global overview of the system state
    - Merging and correlating data from multiple sources

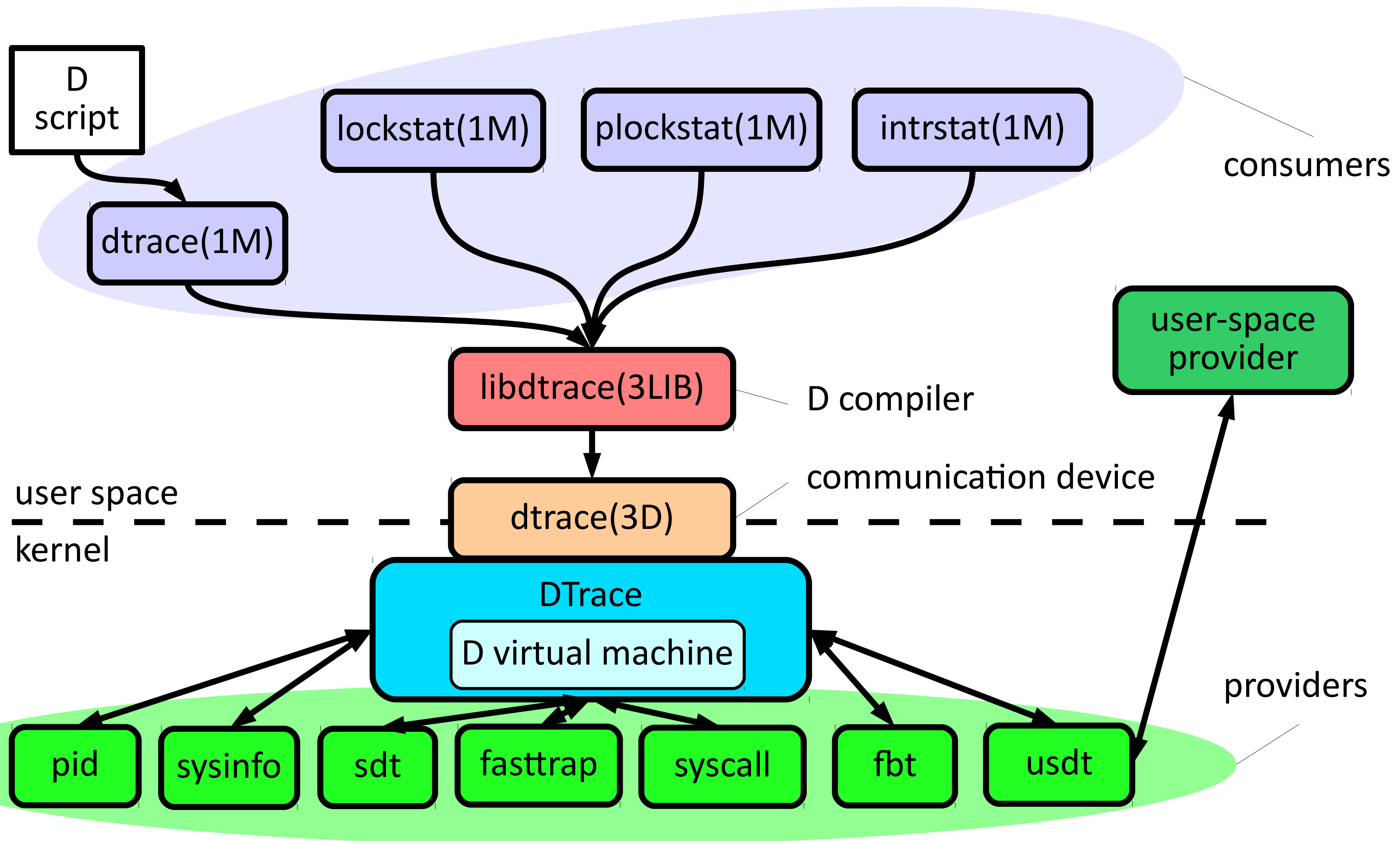
redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Terminology

- **Probe**
  - Location that can be observed
    - A specification of an event, a piece of code, etc.
    - If a probe is **activated** and the event happens (the piece of code is executed), then the probe is **fired**
- **Provider**
  - Provides probes and takes care of their activation, firing and deactivation
- **Consumer**
  - Consumes and post-processes data from fired probes

