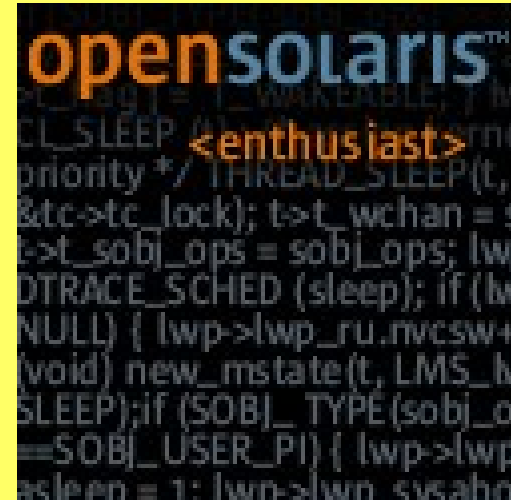


DTrace – new way of thinking

Robert Milkowski
System Group Manager
Wirtualna Polska



What is DTrace?

- Dynamic instrumentation
- No need for special kernel
- No need for recompiling applications
- 0% probe effect when not used
- The same framework for kernel and user space tracing
- No silent data lost
- Scalability
- Ready for production use
- D language

What is Dtrace?, cont.

- **DTrace is Open Source!**
 - It's part of OpenSolaris.org
 - It's been publicly available for about 2 years
 - DTrace is under CDDL license
 - Unfortunately it's not compatible with GPL
 - However it's compatible with many other free licenses
- For example look at the porting effort of DTrace to FreeBSD

D language

- C/AWK like language – very easy to learn
- Actions based
- Predicates
- Global, thread local and probe local variables
- Data aggregation
- Associative arrays
- Speculations
- Both command line and scripts supported
- Many DTrace consumers at the same time
- Access to all global kernel variables and structures
- Built-in variables (pid, ppid, tid, execname, ...)

Probes

- Probe defines point of instrumentation
- Some probes are dynamically created (PID Provider)
- Depending on type probe has *zero* or *near zero* effect
- Probe is described by: Provider, Module, Function, Name

Example probe: **io:nfs:nfs_bio:start**

You can get list of all actual probes by issuing: `dtrace -l`

```
# dtrace -l | wc -l
#      38881
```

Providers

- Provider registers probes in system
- Some providers hide implementation details (IO, mib)
- There are more than 10 providers in a system by default

My favorite providers:

- **syscall** provider – traces syscalls
- **PID** provider – traces all application functions
- **IO** provider – traces I/O in a system in an abstract way
- **FBT** provider – traces all in-kernel functions
- **SCHED** provider – abstracts scheduler in a system

D language structure

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

- Many probes could be specified
- Only if predicate is true actions are executed
- Predicate line could be skipped
- Actions could be skipped – default action is taken
- Probe does not need full description – omitted fields are treated like *

D language structure - example

```
syscall::open:entry
/ pid == 123 /
{
    trace(execname);
}
```

DTrace safety

Design principles

- safe use on production systems
- provide only safe probes
- no recursion for probes
- in-kernel virtual machine
- prevent stores to arbitrary kernel memory
- no loops in kernel
- do not allow run illegal instructions while in kernel
- no divide by zero
- protect I/O space memory
- prevent loads from unmapped memory

More info on safety in DTrace - http://blogs.sun.com/roller/page/bmc?entry=dtrace_safety

DTrace safety, cont.

- Privileges used to specify which user can use Dtrace
- At least one `dtrace_*` privilege is needed to use DTrace

```
bash-3.00# ppriv -lv dtrace_proc,dtrace_user,dtrace_kernel
```

dtrace_kernel

Allows DTrace kernel-level tracing.

dtrace_proc

Allows DTrace process-level tracing.

Allows process-level tracing probes to be placed and enabled in processes to which the user has permissions.

dtrace_user

Allows DTrace user-level tracing.

Allows use of the `syscall` and `profile` DTrace providers to examine processes to which the user has permissions.

Destructive Actions

- Need to be explicitly enabled
- Need all privileges

- *stop()* - stop process which fired the probe
- *raise()* - send a signal to a process
- *copyout()* - copy buffer to user space address
- *copyoutstr()* - copy string to user space address
- *system()* - run external program
- *breakpoint()* - forces kernel to stop and go into debugger
- *panic()* - forces kernel to panic and dump crash dump
- *chill()* - spin in a probe for specified number of ns

syscall Provider

- Can trace all syscall entry and return points
- Access to syscall arguments
- Access to returned value
- Access to errno

syscall Provider - example

```
bash-3.00# dtrace -n syscall::: '{@prog[probefunc]=count();}'  
dtrace: description 'syscall:::' matched 452 probes  
^C
```

sysconfig	6
brk	12
gtime	16
setcontext	18
lwp_sigmask	26
setitimer	26
p_online	42
write	96
writev	132
pollsys	320
read	350
ioctl	426

```
bash-3.00#
```

syscall Provider – example, cont.

```
bash-3.00# dtrace -n syscall::read: '{@prog[execname]=count();}'  
dtrace: description 'syscall::read:' matched 2 probes  
^C
```

notification-are	2
gnome-smproxy	4
xscreensaver	4
ggv-postscript-v	6
gnome-panel	6
gnome-settings-d	6
metacity	6
nautilus	6
wnck-applet	6
dsdm	8
ggv	12
soffice.bin	18
gnome-terminal	148
Xorg	540

```
bash-3.00#
```

syscall Provider – example, cont.

```
bash-3.00# dtrace -n syscall::read:entry '/execname == "Xorg" / \  
                                     {@prog[fds[arg0].fi_pathname]=count();}'  
dtrace: description 'syscall::read:entry' matched 1 probe  
^C
```

