

# Kernels and Tracing

Lecture 2, Part 1: DTrace

Prof. Robert N. M. Watson

2022-2023

# Kernels and tracing

- DTrace

} Lecture 2, Part 1

- The **probe effect**

} Lecture 2, Part 2

- Our lab environment

} Lecture 2, Part 3

# Dynamic tracing with DTrace


- Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Leventhal. *Dynamic Instrumentation of Production Systems*, USENIX ATC 2004.
  - “Facility for dynamic instrumentation of production systems”
  - Unified and safe **instrumentation** of kernel and user space
  - Zero **probe effect** when not enabled
  - Dozens of **providers** representing different trace mechanisms
  - Tens (hundreds?) of thousands of **instrumentation probes**
  - **D language**: C-like scripting language with **predicates**, **actions**
  - Scalar variables, thread-local variables, associative arrays
  - **Data aggregation** and **speculative tracing**
- First-class feature in: Solaris, Mac OS X, FreeBSD, Windows; third-party Linux module
- Wide influence – e.g., on Linux SystemTap, eBPF
- **Our tool of choice in this course**

# DTrace scripts

- Human-facing, C-like **D Programming Language**
- One or more {**probe name**, **predicate**, **action**} tuples
- Expression limited to control side effects (e.g., no loops)
- Specified on command line or via a `.d` file

<b>Probe name</b>	Identifies the probe(s) to instrument; wildcards allowed; identifies the provider and provider-specific probe name
<b>Predicate</b>	Filters cases where action will execute
<b>Action</b>	Describes tracing operations

```
fbt::malloc:entry /execname == "csh"/ { trace(arg0); }
```



# Some FreeBSD DTrace providers

- Providers represent data sources – instrumentation types:

Provider	Description
callout_execute	Timer-driven “callout” event probes
dtmalloc	Kernel malloc()/free()
dtrace	DTrace script events (BEGIN, END)
fbt	Function Boundary Tracing (function prologues, epilogues)
io	Block I/O read/write
ip,udp,tcp,sctp	TCP/IP events
lockstat	Kernel locking primitives
proc,sched	Kernel process, scheduling primitives
profile	Profiling timers
syscall	System-call entry/return
vfs	Virtual File System operations

- Apparent duplication: FBT vs. event-class providers?
  - Efficiency, expressivity, interface stability, portability

# Tracing kernel malloc() calls

- Trace first argument to kernel malloc() for csh
- NB: Captures both successful and failed allocations

```
# dtrace -n  
  'fbt::malloc:entry /execname=="csh"/ { trace(arg0); }'
```

<b>Probe</b>	Use FBT to instrument malloc() function prologue
<b>Predicate</b>	Limit actions to processes executing csh
<b>Action</b>	Trace the first argument (arg0)

<b>CPU</b>	<b>ID</b>	<b>FUNCTION:NAME</b>	
0	8408	malloc:entry	64
0	8408	malloc:entry	2748
0	8408	malloc:entry	48
0	8408	malloc:entry	392

^C

# Aggregations – summarising traces

- **Aggregations** allow early, efficient reduction
  - Scalable multicore implementations (i.e., commutative)

```
@variable = function(.. args ..);  
printa(@variable)
```

Aggregation	Description
count()	Number of times called
sum()	Sum of arguments
avg()	Average of arguments
min()	Minimum of arguments
max()	Maximum of arguments
stddev()	Standard deviation of arguments
lquantize()	Linear frequency distribution (histogram)
quantize()	Log frequency distribution (histogram)

# Profiling kernel malloc() calls by csh

```
fbt::malloc:entry  
/execname=="csh"/  
{ @traces[stack()] = count(); }
```

<b>Probe</b>	Use FBT to instrument malloc() function prologue
<b>Predicate</b>	Limit actions to processes executing csh
<b>Action</b>	Keys of associative array are stack traces (stack()); values are aggregated counters (count())

```
^C  
    kernel`malloc  
    kernel`fork1+0x14b4  
    kernel`sys_vfork+0x2c  
    kernel`swi_handler+0x6a8  
    kernel`swi_exit  
    kernel`swi_exit  
      3  
...
```



# D Intermediate Format (DIF)

```
# dtrace -Sn
```

```
'fbt::malloc:entry /execname == "csh"/ { trace(arg0); }'
```

Predicate

```
DIF0 0x0x8047d2320 returns D type (integer) (size 4)
```

OFF	OPCODE	INSTRUCTION	
00:	29011801	ldgs DT_VAR(280), %r1	! DT_VAR(280) = "execname"
01:	26000102	sets DT_STRING[1], %r2	! "csh"
02:	27010200	scmp %r1, %r2	
03:	12000006	be 6	
04:	0e000001	mov %r0, %r1	
05:	11000007	ba 7	
06:	25000001	setx DT_INTEGER[0], %r1	! 0x1
07:	23000001	ret %r1	