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Architecture



# History

- **January 31st 2005**
  - Integrated into Solaris 10
  - Released as open source
    - CDDL license, first piece of future OpenSolaris release
- **October 27th 2007**
  - Released as part of Mac OS X 10.5 (Leopard)
- **January 6th 2009**
  - Released as part of FreeBSD 7.1 (integrated on September 2nd 2008)
- **February 21st 2010**
  - Ported to NetBSD (i386, amd64, evbarm; not enabled by default)

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# History (2)

- **DTrace on Linux**
  - Beta releases at least since 2008
  - Major issue: License incompatibility (CDDL vs. GPL)
    - Standalone DTrace module
      - No modifications to core kernel sources
      - Still taints the kernel
      - Only a limited set of providers (fbt, syscall, usdt)
    - Isolated glue code under GPL
      - Support for more providers possible
  - Available in Oracle Linux (since December 2012)

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# History (3)

- **QNX**
  - Port still in progress (since 2010)
- **3rd party DTrace probe providers**
  - Apache
  - MySQL
  - PostgreSQL
  - X.Org
  - Firefox
  - Oracle JVM
  - Perl, Ruby, PHP

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# D language

- **Actions executed if a probe fires**
  - Syntax similar to C and AWK
  - Semantics avoids potentially dangerous constructs (branching, loops, changing the state, etc.)
  - A list of statements
    - Many fields of the statement can be omitted
      - Replaced by implicit default values

```
probe /predicate/ {
   actions
}
```

# Probes

- **Probes matched by four fields**
  - General syntax: `provider:module:function:name`
    - Fields can be omitted
      - `foo:bar` matches function *foo* and name *bar* in all available modules and all available providers
    - Fields can be empty
      - `syscall:::` matches all probes provided by the *syscall* provider

```
probe /predicate/ {
   actions
}
```

# Probes (2)

- **Probes matched by four fields (cont.)**
  - Shell pattern matching (wildcards *, ?, [], escape \)
    - `syscall::*lwp*:entry`
      matches all probes provided by the *syscall* provider, in any module, specifically all syscall *entry* points that contain the *lwp* string in them
  - Special probes implemented by the *dtrace* provider
    - `BEGIN`
    - `END`
    - `ERROR`

```
probe /predicate/ {
    actions
}
```

# Predicates

- **Guard for the actions**
  - Expression which evaluates as integer or pointer
    - Zero is false, non-zero is true
    - Any D operators, variables and constants
    - Implicitly true (if absent)

```
probe /predicate/ {
   actions
}
```

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Actions

- **List of statements**
  - Delimited by semicolon
  - No branching
  - No loops
  - Implicit default action (if empty)
    - Usually the probe name is printed out

```
probe /predicate/ {
   actions
}
```

# Types

- **Basic types**
  - Reflect the basic C types
  - Integer types and aliases
    - (unsigned/signed) char, short, int, long, long long
    - int8_t, int16_t, int32_t, int64_t, intptr_t, uint8_t, uint16_t, uint32_t, uint64_t, uintptr_t
  - Floating point types
    - float, double, long double
      - Values can be assigned, but no floating point arithmetics is implemented in DTrace

# Types (2)

- **Derived and special types**
  - Pointers
    - C-like pointers to other data structures
      - int *value, void *ptr
    - Pointer arithmetics
    - NULL constant is zero
    - Weak pointer safety
      - Invalid memory accesses are safe
      - No reference safety

# Types (3)

- **Derived and special types (cont.)**
  - Scalar arrays
    - C-like arrays of basic data types
      - Similar to pointers
      - Can be assigned as value
      - int values[5][6]
  - Strings
    - Special string type descriptor (instead of char *)
      - NULL-terminated character arrays
      - Can be assigned as value (char * assigns reference)
      - Internal strings are always allocated as bounded (predefined maximal length of 256 bytes)

# Types (4)

- **Composed types**
  - Structures
    - C-like structures
      - Members assessed via . and -> operators
      - Variables must be declared explicitly

```
struct callinfo {
    uint64_t ts;
    uint64_t calls;
};

struct callinfo info[string];
```

```
syscall::read:entry, syscall::write:entry {
    info[probefunc].ts = timestamp;
    info[probefunc].calls++;
}

END {
    printf("read  %d %d\n", info["read"].ts,
        info["read"].calls);
    printf("write %d %d\n", info["write"].ts,
        info["write"].calls);
}
```