/devices/pseudo/consms@0:mouse	138
/devices/pseudo/conskbd@0:kbd	139
<unknown>	1560

```
bash-3.00#
```

```
bash-3.00# dtrace -n syscall::read:entry '/execname == "Xorg" / {@stack[ustack()]=count();}'  
dtrace: description 'syscall::read:entry' matched 1 probe  
^C
```

```
libc.so.1`_read+0x15  
Xorg`xf86ReadSerial+0x1a  
Xorg`XisbRead+0x45  
Xorg`vuidReadInput+0x82  
Xorg`xf86Wakeup+0xa8  
Xorg`WakeupHandler+0x40  
Xorg`WaitForSomething+0x1f7  
Xorg`Dispatch+0x95  
Xorg`main+0x4d1  
Xorg`0x807b10a  
112
```

syscall Provider – example, cont.

```
libc.so.1`_read+0x15
Xorg`xf86KbdEvents+0x24
Xorg`xf86Wakeup+0x69
Xorg`WakeupHandler+0x40
Xorg`WaitForSomething+0x1f7
Xorg`Dispatch+0x95
Xorg`main+0x4d1
Xorg`0x807b10a
114

libc.so.1`_read+0x15
Xorg`_XSERVTransSocketRead+0x19
Xorg`_XSERVTransRead+0x17
Xorg`StandardReadRequestFromClient+0x2d3
Xorg`Dispatch+0x27f
Xorg`main+0x4d1
Xorg`0x807b10a
328
```

bash-3.00#

syscall Provider – example, cont.

```
bash-3.00# dtrace -n syscall::read:entry'/execname == "Xorg"/ \
                {@read["read size distribution"]=quantize(arg2);}'
dtrace: description 'syscall::read:entry' matched 1 probe
^C
```

read size distribution

value	----- Distribution -----	count
32		0
64	@@@	131
128		0
256		0
512		0
1024	@@@	133
2048		1
4096	@@@@	1484
8192		0

```
bash-3.00#
```

PID Provider

- Probes for every function entry and return
- Probes for every instruction
- Real 0% effect on non-probed processes
- Doesn't stop traced process – no serialization
- No need for any application changes or recompiling

PID Provider internals - <http://blogs.sun.com/roller/resources/ahl/img0.html>

PID Provider - example

```
bash-3.00# dtrace -F -n pid`pgrep sshd`:a.out::entry, \  
                pid`pgrep sshd`:a.out::return'{'}'  
dtrace: description 'pid483:a.out::entry,pid483:a.out::return'  
matched 1821 probes  
CPU FUNCTION  
0  -> drop_connection  
0  <- drop_connection  
0  -> contracts_pre_fork  
0    -> debug3  
0      -> do_log  
0      <- do_log  
0    <- debug3  
0  <- contracts_pre_fork  
0  -> contracts_post_fork_parent  
0    -> debug3  
0      -> do_log  
0      <- do_log  
0    <- debug3  
0    -> debug3  
0      -> do_log  
0      <- do_log  
0    <- debug3
```

PID Provider – example, cont.

```
bash-3.00# dtrace -q -n pid`pgrep sshd`:a.out:do_log:entry \
                '{trace(copyinstr(arg1));ustack();}'
```

Set active contract

```
sshd`do_log
sshd`debug3+0x14
sshd`contracts_pre_fork+0x40
sshd`main+0xb56
sshd`0x805bada
```

Cleared active contract template (parent)

```
sshd`do_log
sshd`debug3+0x14
sshd`contracts_post_fork_parent+0x40
sshd`main+0xb76
sshd`0x805bada
```

Abandoned latest contract

```
sshd`do_log
sshd`debug3+0x14
sshd`contracts_post_fork_parent+0x103
sshd`main+0xb76
sshd`0x805bada
```

PID Provider – example, cont.

```
bash-3.00# cat vtime-1.d
#!/usr/sbin/dtrace -qs
```

```
pid$1:a.out::entry
{
    self->vt = vtimestamp;
}
```

```
pid$1:a.out::return
/ self->vt /
{
    @vt[probfunc] = sum (vtimestamp - self->vt);
    self->vt = 0;
}
```

```
bash-3.00#
```

PID Provider – example, cont.

```
bash-3.00# ./vtime-1.d `pgrep sshd`
```

```
^C
```

```
drop_connection          4655
xmalloc                  5304
xfree                     9683
do_log                   18740
arc4random_stir          216773
bash-3.00#
```