NAME	ID	KND	SCP	FLAG	TYPE
execname	118	scl	glb	r	string (unknown) by ref (size 256)

Action

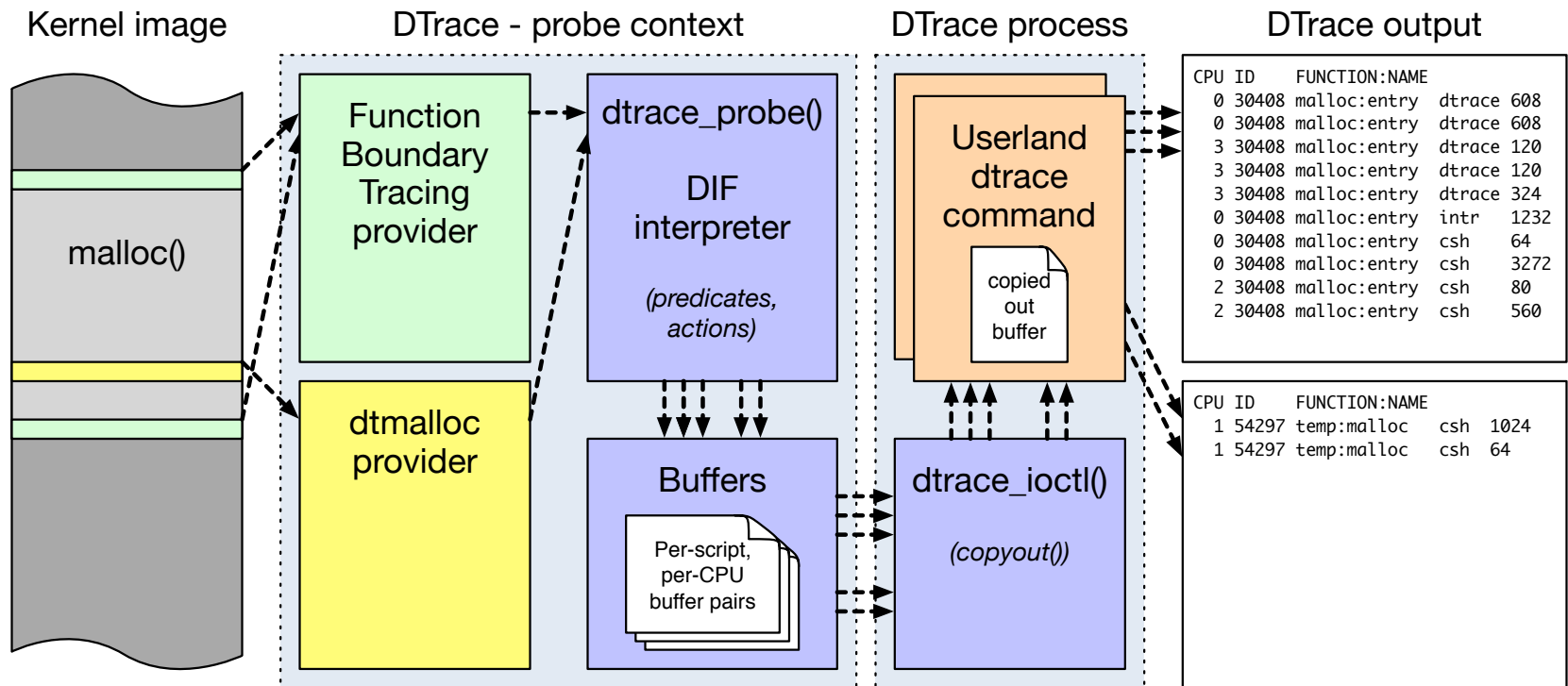
```
DIF0 0x0x8047d2390 returns D type (integer) (size 8)
```

OFF	OPCODE	INSTRUCTION	
00:	29010601	ldgs DT_VAR(262), %r1	! DT_VAR(262) = "arg0"
01:	23000001	ret %r1	

NAME	ID	KND	SCP	FLAG	TYPE
arg0	106	scl	glb	r	D type (integer) (size 8)

# DTrace: Implementation

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



```
dtrace -n 'dtmalloc::temp:malloc /execname="csch"/ { trace(execname); trace(arg3); }'
```