# Types (5)

- **Composed types (cont.)**
  - Unions, Bit-fields, Enums, Typedefs
    - C-like structures
  - Inlines
    - Typed constants

```
inline string desc = "something";

enum typeinfo {
     CHAR_ARRAY = 0,
     INT_ARRAY,
     UINT_ARRAY
};
```

```
struct info {
     enum typeinfo disc;
     union {
          char c[4];
          int32_t i32;
          uint32_t u32;
     } value;

     int a : 3;
     int b : 4;
};

typedef struct info info_t;
```

# Operators

- **Arithmetic**
  - + - * / %
- **Logical**
  - && || ^^ !
    - Short-circuit evaluation
- **Relational**
  - < <= > >= == !=
    - Also lexical comparison on strings
- **Bitwise**
  - & | ^ << >> ~
- **Assignment**
  - = += -= *= /= %= &= |= ^= <<= >>=
- **Increment and decrement**
  - ++ --

# Operators (2)

- **Conditional expression**
  - *condition* ? *true_expression* : *false_expression*
    - Replacement for branching
- **Addressing, member access and sizes**
  - & * . -> sizeof(*type/expr*) offsetof(*type*, *member*)
- **Kernel variables access**
  - `
- **Typecasting**
  - (int) x, (int *) NULL, (string) *expr*, stringof(*expr*)

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Variables

- **Scalar variables**
  - Simple global variables
    - Storing fixed-size data (integers, pointers, fixed-size composed types, bounded strings)
    - Do not have to be declared (but can be), duck-typing

```
BEGIN {
    /* Implicitly declare
       an int variable */
    value = 1234;
}
```

```
/* Explicitly declare an int
   variable (initial value
   cannot be assigned here) */
int val;

BEGIN {
    value = 1234;
}
```

# Variables (2)

- **Associative arrays**
  - Global arrays of scalar values indexed by a key
    - Key signature is a list of scalar types
      - Integer, string or a tuple of scalar types
      - Declared implicitly (duck-typing by assignment) or explicitly
    - Fixed type values
      - Declared implicitly (duck-typing by assignment) or explicitly

```
array0[123, "key"] = 456;

int array1[unsigned int, string];
```

# Variables (3)

- **Thread-local variables**
  - Scalar variables or associative arrays specific to the current thread
    - Special self identifier
    - Uninitialized variables are zero-filled
      - Explicitly assigning zero deallocates the variable

```
/* Implicit declaration */

syscall::read:entry {
	/* Mark this thread */
	self->tag = 1;
}
```

```
/* Explicit declaration */
self int tag;

syscall::read:entry {
	/* Mark this thread */
	self->tag = 1;
}
```

# Variables (4)

- **Clause-local variables**
  - Scalar variables or associative arrays specific to the current probe clause
    - Special this identifier
    - Value kept for multiple clauses of the same probe
    - Not initialized to zero

```
/* Implicit declaration */

syscall::read:entry {
    this->value = 1;
}
```

```
/* Explicit declaration */
this int value;

syscall::read:entry {
    this->value = 1;
}
```

# Aggregations

- **Variables for storing statistical data**
  - Results of aggregative data computation
    - For aggregating functions *f*(...) which satisfy

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

  - Declared similarly as associative arrays

```
@values[123, "key"] = aggfunc(args);
@_[123, "key"] = aggfunc(args);        /* Simple variable */
@[123, "key"] = aggfunc(args);         /* dtto */
```

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Aggregations (2)

- **Aggregation functions**
  - count()
  - sum(*scalar*)
  - avg(*scalar*)
  - min(*scalar*)
  - max(*scalar*)
  - lquantize(*scalar, lower_bound, upper_bound, step*)
    - Linear frequency distribution
  - quantize(*scalar*)
    - Power-of-two frequency distribution

