OpenSolaris: Dynamic Tracing: DTrace
(abbreviated from Jim Mauro's Usenix preso)

Harry J Foxwell, PhD
Principal Consultant
Oracle Corporation

harry.foxwell@oracle.com
Acknowledgements

Some of this material represents an aggregation and consolidation of existing material, including the original *Usenix* paper*. A significant amount of the material contained in these slides was inserted from the slide sets, blogs, emails, letters, post cards, faxes, telegrams and sticky notes of others:

- Stefan Parvu
- Brendan Gregg
- Bryan Cantrill
- Mike Shapiro
- Adam Leventhal
- Jon Haslam
- Bart Smaalders
- Jarod Jenson
- Chad Mynhier
- Jim Fiori
- Jonathan Adams
- John Birrell
- Simon Ritter
- Angelo Rajadurai
- Bob Netherton
- Peter Karlsson
- Roch Bourbonnais
- Richard McDougall
- Keith McGuigan
- John Levon
- Chip Bennett
- **Jim Mauro**

*http://www.sun.com/bigadmin/content/dtrace/dtrace_usenix.pdf*
Resources

- DTrace documentation
  http://wikis.sun.com/display/DTrace/Documentation
- DTrace tutorials, scripts, etc
- DTrace community (lots 'o stuff)
  http://www.opensolaris.org/os/community/dtrace/
- DTrace ToolKit
  http://www.brendangregg.com/dtrace.html#DTraceToolkit
- Blogs, blogs, blogs
  http://blogs.sun.com
Scripts!

- Sample commands and scripts

/usr/demo/dtrace/* .d (on Solaris 10 and OpenSolaris)

http://www.opensolaris.org/os/community/dtrace/dtracetoollkit/
What is DTrace?

- DTrace is a facility for the **dynamic** instrumentation of **production** systems, for the purpose of troubleshooting and analysis.
  - First introduced in Solaris 10 (3/05)
  - Ported to **Mac OS X** and FreeBSD
  - but **not** Linux (See **SystemTap** instead)
- DTrace is many things, in particular:
  - An **instrumentation** framework
  - A **programming language**
- DTrace provides **observability** across the entire software stack from **one tool**. This allows you to examine software execution like never before.
  - Instrument kernel and user software in a unified fashion
System Analysis

- Traditional development and debugging tools are tightly bound to the language and/or development framework
  - SunStudio, IDE tools, etc
  - Lack **system view** process only or kernel only)
- System tools lack correlation to the workload
  - sar, mpstat, vmstat, iostat, etc
  - You can see what the system is doing, but...
- Hard to debug *transient* problems with truss(1), pstack(1), prstat(1M), etc
- Only mdb(1) designed for systemic problems, but intended for postmortem analysis
  - mdb(1) is useful for some live system views
DTrace

- A powerful framework for real-time analysis and observability. System **and** process centric
- Dynamic instrumentation of the kernel **and** applications
- Dynamically interpreted language allows for arbitrary actions and predicates in multiple points of instrumentation
- Designed for live production systems:
  - a **totally safe** way to inspect **live data** on production systems
DTrace's system-wide view allows you to “connect the dots!”", correlating system activity to the workload.

"what's the system doing?"

"what are the processes doing?"
DTrace
An Observability Revolution

- Ease-of-use and instant gratification engenders serious hypothesis testing
- Instrumentation directed by high-level control language (not unlike AWK or C) for easy scripting and command line use
  - Build your DTrace toolbox
- Comprehensive probe coverage and powerful data management allow for concise answers to arbitrary questions
  - What are these system calls, and who's executing them?
DTrace

• Safe and comprehensive: tens-of-thousands of data monitoring points (dtrace -l)
  – Inspect kernel and user space
• Reduced costs: problems usually found in minutes or hours, not days or weeks
• Flexibility: DTrace lets you create your own custom programs to dynamically instrument the system
• No need to instrument your applications via source code modifications; no need to stop or restart them
The Entire Software Stack

• How did you analyze these?

Examples:

- Dynamic Languages: Java, JavaScript, ...
- User Executable: compiled code, /usr/bin/*
- Libraries: /usr/lib/*
- Syscall Interface: man -s2
- Kernel: VFS, DNLC, UFS, ZFS, TCP, IP, ...
- Memory allocation: sd, st, hme, eri, ...
- Device Drivers: NIC, disk data controller, CPU
- Scheduler: CPU
The Entire Software Stack

• It was possible, but difficult.

Previously:
- debuggers
- truss -ua.out
- apptrace, sotruss
- truss
- prex; tnf*
- lockstat
- mdb
- kstat, PICs, guesswork
The Entire Software Stack

- DTrace is all seeing:
  
  - Dynamic Languages: Yes, with providers
  - User Executable: Yes
  - Libraries: Yes
  - Syscall Interface: Yes
  - Kernel: Yes
  - File Systems: Yes
  - Memory allocation: Yes (to some extent, very recent)
  - Device Drivers: Yes
  - Scheduler: Yes
  - Hardware: Yes
Syscall Example

• Using truss,

```
$ truss date
execve("/usr/bin/date", 0x08047C9C, 0x08047CA4) argc = 1
resolvepath("/usr/lib/ld.so.1", "/lib/ld.so.1", 1023) = 12
resolvepath("/usr/bin/date", "/usr/bin/date", 1023) = 13
xstat(2, "/usr/bin/date", 0x08047A58) = 0
open("/var/ld/ld.config", O_RDONLY) = 3
fxstat(2, 3, 0x08047988) = 0
mmap(0x00000000, 152, PROT_READ, MAP_SHARED, 3, 0) = 0xFEFB0000
close(3) = 0
mmap(0x00000000, 4096, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, -1
sysconfig(_CONFIG_PAGESIZE) = 4096
[...]
```

truss slows down the target (probe effect)
Syscall Example

• Using DTrace

You can select which syscall(s)

You choose the output

```
# dtrace -n 'syscall:::entry { printf("%16s %x %x", execname, arg0, arg1); }'
dtrace: description 'syscall:::entry ' matched 233 probes

<table>
<thead>
<tr>
<th>CPU</th>
<th>ID</th>
<th>FUNCTION:NAME</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75943</td>
<td>read:entry</td>
<td>Xorg</td>
<td>8047130</td>
</tr>
<tr>
<td>1</td>
<td>76211</td>
<td>setitimer:entry</td>
<td>Xorg</td>
<td>8047610</td>
</tr>
<tr>
<td>1</td>
<td>76143</td>
<td>writev:entry</td>
<td>Xorg 22</td>
<td>80477f8</td>
</tr>
<tr>
<td>1</td>
<td>76255</td>
<td>pollsys:entry</td>
<td>Xorg 8046da0 1a</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>75943</td>
<td>read:entry</td>
<td>Xorg 22</td>
<td>85121b0</td>
</tr>
<tr>
<td>1</td>
<td>76035</td>
<td>ioctl:entry</td>
<td>soffice.bin 6 5301</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>76035</td>
<td>ioctl:entry</td>
<td>soffice.bin 6 5301</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>76255</td>
<td>pollsys:entry</td>
<td>soffice.bin 8047530 2</td>
<td></td>
</tr>
</tbody>
</table>

[...]```
DTrace Features*

- Dynamic Instrumentation
- Unified Instrumentation
- Arbitrary-context kernel instrumentation
- Data integrity
- Arbitrary actions
- Predicates
- High-level control language
- Scalable data aggregation
- Speculative tracing
- Scalable architecture
- Virtualized consumers

*http://www.sun.com/bigadmin/content/dtrace/dtrace_usenix.pdf
What is DTrace For?

• Troubleshooting performance problems
  – Profile applications and the kernel
  – Latency measurements
  – Looking for areas for improvement even when performance is acceptable

• Troubleshooting software bugs
  – Proving what the problem is, and isn't.
  – Measuring the magnitude of the problem.

• Detailed observability
  – Observing the kernel
  – Observing devices, such as disk or network activity.
  – Observing applications, whether they are from Sun, 3rd party, or in-house.
A Few Words on Operating System Support of DTrace...
DTrace in Solaris/OpenSolaris

• Check out Bryan's blog on DTrace's 5th birthday for some cool history and trivia
  http://blogs.sun.com/bmc/

• DTrace was integrated into Solaris 10, and available with Solaris 10 3/05

• Additional features added in subsequent releases
  – OpenSolaris on the leading edge

• Use the Wiki site for most recent documentation
  http://wikis.sun.com/display/DTrace/Documentation

• Solaris Process Privileges enable non-root users to use DTrace (e.g. in zones!)
  dtrace_user, dtrace_proc, dtrace_kernel
DTrace in Mac OS X

- Added to Leopard (10.5)
- Not all providers implemented
  - e.g. sched not there...
  - some are intentionally omitted (DRM issue!)
- Instruments is built on DTrace
- pid provider, and plockstat are implemented!

macosx> plockstat -A -p 37476
^C
Mutex hold

<table>
<thead>
<tr>
<th>Count</th>
<th>nsec</th>
<th>Lock</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>1057</td>
<td>55473</td>
<td>0x16886fc4</td>
<td>0x1fec60</td>
</tr>
<tr>
<td>62</td>
<td>490985</td>
<td>0x605845c</td>
<td>0x1fec60</td>
</tr>
<tr>
<td>741</td>
<td>20183</td>
<td>0x58395cc</td>
<td>0x1fec60</td>
</tr>
<tr>
<td>72</td>
<td>194605</td>
<td>0x58395cc</td>
<td>0x1fec60</td>
</tr>
<tr>
<td>5</td>
<td>558552</td>
<td>0x605845c</td>
<td>0x1fec60</td>
</tr>
<tr>
<td>50</td>
<td>52090</td>
<td>0x16886fc4</td>
<td>0x1fec60</td>
</tr>
</tbody>
</table>
DTrace in FreeBSD

- DTrace is available in FreeBSD, beginning with the 7.1 beta bits
  - Currently downloadable from the FreeBSD site

- After installing, you need to do a kernel build, reboot, and load the dtrace module
  
  
  - Don't forget “kldload dtraceall” after you reboot! (I did, even though it's in the documentation!)

- Several providers not yet implemented

- Thus far, I have limited experience with the FreeBSD DTrace functionality