USDT Provider

- Developer can put probes in an application
- Hide internal application architecture
- Near 0% effect when probes not active

- Put a macro `DTRACE_PROBE*`()
- Put a provider description in a `.d` file

```
provider test {
    probe session-start(id_t id);
    probe session-finish(id_t id);
};
```
- Run `dtrace -G`
- Link resulting object file into the binary

SCHED Provider

- Probes for scheduler
- No scheduler implementation details needed
- Easy to see what application is consuming CPU
- Why and when thread is off-cpu?
- Easy aggregate for fork-model applications
- vmstat running queue – quick answer

SCHED Provider - example

```
bash-3.00# cat on-cpu.d
#!/usr/sbin/dtrace -qs

sched:::on-cpu
{
    self->vt = vtimestamp;
}

sched:::off-cpu
/ self->vt /
{
    @vtime[execname] = sum(vtimestamp - self->vt);
    self->vt = 0;
}
```

SCHED Provider – example, cont.

```
bash-3.00# ./on-cpu.d  
^C
```

mapping-daemon	74640
ssh-agent	81999
gnome-volcheck	82381
notification-area	86475
nautilus-throbbe	87437
clock-applet	91921
dsdm	93948
gnome-session	99244
ggv-postscript-v	134397
sendmail	135798
xscreensaver	144397
gnome-smproxy	166812
gnome-settings-d	188228
xntpd	259934
gnome-vfs-daemon	289397
gnome-panel	364158
ggv	372447
wnck-applet	416279
metacity	666730
fsflush	1463338
dtrace	2716549
nautilus	2951075
soffice.bin	8612605
Xorg	37180704
gnome-terminal	46984438
sched	3236322785

```
bash-3.00#
```

Why new way of thinking?

- All questions have answers
- No need to recompile or stop application
- Analyzing problems backward – from cause to effect
- Why we've got zombie process?
- Which application is eating CPU?
- Which program is paging-in?
- ...

Sys admins and developers can see system as a whole - from the kernel thru libraries to a user program, all in a safe and consistent way.

FreeBSDtrace

Project started at August 2005

On 9th October “[...] *and have a working dtrace binary!*”

Developers:

- Devon H. O'Dell
- Pascal G. Hofstee

<http://web.pulltheplug.org/dtrace/>

Java & DTrace

- DTrace itself has jstack()
- Java Provider - you can trace all the way from Java thru system libraries to kernel level

Not ready for production use

But still could be useful for development

<https://solaris10-dtrace-vm-agents.dev.java.net/>

More info and examples:

http://blogs.sun.com/roller/page/bmc?entry=your_java_fell_into_my

http://blogs.sun.com/roller/page/ahl/20050418#dtracing_java

PHP & DTrace

PHP core developer Wez Furlong with help from Bryan Cantrill

“Wez implemented it as a shared object, which may then be loaded via an explicit extension directive in php.ini. Once loaded, two probes show up: function-entry and function-return. These probes have as their arguments a pointer to the function name, a pointer to the file name, and a line number.”

Bryan Cantrill

<http://netevil.org./node.php?uuid=42f82a8b-09b7-5013-1667-2f82a8b24ead>
http://blogs.sun.com/roller/comments/bmc/Weblog/dtrace_and_php

PHP & Dtrace, cont.

- Allows to easily trace multiple instances of PHP
- Allows to correlate PHP with system behavior
- Two probes available: function-entry, function-return
- Each probe has three arguments:
 - Name of PHP function
 - Name of the file
 - Line number probe was called-in

Other languages & DTrace

Perl

http://blogs.sun.com/roller/page/alanbur?entry=dtrace_and_perl

Ruby

http://blogs.sun.com/roller/page/bmc?entry=dtrace_and_ruby

DTrace based programs

DTrace toolkit - <http://www.opensolaris.org/os/community/dtrace/dtracetoolkit/>

DTrace scripts - <http://www.opensolaris.org/os/community/dtrace/scripts/>

DTrace scripts - <http://users.tpg.com.au/adsl4yb/dtrace.html>

plockstat(1M)

DTrace documentation

- Check DTrace mailing list at OpenSolaris.org
- **Participate in DTrace development**

Solaris Dynamic Tracing Guide - <http://docs.sun.com/app/docs/doc/817-6223>

BigAdmin DTrace - <http://www.sun.com/bigadmin/content/dtrace/>

Open Solaris DTrace Community - <http://www.opensolaris.org/os/community/dtrace/>

DTrace sources on Open Solaris - <http://cvs.opensolaris.org/source/>

Bryan Cantrill's Blog

<http://blogs.sun.com/roller/page/bmc>

Adam Leventhal's Blog

<http://blogs.sun.com/roller/page/ahl>

Mike Shapiro's Blog

<http://blogs.sun.com/mws>

Q & A

DTrace – new way of thinking

Robert Milkowski
System Group Manager
Wirtualna Polska

rmilkowski@wp-sa.pl
<http://milek.blogspot.com>