# Aggregations (3)

- **Implicit action**
  - Aggregations printed out in END

```
syscall:::entry {
    @counts[probefunc] = count();
}

# dtrace -s counts.d
dtrace: script 'counts.d' matched 235 probes
^C

  resolvepath                                                       8
  lwp_park                                                         10
  gtime                                                            12
  lwp_sigmask                                                      16
  stat64                                                           46
  pollsys                                                          93
  p_online                                                        256
  ioctl                                                          1695
#
```

# Built-in variables

- **Global variables defined by DTrace**
  - Various state-dependent values
    - int64_t arg0, arg1, ..., arg9
      - Input arguments for the current probe
    - args[]
      - Typed arguments for the current probe (e.g. function arguments with appropriate types)
    - uintptr_t caller
      - Instruction pointer of the current kernel thread just before the firing probe
    - uintptr_t ucaller
      - Instruction pointer of the current user space thread just before the firing probe
    - kthread_t *curthread
      - Current thread kernel structure
    - psinfo_t *curpsinfo
      - Current process state structur

# Built-in variables (2)

- string cwd
  - Current working directory
- string execname
  - Name which was used to execute the current process
- pid_t pid
  tid_t tid
  - Current PID, TID
- string probeprov, probemod, probefunc, probename
  - Current probe provider, module, function and name
- chipid_t chip
  processorid_t cpu
  cpuinfo_t *curcpu
  - Current CPU (or physical CPU chip)

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Built-in variables (3)

- uint_t stackdepth
  - Current thread stack depth
- uint_t ipl
  - Current interrupt priority level
- uid_t uid
  gid_t gid
  - Current UID and GID
- uint64_t timestamp
  - Current timestamp (in nanoseconds)
- uint64_t vtimestamp
  - Current virtual timestamp (in nanoseconds)
  - Abstracting Dtrace overhead (Dtrace predicates and actions)
- uint64_t walltimestamp
  - Wall-clock time stamp (nanoseconds since epoch)

# Action statements

- **Output recorded into a *trace buffer***
  - Most of the action statements produce some sort of textual output
    - trace(*expr*)
      - Output the value of an expression
    - tracemem(*address*, *bytes*)
      - Copy bytes from memory to the trace buffer
    - print(*format, …*)
      - Output safely formatted strings

# Action statements (2)

- printa(*aggregation*)
  printa(*format, aggregation*)
  - Start processing *aggregation* data
  - Runs asynchronously, thus actual output can be delayed
- stack()
  stack(*frames*)
  - Output kernel stack trace
- ustack()
  ustack(*frames*)
  - Output user space stack trace
  - Addresses to symbols are translated by the user space consumer (post-processing)

# Action statements (3)

- ustack(*frames*, *buffer_size*)
  - Output user space stack trace
  - Addresses to symbols are translated by the kernel
    - The output is bounded to buffer_size bytes
    - Run-time symbol annotations of the user space stack are required by the probe provider
      - JVM 1.5 or newer provide these annotations
- jstack()
  jstack(*frames*)
  jstack(*frames*, *buffer_size*)
  - Alias for ustack() with non-zero default *buffer_size*

# Output formatting

- **Conversion formats**
  - %a
    - Pointer as kernel symbol name
  - %c
    - ASCII character
  - %C
    - Printable ASCII or escape code
  - %d, %i, %o, %u, %x
  - %e
    - Float as [**-**]*d***.***ddd***e±***dd*
  - %f
    - Float as [**-**]*d***.***ddd***.***ddd*
  - %p
    - Hexadecimal pointer
  - %s
    - ASCII string
  - %S
    - ASCII string or escape codes

# Subroutines

- **Actions that alter the DTrace state**
  - Completely safe with respect to the system state
    - Manipulate only the local memory storage of DTrace (scratch memory)
  - Produce no output to the trace buffer
    - *alloca(*size*)
      - Allocate *size* bytes from the scratch memory
      - Released after the current clause ends
    - bcopy(**src*, **dest*, *size*)
      - Copy *size* bytes from kernel memory to scratch memory

# Subroutines (2)

- ***copyin(**src, size*)**
  - Copy *size* bytes from the user memory (of the current process) to scratch memory
- ***copyinstr(**src*)**
  - Copy NULL-terminated string from the user memory (of the current process) to scratch memory
- **mutex_ownded(**mutex*)**
  - Tell whether a kernel mutex is currently locked
- ***mutex_owner(**mutex*)**
  - Return a pointer to kthread_t of the thread which owns the given mutex (or NULL)
- **mutex_type_adaptive(**mutex*)**
  - Tell whether a kernel mutex is an adaptive mutex