```bash
freebsd# uname -a
FreeBSD freebsd.localdomain 7.1-BETA FreeBSD 7.1-BETA #0: <...> i386
freebsd# dtrace -l | wc -l
 33198
freebsd#
```
DTrace Components

• Probes
  - A point of instrumentation and data generation

• Providers
  - A major component of DTrace, Providers manage probes of specific types, and for a specific area of the system
    - syscall, io, sched, proc, vminfo, etc

• Consumers
  - Users of the framework

    dtrace(1), lockstat(1), plockstat(1), intrstat(1)
DTrace User Components

• Predicates
  – User-defined conditional statements evaluated when probes fire
  – Provides a control flow mechanism for your D programs – data pruning at the source

• Actions
  – What to do when the probe fires
    Data to gather
    Timestamps for profiling
    Many other actions supported
DTrace – What Happens

- `dtrace` command compiles the D language Script.
- Intermediate code checked for safety (like java).
- The compiled code is executed in the kernel by DTrace.
- DTrace instructs the provider to enable the probes.
- As soon as the D program exits all instrumentation removed.
- No limit (except system resources) on number of D scripts that can be run simultaneously.
- Different users can debug the system simultaneously without causing data corruption or collision issues.
DTrace – The Big Picture

- **script.d**
- **dtrace(1M)**
- **lockstat(1M)**
- **plockstat(1M)**
- **libdtrace(3LIB)**
- **dtrace(7D)**

**DTrace**

**dtrace providers**
- sysinfo
- vminfo
- fasttrap
- proc
- syscall
- sdt
- io
- sched

**dtrace consumers**
- userland
- kernel
Inside interpreter: in the kernel space that interprets instructions and verifies that each pointer is safe to access or read

**Protection against memory violations** – accessing a userland memory address results in a disabled probe

**No loops**, avoids the *Halting Problem*

- “Given a description of a program and its initial input, determine whether the program, when executed on this input, ever halts (completes). The alternative is that it runs forever without halting. We say that the halting problem is undecidable over Turing machines.”

DTrace Safety – A Bit More...

• “...the most fundamental is the principle of safety: DTrace must not be able to accidentally induce system failure.” *

• Probes are provided by instrumentation providers that guarantee their safety *
  - Users are not permitted to arbitrarily select instrumentation points

• While in probe context, DTrace itself must not call into any facilities in the kernel-at-large *
  - Probe context – protection against recursion

• Safe execution of user probe actions and predicates
  - Non-native execution: runs in a virtual machine
  - DTrace D programs compiled into a safe intermediate format for execution, and validated for safety

Running DTrace

- **Only root** allowed to run DTrace by default
  - Solaris, OS X and FreeBSD

- In Solaris, process privileges can grant dtrace permission to non-root users;

  ```
  $ ppriv -l | grep dtrace
  dtrace_kernel  Allow DTrace kernel-level tracing
  dtrace_proc    Allow DTrace process-level tracing. Allow process-level tracing probes to be placed and enabled in processes to which the user has perms
  dtrace_user    Allow DTrace user-level tracing. Allow use of the syscall and profile DTrace providers to examine processes for which the user has permissions
  ```

  - Enable using usermod utility

  ```
  # usermod -K defaultpriv=basic,dtrace_kernel,\ dtrace_proc,dtrace_user username
  ```
DTrace Framework

- Probes and Providers
- Actions and Predicates
- The D language
- Aggregations
- Pointers and Arrays
- Strings
- Structs and Unions
- Output formatting
- Speculative tracing
DTrace Probes

• A point of instrumentation, made available by a provider, which has a unique name
• A four-tuple name uniquely identifies every probe; 
  \[ \text{provider:module:function:name} \]
  - \textit{provider}: the DTrace provider that manages the probe (DTrace kernel module)
  - \textit{module}: kernel module or user library where the probe is located
  - \textit{function}: kernel or user function containing the probe
  - \textit{name}: represents an entry point in that function (e.g. \textit{entry} or \textit{return}), or has a meaningful name (e.g. \textit{io:::start}, \textit{proc:::exec})
  - missing component means wildcard
Probes

• Anchored Probes
  – Instrument a specific point in code, e.g.
    fbt:ufs:ufs_read:entry
    io:::start
    ip:::receive

• Unanchored Probes
  – Are not associated with a specific location in code
  – Do not have a module or function component to their name
  – profile and tick
    profile-997hz, tick-10sec
Probes

• List probes
  - Use dtrace(1M) with the '-l' option
  - For each probe the four-tuple name will be displayed, probe components are ':' separated
  - List all probes:
    $ dtrace -l

  - List all probes offered by syscall provider:
    $ dtrace -lP syscall

  - List all probes offered by the ufs module:
    $ dtrace -lm ufs

  - List all providers:
    $ dtrace -l | awk '{print $2}' | sort -u
Probes

- List all read function probes:
  
  $ dtrace -l -f read

• Enabling probes
  
  - Activate a probe by not using '-l' option
  - Default action with enabled probes- the CPU, the probe number and name are displayed whenever the probe fires
  
  - Enable all probes from nfs and ufs module:
    
    $ dtrace -m nfs,ufs
  
  - Enable all read function probes:
    
    $ dtrace -f read
  
  - Enable all probes from io provider:
    
    $ dtrace -P io
Probes

- BEGIN and END
  - BEGIN: fires each time a trace request is made

```bash
# dtrace -n BEGIN
dtrace: description 'BEGIN' matched 1 probe
CPU     ID                    FUNCTION:NAME
  0      1                           :BEGIN
^C
```

- END: fires when the trace finishes

```bash
# dtrace -n END
dtrace: description 'END' matched 1 probe
^C
CPU     ID                    FUNCTION:NAME
  0      2                             :END
```
Probes

- **ERROR**
  - The ERROR probe fires when a runtime error is encountered

```
dhcp-s> dtrace -n 'dtrace:::BEGIN { myvar = *(char *)NULL; } dtrace:::ERROR { printf("OOooppssss....\n"); }'
dtrace: description 'dtrace:::BEGIN' matched 2 probes
CPU   ID                    FUNCTION:NAME
 1      3                           :ERROR OOooppssss....
dtrace: error on enabled probe ID 1 (ID 1: dtrace:::BEGIN): invalid address (0x0) in action #1 at DIF offset 16
^C
```

```
dhcp-s> dtrace -qn 'dtrace:::BEGIN { myvar = *(char *)NULL; } dtrace:::ERROR { printf("OOooppssss....\n"); }'
OOooppssss....
dtrace: error on enabled probe ID 1 (ID 1: dtrace:::BEGIN): invalid address (0x0) in action #1 at DIF offset 16
^C
```
Probes

• Examples
  - syscall:::
  - syscall:::entry
  - syscall:::return
  - syscall:::read::entry{ printf("Process %d", pid); } 
  - syscall:::write::entry/execname=="firefox-bin"/
    { @[probefunc] = count(); } 
  - sysinfo:::readch{ trace(execname); exit(0); } 
  - sysinfo:::writech
  - io:::
Probes and DTrace Built-in Variables

- Among the many built-in variables provided by DTrace, there are probe-specific variables available when a probe fires
  - Function call argument list. Arguments passed to a function instrumentable through an “entry” probe are available as either:
    - int64_t arg0, arg1, ...., arg9 – args available as raw 64 bit integers
    - args[0], args[1], ...., args[9] – typed args, corresponding to the specific data types of the arg list

- Probes in some providers have a specific arg list made available by the provider
  - e.g. the IO provider arg list of pointers to a buf structure, devinfo structure and fileinfo structure when IO provider probes fire

- You need to RTFM to determine what args are available for a given provider !!!
  - For function entry arg lists, you need man pages, kernel source, or just mdb(1) on a running Solaris system
Providers

- A methodology for instrumenting the system
- Providers offer all probes to the DTrace framework
- DTrace framework confirms to providers when a probe is activated
- Providers pass the control to DTrace when a probe is enabled
- Example of providers: syscall, lockstat, fbt, io, mib
Providers

- Providers do a couple interesting things for us...
  - Manage probes
  - Abstract a complex subsystem with intuitive probes, enabling and enhancing observability and analysis
    - sched:::oncpu
    - io:::start
    - etc...
  - You can use DTrace effectively to track application and kernel activity in areas of the kernel that you may not be familiar with
Provider Documentation

● Some providers assume a little background knowledge, other providers assume a lot. Knowing where to find supporting documentation is important.

● Where do you find documentation on -
  – Syscalls?
  – User Libraries?
  – Application Code?
  – Kernel functions?
**Provider Documentation**

- Additional documentation may be found here,

<table>
<thead>
<tr>
<th>Target</th>
<th>Provider</th>
<th>Additional Docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscalls</td>
<td>syscall</td>
<td>man(2)</td>
</tr>
<tr>
<td>libraries</td>
<td>pid:lib*</td>
<td>man(3C)</td>
</tr>
<tr>
<td>app code</td>
<td>pid:a.out</td>
<td>source code, ISV, developers</td>
</tr>
</tbody>
</table>
Providers

nv98> dtrace -l | awk '{ print $2}' | sort -u

PROVIDER
Xserver767
dtrace
fbt
fsinfo
io
ip
lockstat
lx-syscall
mib
proc
profile
sched
sdt
syscall
sysevent
sysinfo
vminfo

macosx> dtrace -l | awk '{ print $2}' | sort -u

PROVIDER
dslockstat87530
dtrace
fbt
io
lockstat
mach_trap
mds66
plockstat16190
plockstat16191
plockstat16192
proc
profile
syscall
vminfo
macosx>
Providers – dtrace

- dtrace
  - Aforementioned BEGIN, END, ERROR probes
  - Useful for printing headers (BEGIN), data summary (END), and gathering more information on errors (ERROR)
  - The ERROR probe provides args with additional information
    - arg1 – EPID of probe that caused the error
    - arg2 – Index of the action that caused the fault
    - arg3 – DIF action
    - arg4 – Fault type
    - arg5 – Value particular to fault type
Providers - syscall

- Manages probes where “applications meet the kernel”
- Two probes for each system call
  - entry
  - return
- Arguments
  - entry – arg0...argn – the arg list to the system call
  - return – arg0 and arg1 – return value
  - D variable errno provide system call failure info
- Note some system calls do not directly map to syscall probefuncs
  - System V IPC
Providers – profile & tick

• Time-based interrupt firing
  – profile – fires on all CPUs
  – tick – fires on only 1 CPU

• Specify interval in probe
  – hz, sec or s, min or m, msec or ms, usec or u, etc

• Two args
  – arg0 – PC if in the kernel (sys mode)
  – arg1 – PC if in user (usr mode)
  – Very handy for system-wide profiling...

profile-997hz / arg0 != 0 / { ... }   Am I in the kernel?
profile-997hz / arg1 != 0 / { ... }   Am I in user land?
Provider - tick take 2

● tick-nnn also handy for
  – Building scripts that provide output at intervals (like the *stat commands)
    
    ```
    tick-1sec { print(data); clear(data); }
    ```

  – Bail-out mechanism
    
    ```
    tick-500ms { print(data); exit(0); }
    ```
Providers - sdt

• Statically Defined Tracing
  – Probes inserted at points of interest in the kernel
  – Allows the programmer to add probes to code with meaningful names without creating a new provider, using DTrace macros (sys/sdt.h);

    DTRACE_PROBE(name);
    DTRACE_PROBE1(name, type1, arg1);
    DTRACE_PROBE2(name, type1, arg1, type2, arg2);
    DTRACE_PROBE3(name, type1, arg1, type2, arg2, type3, arg3);
    DTRACE_PROBE4(name, type1, arg1, type2, arg2, type3, arg3, type4, arg4);
Providers - fbt

- Function boundary tracing
  - Enable probes at kernel function entry and return points
  - Use requires some knowledge of the kernel

- Args
  - On entry probes, the arguments passed to the function are available as;
    - `args[]` array – typed
    - `arg0 … argn` – int64_t's
  - On return probes, function return values available in `args[1]`
Kernel Function Args...

mdb(1) & dtrace(1) – Perfect Together

```c
int (*)(struct vnode *, struct uio *, int, struct cred *, struct caller_context *)
```

```bash
# dtrace -n 'ufs_read:entry { printf("%s\n",stringof(args[0]->v_path));}';
dtrace: description 'ufs_read:entry ' matched 1 probe
```
```c
kmutex_t v_lock {
    void * [1] _opaque
} uint_t v_flag
uint_t v_count
void *v_data
struct vfs *v_vfsp
struct stdata *v_stream
enum vtype v_type
dev_t v_rdev
struct vfs *v_vfsmountedhere
struct vnodeops *v_op
struct page *v_pages
pgcnt_t v_npages
... char *v_path
...}
```
Providers - sysinfo

- DTrace probes that enable gathering values of kernel statistics – sys kstats;

```
nv98> kstat -n sys
module: cpu                             instance: 0
name:   sys                             class:    misc
bawrite                         139
bread                           1122
bwrite                          1418
canch                           66
. . .
```

- Args

  arg0 – Value by which the statistic will be incremented
  arg1 – Pointer to the current value
  arg2 – pointer to the cpu_t of the CPU the statistic is being incremented on
Providers - vminfo

- Similar to sysinfo – probes that correspond to named vm kstats
  - arg0 – value by which the stat will be incremented
  - arg1 – pointer to the current value of the stat
- Enables correlation of virtual memory events to processes/threads

```
#dtrace -n 'vminfo / execname != "dtrace" / { @vm[execname]=count(); }'
```
Providers - proc

• Events related to processes
  – create, exec, lwp-create, signals, etc

• The args vary, depending on which specific probe is enabled
  /usr/demo/dtrace/whoexec.d
Providers - pid

• Using DTrace to look up into userland!
  – No code modifications required – it's all dynamic!
    pid1234:shared_object:function:name
    pid3402:libc:malloc:entry

• PIDs can be set using the DTrace $target macro
  – Set when either -c <command> or -p <PID> is used
    'pid$target:::entry { @[probemod, probefunc] = count() }' -c date
Providers - plockstat

• User level lock statistics
  – Similar to what lockstat(1) does for kernel lock stats
  – User mutex locks and Reader/Writer locks

• Check out the -V option...
  – Will generate the actual D executing...

```sh
plockstat -V -A -p 840 > pl.out 2>&1
```
Providers

• io
  – disk input and output requests
  – I/O by device, process, size, filename

• mib
  – counters for management information bases
  – IP, IPv6, ICMP, IPSec

• sched
  – kernel scheduler events
  – on-cpu, off-cpu, resume, preempt
Providers

• fsinfo
  – file system operations of interest

• ip
  – network events (packet send/receive)
Providers

• DTrace refers to most providers as “Stable” providers
  - The probes and args will not change across releases
  - Provides for building a toolbox that will work indefinitely
  - io, sched, proc, vminfo, sysinfo, fpuinfo, mib, etc, are all stable providers
  - fbt is not, since fbt by definition instruments the kernel functions entry and return points.
  - It is generally recommended to stick with stable providers, at least while you're getting started
  - Check the documentation for the specific stability level of a provider
  - New providers under development!
Providers, cont.

- Examples

  - proc:::exec
  - sched:::oncpu
  - fbt:ufs:ufs_read:entry
  - syscall::read:entry{ printf("Process %d", pid); }
  - syscall::write:entry/execname=="firefox-bin"/
    { @[probefunc] = count(); }
  - sysinfo:::readch{ trace(execname); exit(0); }
  - sysinfo:::writech
  - io:::start
The D language

- A simple (?) dynamically interpreted language used by dtrace(1M)
- Similar to C language and awk(1):
  - Supports ANSI C operators and has support for strings
  - Supports several variable types, including built-in variables: pid, execname, timestamp, curthread, etc
- **No control-flow constructs:**
  - loops, if statements
- Arithmetic may only be performed on integers in D programs, floating-point arithmetic is not permitted in D
A DTrace D Program

probe
/ optional predicate /
{
    clause
    what to do when the probe(s) fire, and the predicate,
    if present, evaluates true
}

Example;

syscall::read:entry
/ execname == "java" /
{
    @reads[pid, fds[arg0].fi_pathname] = count();
}

Or, via the command line;

#dtrace -n 'syscall::read:entry / execname == "java" /
    { @reads[pid, fds[arg0].fi_pathname] = count(); }'
The D language, cont.

- **Data Types**
  - **Integer types**
    - char
    - short
    - int
    - long
    - long long
  - **Float types**
    - float
    - double
    - long double
  - **String type**
    - string
The D language, cont.

- Operators
  - Arithmetic Operators, similar as in ANSI C
    
    +, -, *, /, %
    
    may only be performed on integer operands, or on pointers
    not applicable on floats
  - Relational Operators
    
    >, >=, <, <=, ==, !=
  - Logical Operators
    
    &&, ||, ^^  
  - Assignment Operators, similar as in ANSI C
    
    =, +=, ANSI-C compliant
The D language, cont.

• Variables: no need to declare them

• Scalar Variables
  – represents integers, strings, pointers
  – Three different types that define the variable scope;
    Global
    Thread-Local
    Clause-Local
  – created automatically – D figures out the type
The D Language

- Global variables
  - Visible in every clause of the D program
  - Name and data storage location define once

```d

dtrace:::BEGIN
{
    x = 123;
}

int n;

dtrace:::END
{
    n = 456;
    printf("n: %d, x: %d\n", n, x);
}
```

- Global variable x
- Explicit variable declaration, not needed. You can do this outside probe clause
The D language, cont.

• Thread-local variables
  
  – Variable storage local to each OS thread
  – Useful for setting trace flags
  – Use the “self->” identifier to declare a thread-local variable
  – Example which associates a thread-local variable called flag in function entry to trace desired kernel thread in corresponding return function

```d
syscall::write:entry
/ pid == 3406 /
{
    self->flag = 1;
}
syscall::write:return
/ self->flag /
{
    self->flag = 0;
    ....
```
The D language, cont.

- Thread-local variables useful for computing the time spent in functions

- Example:

```d
syscall::read:entry
{
    self->st = timestamp;
}
syscall::read:return
/ self->st / /* this is the same as “self->st != 0” */
{
    self->rt = timestamp - self->st;
    self->st = 0;
    printf("PID %d, read time: %d\n", pid, self->rt);
}
```
The D language, cont.

- **Clause-Local Variables**
  - Their storage is reused for each program clause
  - Similar to automatic variables in a C, C++, or Java language
  - Are created on their first assignment
  - Referenced and assigned by using “this->” operator