# Subroutines (3)

- strlen(**str*)
  - Return the length of a NULL-terminated string
- *strjoin(**a*, **b*)
  - Concatenate two NULL-terminated strings
- *basename(**path*)
  - Return a basename of a path
- *dirname(**path*)
- *cleanpath(**path*)
  - Return a file system path without elements such as ../
- rand()
  - Return a (weak) pseudo-random number
- exit(*status*)
  - Exit the tracing session and return the given status to the consumer

# Destructive actions

- **Changing the state of the system**
  - Deterministic, but potentially dangerous in production environment
    - Need to be explicitly enabled using `dtrace -w`
    - stop()
      - Stop the current process (e.g. for dumping the core or attaching a debugger)
    - raise(*signal*)
      - Send a signal to the current process
    - panic()

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Destructive actions (2)

- copyout(**src*, **dest*, *size*)
  - Copy *size* bytes from the scratch memory to the user memory (of the current process)
  - Potential page faults and detected and avoided
- copyoutstr(**src*, **dest*, *size*)
  - Copy at most *size* bytes of a NULL-terminated string from the scratch memory to the user memory (of the current process)
- system(*program*, …)
  - Execute a program by a shell (*program* is a printf() format specifier)
- breakpoint()
  - Induce a breakpoint (if the kernel debugger is loaded, it is executed)

# Destructive actions (3)

- chill(*nanoseconds*)
  - Spin actively for a given number of nanoseconds
  - Useful for analyzing timing bugs

# Speculative tracing

- **Filtering events**
  - Predicates are for filtering out unimportant probes **before** they are fired
  - Speculative tracing is for filtering out unimportant probes eventually some time **after** they are fired
    - You can tell whether you are interested in the data from the *n*-th probe only after the (*n*+*k*)-th probe fired ($k > 0$)
    - Speculatively record all the data, but decide later whether to commit it or not

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Speculative tracing (2)

- speculation()
  - Create a new speculative trace buffer and return its ID
  - The number of speculative trace buffers is limited (by default to 1)
- speculate(*id*)
  - The rest of the clause will be recorded to the given speculative trace buffer
  - This must be the first data processing action in a clause
  - Aggregating and destructive actions are not allowed
- commit(*id*)
  - Commit the given speculative trace buffer to the trace buffer

redhat. SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Providers

- syscall
  - Tracing of kernel system calls
    - Probes for entry and exit points of a syscall
      - Access to the arguments (on entry, in arg0, etc.)
      - Access to the return value (on return, in arg0)
      - Access to kernel errno (in errno)
      - Access to kernel variables

# Providers (2)

- **fbt**
  - Function boundary tracing
    - Probes on function entry point and (all) exit points of almost all kernel functions
      - Except inlined and lead-optimized functions
    - entry
      - Access to the (typed) arguments (in args[])
    - return
      - Offset of the return instruction in arg0
      - Access to the (typed) return value (in args[1])

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Providers (3)

- **sdt**
  - Static kernel probes
    - Probes declared on arbitrary placees in the kernel code
      - Via a macro
    - Currently just a few actually defined
      - interrupt-start
        interrupt-complete
        - Pointer to dev_info structure in arg0

# Providers (4)

- **proc**
  - Probes corresponding to process and thread life-cycle
    - Creating a process (using fork() and friends)
    - Executing a binary
    - Exiting a process
    - Creating a thread
    - Destroying a thread
    - Receiving a signal

# Providers (5)

- sched
  - Kernel scheduler events
    - Changing of priorities
    - Thread being scheduled
    - Thread being preempted
    - Thread going to sleep
    - Thread waking up
- io
  - I/O events
    - Starting an I/O request
    - Finishing an I/O request
    - Waiting for a device

# Providers (6)

- **pid**
  - Tracing of user space functions
    - Probe firing does not enforce serialization
      - The traced process is never stopped
    - Boundary probes similar to fbt
      - Function entry and return probes
      - Arguments in arg0, arg1, …, arg9 are raw unfiltered int64_t values
    - Arbitrary function offset probes
    - User space symbol information is required to support symbolic function names
      - Standard shared libraries contain symbol information on Solaris