```d
BEGIN
{
  this->secs = timestamp / 1000000000;
  ...
}
```
The D language, cont.

- **Associative Arrays**
  - Collection of data elements
  - No predefined number of elements
  - Used to simulate hashes or data dictionaries
  - Very simple to use and different than a scalar array
  - Defined as: `name[key] = expression`

  - `e.g.: a[123,"abc"] = 456`
  - `(a is associative array: a[int, string] stores an integer)`
The D language, cont.

• Built-in Variables
  - pid: the current process ID
  - execname: the current executable name
  - timestamp: the time since boot, in nanoseconds
  - curthread: the current thread
  - probeprov, probemod, probefunc and probename identify the current probe name fields

• External Variables
  - used in some other parts: OS, kernel modules. e.g: `kmem_flags, `physmem
The D language, cont.

- Scripting in D
- Easy to create D scripts to hold one or more probe clauses
- All D scripts end in dot d (script_name.d)
- Add the interpreter as the first line in the script
  ```
  #!/usr/sbin/dtrace -s
  ```
- Or create the script and run as;
  ```
  #dtrace -s ./script.d
  ```
Actions & Subroutines

• Taken when a probe fires
• Indicated by following a probe specification with “\{ action \}”
• Actions trace data and modify state external to DTrace
  – Data recording actions operate on the principle buffer
  – The default action when a probe fires is to generate the CPU ID the probe fired on, the numeric ID of the DTrace probe, and the probe function and name

• Subroutines affect internal DTrace state
Actions, cont.

- **Data Recording Actions**
  - `trace(expression)`
    - records the result of trace to the directed buffer
    - `trace(pid)` traces the current process id
    - `trace(execname)` traces the current application name
  - `printf()`
    - traces a D expression
    - allows output style formatting
    - `printf("execname is %s", execname);`
  - `printa(aggregation)`
    - used to display and format aggregations
    - `printa(@aggl)`
    - `printa("%-@32s, %-@8d\n", @execs, @pids);`
• **Data Recording Actions**
  
  - `stack()`
    - records a kernel stack trace
    - `dtrace -n 'syscall::open:entry{ stack(); }'`
  
  - `ustack()`
    - records a user process stack trace
    - allows to inspect userland stack processes
    - `dtrace -n 'syscall::open:entry{ ustack(); }' -c ls`
  
  - `jstack()`
    - similar with `ustack()`, but specifically for Java
    - more space for deeper stack frames and longer symbol strings
## Destructive Actions

- used to change the state of the system
- use with **caution**, it is disabled by default!!

<table>
<thead>
<tr>
<th>Process Destructive</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop()</td>
<td>Stops the process which has executed the probe</td>
</tr>
<tr>
<td>raise()</td>
<td>Used to signal a process at a precise point during execution</td>
</tr>
<tr>
<td>copyout, copyoutstr()</td>
<td></td>
</tr>
<tr>
<td>system()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel Destructive</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>breakpoint()</td>
<td>Stops the system and transfers the control to the kernel debugger</td>
</tr>
<tr>
<td>panic()</td>
<td>Triggers a panic. Used to force a crash dump</td>
</tr>
<tr>
<td>chill()</td>
<td>A sophisticated routine to inject a short delay. Used for timings measurements</td>
</tr>
</tbody>
</table>
Actions, cont.

- **Special Actions**
  - `exit(int)` - stop tracing and exits

- Other subroutines:
  - `alloca()` – allocates a n size bytes buffer
  - `basename()` - formats the path names
  - `copyin()` - creates a buffer and returns its address
  - `copyinstr()` - creates a buffer and returns its address
  - `rand()` - returns a weak pseudo-random number
  - `strlen()` - returns the length of a string in bytes
  - `strjoin()` - returns a string as a concatenation of str1 and str2
Actions, cont.

• Examples
  - syscall:::
  - syscall:::entry
  - syscall:::return
  - syscall:::read:entry{ printf("Process %d", pid); }
  - syscall:::write:entry/execname=="firefox-bin"/
    { @[probefunc] = count(); }
  - sysinfo:::readch{ trace(execname); exit(0); }
  - sysinfo:::writech
  - io:::
Predicates

- D expressions that define a conditional test
- Allow actions to only be taken when certain conditions are met. A predicate has this form: `/predicate/`
- The actions will be activated only if the value of the predicate expression is true
- Used to filter and meet certain conditions: look only for a process which has the pid = 1203, match a process which has the name firefox-bin
Predicates, cont.

• Examples
  - syscall:::
  - syscall:::entry
  - syscall:::return
  - syscall:::read::entry { printf(“Process %d”, pid); }
  - syscall:::write::entry/execename=="firefox-bin"/
    { @[probefunc] = count(); }
  - sysinfo:::readch { trace(execname); exit(0); }
  - sysinfo:::writetch
  - io:::
Aggregations

- Used to aggregate data and look for trends
- Simple to generate reports about: total system calls used by a process or an application, the total number of read or writes by process...
- Has the general form:
  \[ \text{@name[keys]} = \text{aggfunc(args)} \]
- There is no need to use other tools like: awk(1), perl(1)
- The general definition of aggregating function:
  \[ f(f(x_0) \cup f(x_1) \cup ... \cup f(x_n)) = f(x_0 \cup x_1 \cup ... \cup x_n) \]
Aggregations

• Aggregating functions
  – count() : the number of times called, used to count for instance the total number of reads or system calls
  – sum() : the total value of the specified expressions
  – avg() : the arithmetic average of the specified expression
  – min() : the smallest value of the specified expression
  – max() : the largest value of the specified expression
  – quantize() : a power-of-two frequency distribution, simple to use to draw distributions

• Non-aggregating functions
  – mode and median
Aggregations, cont.

- What's going on with my system?
  \[
  \text{dtrace } -n \text{ syscall:::entry}
  \]

- Difficult to read, start aggregating...
  \[
  \text{dtrace } -n \ '\text{syscall:::entry}\{@[execname] = \text{count}();\}'
  \]

- Filter on read system call
  \[
  \text{dtrace } -n \ '\text{syscall::read*:entry}\{@[execname]=\text{count}();\}'
  \]

- Add the file descriptor information
  \[
  \text{dtrace } -n \ '\text{syscall::read*:entry}\{@[execname,arg0]=\text{count}();\}'
  \]
Aggregations, cont.

- Drill-down and get a distribution of each read by application name

```c
syscall::read*:entry
{
    self -> ts = timestamp;
}

syscall::read*:return
/self -> ts/
{
    @time[execname] = quantize(timestamp - self->ts);
    self->ts = 0;
}
```
Aggregations, cont.

• Data normalization
  – used to aggregate over a specific constant reference: e.g.: system calls per second
  – normalize()
  – denormalize()

• Truncate
  – used to minimize the aggregation results, keep certain top results
  – trunc( aggregation, trunc value)
Pointers and Arrays

• Pointers determines which location in memory we are referencing
• Similar mechanism as in ANSI-C
• Safe access and control of pointers by DTrace
• Invalid memory access and alignment checks