# Providers (7)

- **Many more providers**
  - vminfo (kernel memory management events)
  - mid (kernel network stack events)
  - profile (periodic timer events)
  - Application-specific providers
    - Usually static probes or specific events
    - X.Org, PostgreSQL, Firefox, etc.
  - VM-based providers
    - JVM, PHP, Perl, Ruby

# Total observability

- **Goal of combination of various providers**
  - Causal correlation of events and information from various level of the system
  - Examples
    - Which specific SQL transaction is generating a particular I/O load?
    - What Java methods trigger specific library calls and what kernel syscalls are triggered by them?
    - If a JVM runs a garbage collection cycle, what other threads content on kernel locks during that time?

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Instrumentation techniques

| | uninstrumented | instrumented |
|---|---|---|
| squeue_enter_chain+0x1af: | xorl %eax,%eax | xor %eax,%eax |
| squeue_enter_chain+0x1b1: | nop | nop |
| squeue_enter_chain+0x1b2: | nop | nop |
| squeue_enter_chain+0x1b3: | nop | lock nop (later replaced by a call) |
| squeue_enter_chain+0x1b4: | nop | |
| squeue_enter_chain+0x1b5: | nop | nop |
| squeue_enter_chain+0x1b6: | movb %bl,%bh | movb %bl,%bh |
| ufs_mount: | pushq %rbp | int $0x3 |
| ufs_mount+1: | movq %rsp,%rbp | movq %rsp,%rbp |
| ufs_mount+4: | subq $0x88,%rsp | subq $0x88,%rsp |
| ufs_mount+0xb: | pushq %rbx | pushq %rbx |
| ...... | | |
| ufs_mount+0x3f3: | popq %rbx | popq %rbx |
| ufs_mount+0x3f4: | movq %rbp,%rsp | movq %rbp,%rsp |
| ufs_mount+0x3f7: | popq %rbp | popq %rbp |
| ufs_mount+0x3f8: | ret | int $0x3 |

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Dtrace and mdb

- **Accessing Dtrace data from a crash dump**
  - Analyzing Dtrace state
    - Trace buffers, consumers, etc

```
> ::dtrace_state
    ADDR MINOR     PROC NAME                    FILE
ccaba400     2        - <anonymous>                -
ccab9d80     3 d1d6d7e0 intrstat            cda37078
cbfb56c0     4 d71377f0 dtrace              ceb51bd0
ccabb100     5 d713b0c0 lockstat            ceb51b60
d7ac97c0     6 d713b7e8 dtrace              ceb51ab8
```

redhat SUSE ORACLE Department of Distributed and Dependable Systems D3S

# Dtrace and mdb (2)

- Displaying the contents of a trace buffer

```
> ccaba400::dtrace
CPU     ID                    FUNCTION:NAME
  0    344           resolvepath:entry   init
  0     16                 close:entry   init
  0    202                 xstat:entry   init
  0    202                 xstat:entry   init
  0     14                  open:entry   init
  0    206                fxstat:entry   init
  0    186                  mmap:entry   init
  0    186                  mmap:entry   init
  0    186                  mmap:entry   init
  0    190                munmap:entry   init
  0    344           resolvepath:entry   init
  0    216               memcntl:entry   init
  0     16                 close:entry   init
  0    202                 xstat:entry   init
...
```

# Dtrace and mdb (3)

- **Interpreting the results**
  - The output of `::dtrace` has the same format as the output of the `dtrace` utility (the default consumer)
    - The order of events is always oldest to newest within each CPU
    - The CPU buffers are displayed in numerical order
      - Use `::dtrace -c` to show only a specific CPU
    - Only in-kernel data yet unprocessed by the user space consumer can be displayed
      - The default consumer can be forced to keep as much data as possible in the kernel buffer

```
dtrace -s … -b 64k -x bufpolicy=ring
```

# References

- **Oracle Solaris Dynamic Tracing Guide**
  - http://docs.oracle.com/cd/E23824_01/html/E22973/toc.html
- **illumos Dynamic Tracing Guide**
  - http://dtrace.org/guide/