```c
BEGIN
{
    x = (int *)NULL;
    y=*x;       // Redacted
    trace(y);
}
```
Pointers and Arrays, cont.

- Support for scalar arrays, similar with C/C++
- Indexed from 0, fixed length
- Sometimes used to access certain OS array data structures
- Defined as: `int a[int]`
  
  Example: `int a[4];` 4 elements: `a[0], a[1], a[2], a[3]`

- Scalar and associative arrays

<table>
<thead>
<tr>
<th>Item</th>
<th>Predefined Size</th>
<th>Consecutive storage order</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Array</td>
<td>Yes</td>
<td>Yes</td>
<td><code>int a[4]</code></td>
</tr>
<tr>
<td>Associative Array</td>
<td>No</td>
<td>No</td>
<td><code>a[123,&quot;abc&quot;]</code></td>
</tr>
</tbody>
</table>
DTrace Framework

- Introduction
- Probes, Providers, Actions, Predicates
- The D language
- Aggregations
- Pointers and Arrays
- Strings
- Structs and Unions
- Output formatting
- Speculative tracing
Strings

- Support for strings in D
- Built-in data type very easy to use
- Strings constants defined between " "
- String assignment using = operator
  - Example: s = "my string";
- String comparison using the relational operators ( <, >, <=, >=, ==, != )
  - Example: execname == "firefox-bin"
- Comparison is done byte-by-byte as in C like in strcmp(3C) routine
DTrace Framework

- Introduction
- Probes, Providers, Actions, Predicates
- The D language
- Aggregations
- Pointers and Arrays
- Strings
- **Structs and Unions**
- Output formatting
- Speculative tracing
Output formatting

- Special routines to format the output: trace(), printf() or printa()
- For specific output format use built-in printf()
  - `printf("execname is %s", execname);`
  - `printf("%d spent %d secs in read\n", pid, timestamp - t);`
- For aggregations use printa()
  - `printa("Aggregation is:", @a);`
  - `printa(@count);`
- Basic trace()
  - `trace(execname);`
DTrace Methods and Use

• Learning the mechanics of DTrace is great, but DTrace is, after all, a tool
• Like any tool, it's usefulness depends on the skill set and experience of the user
• The great news is DTrace is really easy to use!
  – It's easy to do simple things in DTrace that tell you a LOT about what your system and application is doing
• With time and experience, you'll only get better at root-causing sticky problems
DTrace One Liners

- **System Calls Count by Application**
  
  $ dtrace -n 'syscall:::entry{@[execname] = count();}''$

- **System Calls Count by Application and Process**
  
  $ dtrace -n 'syscall:::entry{@[execname,pid] = count();}''$

- **How many times a file has been opened**
  
  $ dtrace -n 'syscall::open::entry{@[copyinstr(arg0)] = count();}''$
DTrace One Liners

- Files Opened by process
  
  ```
  $ dtrace -qn
  'syscall::open*:entry{ printf("%s %s\n",execname,copyinstr(arg0)); }'
  ```

- Read Bytes by process
  
  ```
  $ dtrace -n 'sysinfo:::readch{ @[execname] = sum(arg0);}'
  ```

- Write Bytes by process
  
  ```
  $ dtrace -n 'sysinfo:::writech{ @[execname] = sum(arg0);}'
  ```
DTrace One Liners, cont.

- How big a read is
  
  $ \texttt{dtrace -n 'syscall::read:entry[@\{execname\} = quantize(arg2);']}'$

- How big a write is
  
  $ \texttt{dtrace -n 'syscall::write:entry[@\{execname\} = quantize(arg2);']}'$

- Disk size by process
  
  $ \texttt{dtrace -qn 'io:::start\{printf(\"%d %s %d\n", pid, execname, args[0]->b_bcount); \}'$
DTrace One Liners, cont.

- **High system time**
  
  ```
  $ dtrace -n profile-501'[@[stack()] = count()]END{trunc(@, 25)}'
  ```

- **What processes are using fork**
  
  ```
  $ dtrace -n 'syscall::fork*:entry{printf("%s %d",execname,pid);}'
  ```
The DTrace Toolkit

Brendan Gregg developed the toolkit
Stefan Parvu wrote the slides
DTraceToolkit

- Introduction
- Installation and Setup
- Toolkit elements
- Categories
- Free your mind
- Examples
Introduction

● The DTraceToolkit is a collection of useful documented scripts developed by the OpenSolaris DTrace community built on top of DTrace framework

● Available under www.opensolaris.org

● Ready DTrace scripts

● The toolkit contains:
  - the scripts
  - the man pages
  - the example documentation
  - the notes files
  - the tutorials
Script Categories: collection of D scripts

DTrace Toolkit

Applications  Cpu  Disk  Kernel  Network  Memory  Processes  System  Extra, User, System

DTrace Framework
DTrace Toolkit

- Introduction
- Installation and Setup
- Toolkit elements
- Categories
- Free your mind
- Examples
Installation and Setup

• Download the toolkit
  http://www.opensolaris.org/os/community/dtrace/dtracetoolkit

• Installation Notes
  – gunzip and "tar xvf" the file
  – run ./install – default installation /opt/DTT
  – read Guide to find out how to get started
  – a list of scripts is in Docs/Contents

• Setup DTT
  – PATH=$PATH:/opt/DTT/Bin
  – MANPATH=$MANPATH:/opt/DTT/Man
    (assuming the toolkit was installed in /opt/DTT)
DTrace Toolkit

- Introduction
- Installation and Setup
- **Toolkit Elements**
- Categories
- Free your mind
- Examples
Toolkit Elements

DTraceToolkit-X.XX/

Bin/ Symlinks to the scripts
Apps/ Application specific scripts
Cpu/ Scripts for CPU analysis
Disk/ Scripts for disk I/O analysis
Docs/ Documentation
  Contents Command list for the Toolkit
  Examples/ Examples of command usage
Faq Frequently asked questions
Links Further DTrace links
Notes/ Notes on Toolkit commands
Readme Readme for using the docs
Extra/ Misc scripts
Guide This file!
Kernel/ Scripts for kernel analysis
License The CDDL license
Locks/ Scripts for lock analysis
Man/ Man pages
  man1m/ Man pages for the Toolkit commands
Mem/ Scripts for memory analysis
Net/ Scripts for network analysis
Proc/ Scripts for process analysis
System/ Scripts for system analysis
User/ Scripts for user based activity analysis
Zones/ Scripts for analysis by zone
Version DTraceToolkit version
install Install script, use for installs only
Toolkit Elements, cont.

- Categories
  - Apps – scripts for certain applications: Apache, NFS
  - Cpu – scripts for measuring CPU activity
  - Disk – scripts to analyse I/O activity
  - Extra – other categories
  - Kernel – scripts to monitor kernel activity
  - Locks – scripts to analyse locks
  - Mem – scripts to analyse memory and virtual memory
  - Net – scripts to analyse activity of the network interfaces, and the TCP/IP stack
  - Proc – scripts to analyse activity of a process
  - System – scripts to measure system wide activity
  - User – scripts to monitor activity by UID
  - Zones – scripts to monitor activity by zone
Toolkit Elements, cont.

- Documentation
  - Man/: all scripts are documented as UNIX manual pages
  - Docs/: a generic place to find the documentation
  - Docs/Notes/: several short guides about toolkit's commands
  - Docs/Example/: examples of command usage
  - Docs/Content/: complete list of all commands
  - Docs/Faq/: DTT Frequently Asked Questions
DTrace Toolkit

- Introduction
- Installation and Setup
- Toolkit Elements
- Categories
- Free your mind
- Examples
Categories

• Applications

– Used to measure and report certain metrics from applications like: Apache Web server, NFS client, UNIX shell

– `httpdstat.d`: computes real-time Apache web statistics: the number of connections, GET, POST, HEAD and TRACE requests

– `nfswizard.d`: used to measure the NFS client activity regarding response time and file accesses

– `shellsnoop`: captures keystrokes, used to debug and catch command output. Use with caution!

– `weblatency.d`: counts connection speed delays, DNS lookups, proxy delays, and web server response time. Uses by default Mozilla browser
Categories, cont.

• Cpu
  - Reports and list the CPU activity like: cross calls, interrupt activity by device, time spent servicing interrupts, CPU saturation
  - `cputypes.d`: lists the information about CPUs: the number of physical install CPUs, clock
  - `loads.d`: prints the load average, similar to uptime
  - `intbycpu.d`: prints the number of interrupts by CPU
  - `intoncpu.d`: lists the interrupt activity by device; example: the time consumed by the ethernet driver, or the audio device
  - `inttimes.d`: reports the time spent servicing the interrupt
Categories, cont.

- **Cpu**
  - `xcallsbypid.d` – list the inter-processor cross-calls by process id. The inter-process cross calls is an indicator how much work a CPU sends to another CPU.
  - `dispqlen.d` – dispatcher queue length by CPU, measures the CPU saturation.
  - `cpuwalk.d` – identify if a process is running on multiple CPUs concurrently or not.
  - `runocc.d` – prints the dispatcher run queue, a good way to measure CPU saturation.
Categories, cont.

• Disk
  - Analyses I/O activity using the io provider from DTrace: disk I/O patterns, disk I/O activity by process, the seek size of an I/O operation
  - iotop: a top like utility which lists disk I/O events by processes
  - iosnoop: a disk I/O trace event application. The utility will report UID, PID, filename regarding for a I/O operation
  - bitesize.d: analyse disk I/O size by process
  - seeksize.d: analyses the disk I/O seek size by identifying what sort I/O operation the process is making: sequential or random
Categories, cont.

• Disk
  - `iofile.d`: prints the total I/O wait times. Used to debug applications which are waiting for a disk file or resource.
  - `iopattern`: computes the percentage of events that were of a random or sequential nature. Used easily to identify the type of an I/O operation and the average, totals numbers.
  - `iopending`: prints a plot for the number of pending disk I/O events. This utility tries to identify the "serialness" or "parallelness" of the disk behavior.
  - `diskhits`: prints the load average, similar to `uptime`.
  - `iofileb.d`: prints a summary of requested disk activity by pathname, providing totals of the I/O events in bytes.
Categories, cont.

- **FS**
  - Analyses the activity on the file system level: write cache miss, read file I/O statistics, system calls read/write
  - `vopstat`: traces the vnode activity
  - `rfsio.d`: provides statistics on the number of reads: the bytes read from file systems (logical reads) and the number of bytes read from physical disk
  - `fspaging.d`: used to examine the behavior of each I/O layer, from the syscall interface to what the disk is doing
  - `rfileio.d`: similar with `rfsio.d` but reports by file
Categories, cont.

• Kernel
  – Analyses kernel activity: DNLC statistics, CPU time consumed by kernel, the threads scheduling class and priority
  – `dnlnocstat`: inspector of the Directory Name Lookup Cache (DNLC)
  – `cputimes`: print CPU time consumed by the kernel, processes or idle
  – `cpudist`: print CPU time distributions by kernel, processes or idle
  – `cswstat.d`: prints the context switch count and average
  – `modcalls.d`: an aggregation for kernel function calls by module
Categories, cont.

• Kernel
  - dnlcps.d: prints DNLC statistics by process
  - dnlcsnoop.d: snoops DNLC activity
  - kstat_types.d: traces kstat reads
  - pridist.d: outputs the process priority distribution. Plots which process is on the CPUs, and under what priority it is
  - priclass.d: outputs the priority distribution by scheduling class. Plots a distribution
  - whatexec.d: determines the types of files which are executed by inspected the first four bytes of the executed file
Categories, cont.

• Locks
  - Analyses lock activity using lockstat provider
  - lockbydist.d: lock distribution by process name
  - lockbyproc.d: lock time by process name
Categories, cont.

• Memory
  – This category analyses memory and virtual memory things: virtual memory statistics, page management, minor faults
  – `vmstat.d`: a vmstat like utility written in D
  – `vmstat-p.d`: a vmstat like utility written in D which does display what “vmstat -p” does: reporting the paging information
  – `xvmstat`: a much improved version of vmstat which does count the following numbers: free RAM, virtual memory free, major faults, minor faults, scan rate
Categories, cont.

• Memory
  – `swapinfo.d`: prints virtual memory info, listing all memory consumers related with virtual memory including the swap physical devices
  – `pgpginbypid.d`: prints information about pages paged in by process id
  – `minfbypid.d`: detects the biggest memory consumer using minor faults, an indication of memory consumption
Categories, cont.

- **Network**
  - These scripts analyse the activity of the network interfaces and the TCP/IP stack. Some scripts are using the mib provider. Used to monitor incoming
  - `icmpstat.d`: reports ICMP statistics per second, based on mib
  - `tcpstat.d`: prints TCP statistics every second, retrieved from the mib provider: TCP bytes received and sent, TCP bytes retransmitted
  - `udpstat.d`: prints UDP statistics every second, retrieved from the mib provider
  - `tcpsnoop.d`: analyses TCP network packets and prints the responsible PID and UID. Useful to detect which processes are causing TCP traffic
Categories, cont.

- **Network**
  - connections: prints the inbound TCP connections. This displays the PID and command name of the processes accepting connections
  - `tcptop`: display top TCP network packets by process. It can help identify which processes are causing TCP traffic
  - `tcpwdist.d`: measures the size of writes from applications to the TCP level. It can help identify which process is creating network traffic
Categories, cont.

• Process
  - Analyses process activity: system calls/process, bytes written or read by process, files opened by process,
  - sampleproc: inspect how much CPU the application is using
  - threaded.d: see how well a multithreaded application uses its threads
  - writebytes.d: how many bytes are written by process
  - readbytes.d: how many bytes are read by process
  - kill.d: a kill inspector. What how signals are send to what applications
  - newproc.d: snoop new processes as they are executed
Categories, cont.

• Process

- syscallbyproc.d & syscallbypid.d: system calls by process or by PID
- filebyproc.d: files opened by process
- fddist: a file descriptor reporter, used to print distributions for read and write events by file descriptor, by process. Used to determine which file descriptor a process is doing the most I/O with
- pathopens.d: prints a count of the number of times files have been successfully opened
- rwbypid.d: reports the no. of read/writes calls by PID
- rwbytype.d: identifies the vnode type of read/write activity - whether that is for regular files, sockets, character special devices
Categories, cont.

• Process
  - **sigdist.d**: prints the number of signals received by process and the signal number
  - **topsysproc**: a report utility listing top number of system calls by process
  - **pfilestat**: prints I/O statistics for each file descriptor within a process. Very useful for debug certain processes
  - **stacksize.d**: measures the stack size for running threads
  - **crash.d**: reports about crashed applications. Useful to identify the last seconds of a crashed application
  - **shortlived.d**: snoops the short life activity of some processes
Categories, cont.

- **System**
  - Used to measure system wide activity
  - `uname-a.d`: simulates 'uname -a' in D
  - `syscallbysysc.d`: reports a total on the number of system calls on the system
  - `sar-c.d`: reports system calls usage similar to 'sar -c'
  - `topsyscall`: prints a report of the top system calls on the system
DTrace Toolkit

- Introduction
- Installation and Setup
- Toolkit Elements
- Categories
- Free your mind
- Real Examples
DTrace Toolkit

- Introduction
- Installation and Setup
- Toolkit Elements
- Categories
- Free your mind
- Real Examples
1. High System Calls

- A case where `vmstat 1` reports a high number of system calls
- What to do?
  - Count the total number of system calls
  - Use a simple DTrace aggregation to find out what application are responsible for that
  - Think to enhance the aggregation for a better reporting or better...
- Use DTT utilities to find out what is going on, getting as well a nice report
## 1. High System Calls, cont.

<table>
<thead>
<tr>
<th>kthr</th>
<th>memory</th>
<th>page</th>
<th>disk</th>
<th>faults</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b w</td>
<td>swap</td>
<td>free</td>
<td>re mf</td>
<td>pi po</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883592</td>
<td>1077152</td>
<td>28</td>
<td>281</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883592</td>
<td>1077152</td>
<td>28</td>
<td>278</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883592</td>
<td>1077152</td>
<td>28</td>
<td>281</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883592</td>
<td>1077152</td>
<td>28</td>
<td>296</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883592</td>
<td>1077152</td>
<td>28</td>
<td>304</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>278</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>279</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>279</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>282</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>27</td>
<td>276</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>282</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>278</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>279</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>279</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>279</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>281</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>278</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>2883584</td>
<td>1077144</td>
<td>28</td>
<td>279</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
1. High System Calls, cont.

- Start a simple aggregation:
  $ dtrace -n 'syscall:::entry {@[execname] = count();}'

- Select the top consumer and start aggregating again:
  $ dtrace -n 'syscall:::entry/execname=="your-app"/ {@[probefunc] = count();}'

- Count the number of system calls globally:
  $ dtrace -n 'syscall:::entry{@[probefunc] = count();}'

- Better run `topsysproc` from Proc Category
1. High System Calls, cont.

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>httpd</td>
<td>3</td>
</tr>
<tr>
<td>xscreensaver</td>
<td>9</td>
</tr>
<tr>
<td>mixer_applet2</td>
<td>10</td>
</tr>
<tr>
<td>nscd</td>
<td>10</td>
</tr>
<tr>
<td>gnome-netstatus-</td>
<td>12</td>
</tr>
<tr>
<td>intrd</td>
<td>15</td>
</tr>
<tr>
<td>java</td>
<td>18</td>
</tr>
<tr>
<td>gnome-panel</td>
<td>31</td>
</tr>
<tr>
<td>webservd</td>
<td>43</td>
</tr>
<tr>
<td>tput</td>
<td>49</td>
</tr>
<tr>
<td>vmstat</td>
<td>51</td>
</tr>
<tr>
<td>gnome-terminal</td>
<td>62</td>
</tr>
<tr>
<td>dtrace</td>
<td>65</td>
</tr>
<tr>
<td>soffice.bin</td>
<td>69</td>
</tr>
<tr>
<td>at-spi-registryd</td>
<td>72</td>
</tr>
<tr>
<td>sh</td>
<td>122</td>
</tr>
<tr>
<td>clear</td>
<td>136</td>
</tr>
<tr>
<td>Xorg</td>
<td>151</td>
</tr>
<tr>
<td>firefox-bin</td>
<td>151</td>
</tr>
<tr>
<td>realplay.bin</td>
<td>1473</td>
</tr>
</tbody>
</table>
1. High System Calls, cont.

**Conclusions:**

- Not able to see who does all those system calls using basic utilities: `vmstat`, `iostat`, `prstat`
- Easy to detect and get the report about the top system calls consumers using DTT utility: `topsysproc`
2. High CPU Utilization

- There is a high CPU utilisation under the system without any sign who is generating that.
- What to do?
- Does it help to run: `prstat`, `mpstat`, `vmstat`, `iostat`?
- Solve the problem by using: `topsysproc`, and `execsnoop` from DTT.
2. High CPU Utilisation, cont.

- The output from `vmstat 1`:

```
ketb   memory   page   disk   faults   cpu
r  b  w  swap  free  re  mf  pi  po  fr  de  sr  cd  cd  cd  f0  in  sy  cs  us  sy  id
0  0  0 2791884 983816  13   169   2   0   0   2   0   0   0   0  903 1550 805  2  1  98
0  0  0 2762448 973096  5510  74407  0   0   0   0   0   0   0   0  2847 51987 5458 14 44 43
0  0  0 2762448 973124  5429  73284  0   0   0   0   0   0   0   0  2741 51068 5333 15 43 42
0  0  0 2762548 973096  5445  73504  0   0   0   0   0   0   1   0   0  2710 51428 5335 13 45 42
0  0  0 2762432 973084  5446  73548  0   0   0   0   0   0   0   0  2758 51364 5343 14 43 43
0  0  0 2762476 973044  5454  73573  0   0   0   0   0   0   0   0  2791 51366 5433 14 43 42
0  0  0 2762576 973128  5459  73745  0   0   0   0   0   0   0   0  2776 51501 5408 14 44 42
0  0  0 2762576 973128  5514  74416  0   0   0   0   0   0   0   0  2821 51881 5429 14 43 44
0  0  0 2762468 973032  5419  73135  0   0   0   0   0   0   0   0  2774 51382 5331 15 43 42
0  0  0 2762476 973040  5485  74017  0   0   0   0   0   0   0   0  2806 51692 5438 13 43 43
0  0  0 2762540 973092  5431  73348  0   0   0   0   0   0   0   0  2757 51242 5332 14 43 42
0  0  0 2762504 973080  5493  74114  0   0   0   0   0   0   0   0  2771 51682 5407 14 43 43
0  0  0 2762440 973100  5431  73367  0   0   0   0   0   3   0   0   0  2784 51210 5365 14 43 42
1  0  0 2762576 973128  5446  73504  0   0   0   0   0   0   0   0  2765 51299 5336 14 43 43
0  0  0 2762448 973128  5438  73422  0   0   0   0   0   0   0   0  2863 51713 5629 14 44 43
0  0  0 2762564 973116  5441  73401  0   0   0   0   0   0   0   0  2835 52062 5700 15 43 42
0  0  0 2762432 973084  5428  73341  0   0   0   0   0   0   0   0  2850 51972 5662 14 44 42
0  0  0 2762500 973064  4656  63220  0   0   0   0   0   0   0   0  2644 52488 6327 28 41 31
```
2. High CPU Utilisation, cont.

- The output from `mpstat 1`:

```plaintext
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
0 51776 0 0 1446 559 2109 130 612 574 0 32116 19 56 0 25
1 21034 0 0 1165 9 3126 129 352 285 0 18848 14 30 0 56
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
0 58151 0 0 1374 546 1975 107 623 648 0 35360 19 62 0 20
1 14682 0 0 1230 10 3236 88 282 245 0 15526 13 24 0 63
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
0 53541 0 0 1324 552 1828 139 608 589 0 32613 26 56 0 18
1 18246 0 0 1163 17 3291 135 286 238 0 18093 15 29 0 56
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
0 45416 0 0 1516 551 2257 142 548 572 0 29019 19 50 0 31
1 28010 0 0 1168 10 3081 121 397 349 0 22440 12 36 0 52
```
2. High CPU Utilisation, cont.

- The output from `prstat -a`:

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>SIZE</th>
<th>RSS STATE</th>
<th>PRI NICE</th>
<th>TIME</th>
<th>CPU</th>
<th>PROCESS/NLWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>8120</td>
<td>sparyu</td>
<td>1204K</td>
<td>716K run</td>
<td>0</td>
<td>0:01:51</td>
<td>5.9%</td>
<td>ksh/1</td>
</tr>
<tr>
<td>2961</td>
<td>root</td>
<td>197M</td>
<td>174M sleep</td>
<td>59</td>
<td>0:46:03</td>
<td>3.6%</td>
<td>Xorg/1</td>
</tr>
<tr>
<td>3169</td>
<td>sparyu</td>
<td>70M</td>
<td>32M sleep</td>
<td>59</td>
<td>0:05:40</td>
<td>2.5%</td>
<td>gnome-terminal/2</td>
</tr>
<tr>
<td>6971</td>
<td>sparyu</td>
<td>63M</td>
<td>14M sleep</td>
<td>59</td>
<td>0:04:11</td>
<td>1.3%</td>
<td>realplay.bin/1</td>
</tr>
<tr>
<td>6725</td>
<td>sparyu</td>
<td>129M</td>
<td>77M sleep</td>
<td>59</td>
<td>0:04:26</td>
<td>0.3%</td>
<td>firefox-bin/3</td>
</tr>
<tr>
<td>1922</td>
<td>root</td>
<td>5752K</td>
<td>4544K sleep</td>
<td>59</td>
<td>0:20:49</td>
<td>0.2%</td>
<td>intrd/1</td>
</tr>
<tr>
<td>7068</td>
<td>sparyu</td>
<td>249M</td>
<td>134M sleep</td>
<td>49</td>
<td>0:02:43</td>
<td>0.1%</td>
<td>soffice.bin/5</td>
</tr>
<tr>
<td>3291</td>
<td>sparyu</td>
<td>44M</td>
<td>7836K sleep</td>
<td>59</td>
<td>0:01:53</td>
<td>0.1%</td>
<td>at-spi-registry/1</td>
</tr>
<tr>
<td>3154</td>
<td>sparyu</td>
<td>50M</td>
<td>12M sleep</td>
<td>59</td>
<td>0:01:32</td>
<td>0.0%</td>
<td>gnome-netstatus/1</td>
</tr>
<tr>
<td>1967</td>
<td>root</td>
<td>102M</td>
<td>36M sleep</td>
<td>29</td>
<td>0:01:14</td>
<td>0.0%</td>
<td>webservd/31</td>
</tr>
<tr>
<td>1984</td>
<td>webservd</td>
<td>142M</td>
<td>54M sleep</td>
<td>59</td>
<td>0:01:11</td>
<td>0.0%</td>
<td>webservd/76</td>
</tr>
<tr>
<td>23319</td>
<td>sparyu</td>
<td>3416K</td>
<td>2872K cpu0</td>
<td>59</td>
<td>0:00:00</td>
<td>0.0%</td>
<td>prstat/1</td>
</tr>
<tr>
<td>1810</td>
<td>noaccess</td>
<td>177M</td>
<td>65M sleep</td>
<td>59</td>
<td>0:00:56</td>
<td>0.0%</td>
<td>java/28</td>
</tr>
<tr>
<td>3133</td>
<td>sparyu</td>
<td>49M</td>
<td>14M sleep</td>
<td>59</td>
<td>0:00:46</td>
<td>0.0%</td>
<td>metacity/1</td>
</tr>
<tr>
<td>3152</td>
<td>sparyu</td>
<td>51M</td>
<td>14M sleep</td>
<td>59</td>
<td>0:00:31</td>
<td>0.0%</td>
<td>wnck-applet/1</td>
</tr>
<tr>
<td>27399</td>
<td>sparyu</td>
<td>71M</td>
<td>36M sleep</td>
<td>59</td>
<td>0:00:13</td>
<td>0.0%</td>
<td>gimp-2.0/1</td>
</tr>
<tr>
<td>3156</td>
<td>sparyu</td>
<td>48M</td>
<td>11M sleep</td>
<td>59</td>
<td>0:00:25</td>
<td>0.0%</td>
<td>mixer_applet2/1</td>
</tr>
<tr>
<td>6983</td>
<td>sparyu</td>
<td>1204K</td>
<td>908K sleep</td>
<td>59</td>
<td>0:00:00</td>
<td>0.0%</td>
<td>ksh/1</td>
</tr>
<tr>
<td>3137</td>
<td>sparyu</td>
<td>56M</td>
<td>19M sleep</td>
<td>59</td>
<td>0:00:31</td>
<td>0.0%</td>
<td>gnome-panel/1</td>
</tr>
<tr>
<td>3111</td>
<td>sparyu</td>
<td>6052K</td>
<td>3036K sleep</td>
<td>59</td>
<td>0:00:06</td>
<td>0.0%</td>
<td>xscreensaver/1</td>
</tr>
<tr>
<td>3116</td>
<td>sparyu</td>
<td>8148K</td>
<td>3740K sleep</td>
<td>59</td>
<td>0:00:05</td>
<td>0.0%</td>
<td>gnome-smproxy/1</td>
</tr>
<tr>
<td>360</td>
<td>root</td>
<td>4660K</td>
<td>1848K sleep</td>
<td>59</td>
<td>0:00:00</td>
<td>0.0%</td>
<td>automountd/2</td>
</tr>
<tr>
<td>480</td>
<td>root</td>
<td>1736K</td>
<td>544K sleep</td>
<td>59</td>
<td>0:00:00</td>
<td>0.0%</td>
<td>smcboot/1</td>
</tr>
<tr>
<td>478</td>
<td>root</td>
<td>1740K</td>
<td>944K sleep</td>
<td>59</td>
<td>0:00:00</td>
<td>0.0%</td>
<td>smcboot/1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NPROC</th>
<th>USERNAME</th>
<th>SIZE</th>
<th>RSS MEMORY</th>
<th>TIME</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>sparyu</td>
<td>1491M</td>
<td>543M 27%</td>
<td>0:26:49</td>
<td>10%</td>
</tr>
<tr>
<td>39</td>
<td>root</td>
<td>560M</td>
<td>307M 15%</td>
<td>1:08:55</td>
<td>3.8%</td>
</tr>
<tr>
<td>1</td>
<td>webservd</td>
<td>142M</td>
<td>54M 2.6%</td>
<td>0:01:11</td>
<td>0.0%</td>
</tr>
<tr>
<td>1</td>
<td>noaccess</td>
<td>177M</td>
<td>65M 3.2%</td>
<td>0:00:56</td>
<td>0.0%</td>
</tr>
<tr>
<td>1</td>
<td>smmnp</td>
<td>6872K</td>
<td>1480K 0.1%</td>
<td>0:00:00</td>
<td>0.0%</td>
</tr>
<tr>
<td>1</td>
<td>lp</td>
<td>2936K</td>
<td>1084K 0.1%</td>
<td>0:00:00</td>
<td>0.0%</td>
</tr>
<tr>
<td>4</td>
<td>daemon</td>
<td>11M</td>
<td>6004K 0.3%</td>
<td>0:00:00</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Total: 104 processes, 371 lwp's, load averages: 1.36, 1.30, 0.96
2. High CPU Utilisation, cont.

- Run `topsysproc`:

```
2006 May 28 17:43:08, load average: 0.56, 0.22, 0.12  syscalls: 46333

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>gnome-vfs-daemon</td>
<td>3</td>
</tr>
<tr>
<td>httpd</td>
<td>3</td>
</tr>
<tr>
<td>mixer_applet2</td>
<td>8</td>
</tr>
<tr>
<td>xscreensaver</td>
<td>9</td>
</tr>
<tr>
<td>gnome-netstatus-</td>
<td>10</td>
</tr>
<tr>
<td>intrd</td>
<td>15</td>
</tr>
<tr>
<td>java</td>
<td>20</td>
</tr>
<tr>
<td>gnome-panel</td>
<td>31</td>
</tr>
<tr>
<td>mpstat</td>
<td>35</td>
</tr>
<tr>
<td>tput</td>
<td>49</td>
</tr>
<tr>
<td>webservrd</td>
<td>49</td>
</tr>
<tr>
<td>dtrace</td>
<td>62</td>
</tr>
<tr>
<td>firefox-bin</td>
<td>84</td>
</tr>
<tr>
<td>soffice.bin</td>
<td>108</td>
</tr>
<tr>
<td>sh</td>
<td>122</td>
</tr>
<tr>
<td>clear</td>
<td>136</td>
</tr>
<tr>
<td>Xorg</td>
<td>628</td>
</tr>
<tr>
<td>gnome-terminal</td>
<td>2455</td>
</tr>
<tr>
<td>ksh</td>
<td>9727</td>
</tr>
<tr>
<td>date</td>
<td>32778</td>
</tr>
</tbody>
</table>
```
2. High CPU Utilisation, cont.

- Run `execsnoop`:

```
sparvu@earth> ./execsnoop
UID   PID   PPID  ARGS
 100  13575  2540  date
 100  13576  2540  date
 100  13577  2540  date
 100  13578  2540  date
 100  13579  2540  date
 100  13580  2540  date
 100  13581  2540  date
 100  13582  2540  date
 100  13583  2540  date
 100  13584  2540  date
 100  13585  2540  date
 100  13586  2540  date
 100  13587  2540  date
 100  13588  2540  date
 100  13589  2540  date
 100  13590  2540  date
 100  13591  2540  date
 100  13592  2540  date
 100  13593  2540  date
 100  13594  2540  date
 100  13595  2540  date
 100  13596  2540  date
 100  13597  2540  date
```
2. High CPU Utilisation, cont.

• Conclusions:
  - A high CPU utilisation was detected by `vmstat` and `prstat`. However, the CPU consumption was not easy related to any process on the system.
  - Using DTT utilities: `topsysproc` and `execsnoop` the real problem was very easily found and the process/owner generating all the load was easy identified.
3. High Cross-Calls

- It has been detected on a multiprocessor server a high number of inter-processor cross-calls per second. This was discovered using `mpstat`.
- Inter-processor cross-calls is a number indicating how often CPUs are sending the work from one to another. A clear indication of overhead.
- Investigate using `mpstat` and see if it is easy to find out who generates all these cross-calls.
- Solve the problem by using: `xcallsbypid.d` from DTT Cpu category.
3. High Cross-Calls, cont.

- `mpstat` reports:

```
  CPU  minf mjf  xcal  intr  ithr  csw  icsw  migr  smtx  srw  syscl  usr  sys  wt  idl
0   0   0   0  494   371  260   1   36   1   0  307   1   1   0  98
1   0   0   0  125    3   25   8   48   2   0  552   1   0  0  99
0   0   0   0  517   380  371   2   76   9   0  839   2   0  0  98
1   0   0   0  152    5   406  459   7   0  817   2   1  0  97
0   0   0   4  506   384  279   6   52   4   0  306   1   0  0  99
1   0   0   1  154   10   312  650   1   0  272   0   0  0 100
0   0   0   0  684   443  431  13  60  13   0  702  10   1  0  89
1   0   0   0  288    7  714   9  63   2   0  906   5   1  0  94
0   171  93  5915 4832   746  2227 318  117 392   0 143341 13  37  0  50
1   573  62  3507 7098   4971  648   128 247   0  54178 23   19  0  58
0   0   0  3089 4004  410  3283 468 126 3364   0  79532 16   47  0  38
1   0   4  2715 4010   9  3296 541 121 3500   0  83183 16   49  0  36
0   0   0  3274 5373  391  2660 377 148 1904   0  67229 16  37  0  47
1   2   0  4236 4169   5  3172 683 142 2076   0  88053 18   53  0  29
```
3. High Cross-Calls, cont.

- Run `xcallsbypid.d` from Cpu category:
3. High Cross-Calls, cont.

• Conclusions:
  - Solaris's `mpstat` was used to identify the high `xcalls`, however `mpstat` was not reporting on who was generating that big number.
  - Very easy to identify the process/application which was generating lots of cross calls directly using DTT utility: `xcallsbypid.d`
4. Network Connections

- The network status utility `netstat` displays a status of all network connections on a system.
- With the current tools there is no easy way to find out and co-relate a network connection with a process or the owner of it.
- Extra tools like `lsof` can list what connections were made and by who.
- What about incoming connections?
- Solve the problem by using: `tcptop`, `tcpsnoop` and `connections` utilities from DTT.
### 4. Network Connections, cont.

**Under Net category execute:** `tcpsnoop`

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>LADDR</th>
<th>LPORT</th>
<th>DR</th>
<th>RADDR</th>
<th>RPORT</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>→</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>→</td>
<td>212.58.224.163</td>
<td>554</td>
<td>470</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>66</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>→</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>381</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>→</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>511</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>1514</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>→</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>1514</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>632</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>624</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>226</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>307</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>137</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>271</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>236</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>231</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>137</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>11336</td>
<td>192.168.1.5</td>
<td>42931</td>
<td>←</td>
<td>212.58.224.163</td>
<td>554</td>
<td>54</td>
<td>realplay.bin</td>
</tr>
<tr>
<td>100</td>
<td>2287</td>
<td>192.168.1.5</td>
<td>52043</td>
<td>→</td>
<td>72.5.124.61</td>
<td>80</td>
<td>54</td>
<td>firefox-bin</td>
</tr>
<tr>
<td>100</td>
<td>2287</td>
<td>192.168.1.5</td>
<td>52043</td>
<td>→</td>
<td>72.5.124.61</td>
<td>80</td>
<td>706</td>
<td>firefox-bin</td>
</tr>
<tr>
<td>100</td>
<td>2287</td>
<td>192.168.1.5</td>
<td>52043</td>
<td>←</td>
<td>72.5.124.61</td>
<td>80</td>
<td>66</td>
<td>firefox-bin</td>
</tr>
<tr>
<td>100</td>
<td>2287</td>
<td>192.168.1.5</td>
<td>52043</td>
<td>→</td>
<td>72.5.124.61</td>
<td>80</td>
<td>54</td>
<td>firefox-bin</td>
</tr>
<tr>
<td>100</td>
<td>2287</td>
<td>192.168.1.5</td>
<td>52043</td>
<td>←</td>
<td>72.5.124.61</td>
<td>80</td>
<td>54</td>
<td>firefox-bin</td>
</tr>
</tbody>
</table>

- To display top network packets run `tcptop`:

```
2006 May 28 18:31:34, load: 0.28, TCPin: 104 KB, TCPout: 20 KB

UID   PID    LADDR      LPORT   RADDR      RPORT   SIZE   NAME
100    2287   192.168.1.5 52155   65.205.8.181 80   1078   firefox-bin
100    11359  192.168.1.5 43839   212.58.227.71 80   1331   realplay.bin
100    11359  192.168.1.5 59306   212.58.224.54 554   1672   realplay.bin
100    2287   192.168.1.5 36402   72.5.124.59   80   2730   firefox-bin
100    2287   192.168.1.5 58374   216.52.17.7   80   2983   firefox-bin
100    2287   192.168.1.5 39219   72.5.124.59   80   4420   firefox-bin
100    2287   192.168.1.5 44541   72.5.124.61   80   8753   firefox-bin
100    2287   192.168.1.5 48599   72.5.124.61   80  19620   firefox-bin
100    2287   192.168.1.5 64240   212.58.227.71 80  24082   firefox-bin
100    2287   192.168.1.5 47685   72.5.124.61   80  47258   firefox-bin
100    2287   192.168.1.5 56155   212.58.227.71 80  49685   firefox-bin
```

- To monitor and check the incoming connections run `connections`:

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>CMD</th>
<th>TYPE</th>
<th>PORT</th>
<th>IP_SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>266</td>
<td>inetd</td>
<td>tcp</td>
<td>23</td>
<td>192.168.1.3</td>
</tr>
<tr>
<td>0</td>
<td>422</td>
<td>sshd</td>
<td>tcp</td>
<td>22</td>
<td>192.168.1.3</td>
</tr>
<tr>
<td>80</td>
<td>1984</td>
<td>webservd</td>
<td>tcp</td>
<td>80</td>
<td>192.168.1.3</td>
</tr>
<tr>
<td>0</td>
<td>422</td>
<td>sshd</td>
<td>tcp</td>
<td>22</td>
<td>192.168.1.3</td>
</tr>
<tr>
<td>0</td>
<td>422</td>
<td>sshd</td>
<td>tcp</td>
<td>22</td>
<td>192.168.1.3</td>
</tr>
<tr>
<td>0</td>
<td>266</td>
<td>inetd</td>
<td>tcp</td>
<td>21</td>
<td>192.168.1.3</td>
</tr>
</tbody>
</table>

- Conclusions:
  - Not very easy to relate network connections to processes on the system or list the top of connections
  - Net category has a lot of scripts which can easily help like: `tcpsnoop`, `tcptop` and `connections`
5.Disk Utilization

- Disk utilisation can be monitored using `iostat` – but to co-relate the utilisation with a process is a hard mission
- There are tools to check CPU usage by process but there are no tools to check disk I/O by process
- The old good friend: `iostat -xnmp`
- I/O type: reading `iostat` data a SysAdmin can describe if the I/O is sequential or random
5. Disk Utilization, cont.

- It is important to know what type of I/O there is: sequential or random.
- How can you list what processes are generating I/O, or list disk events or how much a process is using the disk (size of the disk event or the service time of the disk events)?
- Easily use the following DTT scripts: `iotop`, `iosnoop` from DTT root directory.
5. Disk Utilization, cont.

- One Liner says:

```
sparvu@earth>dtrace -n 'io:::start{printf("%d %s %d",pid,execname,args[0]->b_bcount);};'
dtrace: description 'io:::start' matched 6 probes

CPU ID FUNCTION:NAME
0 71 bdev_strategy:start 5637 bart 8192
0 71 bdev_strategy:start 5637 bart 4096
0 71 bdev_strategy:start 5637 bart 3072
0 71 bdev_strategy:start 5637 bart 8192
0 71 bdev_strategy:start 5637 bart 8192
0 71 bdev_strategy:start 5637 bart 12288
0 71 bdev_strategy:start 5637 bart 4096
0 71 bdev_strategy:start 5637 bart 20480
0 71 bdev_strategy:start 5637 bart 12288
0 71 bdev_strategy:start 5637 bart 4096
0 71 bdev_strategy:start 5637 bart 3072
0 71 bdev_strategy:start 5640 find 1024
0 71 bdev_strategy:start 5640 find 1024
0 71 bdev_strategy:start 5640 find 1024
0 71 bdev_strategy:start 5640 find 1024
0 71 bdev_strategy:start 5637 bart 32768
0 71 bdev_strategy:start 5640 find 2048
0 71 bdev_strategy:start 5637 bart 8192
0 71 bdev_strategy:start 5640 find 2048
0 71 bdev_strategy:start 5637 bart 24576
0 71 bdev_strategy:start 5637 bart 3072
0 71 bdev_strategy:start 5640 find 1024
1 71 bdev_strategy:start 5640 find 1024
```
5. Disk Utilization, cont.

- Run `iotop`:

```plaintext
2006 Jun 4 14:40:33, load: 0.27, disk_r: 10416 KB, disk_w: 8 KB

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>PPID</th>
<th>CMD</th>
<th>DEVICE</th>
<th>MAJ</th>
<th>MIN</th>
<th>D</th>
<th>%I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1968</td>
<td>1</td>
<td>gconfd-2</td>
<td>cmdk0</td>
<td>102</td>
<td>7</td>
<td>W</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>121</td>
<td>1</td>
<td>nscld</td>
<td>cmdk0</td>
<td>102</td>
<td>0</td>
<td>R</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>5568</td>
<td>1</td>
<td>gnome-panel-scre</td>
<td>cmdk0</td>
<td>102</td>
<td>1</td>
<td>R</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>1968</td>
<td>1</td>
<td>gconfd-2</td>
<td>cmdk0</td>
<td>102</td>
<td>7</td>
<td>R</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>392</td>
<td>386</td>
<td>Xorg</td>
<td>cmdk0</td>
<td>102</td>
<td>0</td>
<td>R</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>5555</td>
<td>4816</td>
<td>bart</td>
<td>cmdk0</td>
<td>102</td>
<td>7</td>
<td>R</td>
<td>16</td>
</tr>
<tr>
<td>100</td>
<td>5568</td>
<td>1</td>
<td>gnome-panel-scre</td>
<td>cmdk0</td>
<td>102</td>
<td>7</td>
<td>R</td>
<td>21</td>
</tr>
<tr>
<td>100</td>
<td>5568</td>
<td>1</td>
<td>gnome-panel-scre</td>
<td>cmdk0</td>
<td>102</td>
<td>0</td>
<td>R</td>
<td>54</td>
</tr>
</tbody>
</table>
```
5. Disk Utilization, cont.

- Run now *iosnoop*:

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>D</th>
<th>BLOCK</th>
<th>SIZE</th>
<th>COMM PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>3475216</td>
<td>8192</td>
<td>bart /opt/openoffice.org2.0/program/libres680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>3475232</td>
<td>8192</td>
<td>bart /opt/openoffice.org2.0/program/libres680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>3475248</td>
<td>16384</td>
<td>bart /opt/openoffice.org2.0/program/libres680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>3462668</td>
<td>2048</td>
<td>bart /opt/openoffice.org2.0/program/libres680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56037520</td>
<td>8192</td>
<td>bart &lt;none&gt;</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038128</td>
<td>8192</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038168</td>
<td>8192</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038192</td>
<td>4096</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038272</td>
<td>8192</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038296</td>
<td>4096</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038336</td>
<td>20480</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038392</td>
<td>8192</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038416</td>
<td>16384</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038528</td>
<td>45056</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038688</td>
<td>36864</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038792</td>
<td>53248</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038952</td>
<td>4096</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038968</td>
<td>4096</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56038984</td>
<td>4096</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56039040</td>
<td>57344</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56039152</td>
<td>4096</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56039224</td>
<td>4096</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
<tr>
<td>100</td>
<td>5603</td>
<td>R</td>
<td>56039288</td>
<td>8192</td>
<td>bart /opt/openoffice.org2.0/program/libsb680si.so</td>
</tr>
</tbody>
</table>
5. Disk Utilization, cont.

- How much the process reads... use `bitesize.d`:

<table>
<thead>
<tr>
<th>PID</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>5602</td>
<td>find /opt/0</td>
</tr>
</tbody>
</table>

```
value -------------- Distribution -------------- count
  512 | | | | | | | | | | | 0
 1024 | | | | | | | | | | 21
 2048 | | | | | | | | | | 2
 4096 | | | | | | | | | | 0
 8192 | | | | | | | | | | 3
16384 | | | | | | | | | | 0

5611 find /\0

value -------------- Distribution -------------- count
  512 | | | | | | | | | | 0
 1024 | | | | | | | | | | 886
 2048 | | | | | | | | | | 71
 4096 | | | | | | | | | | 21
 8192 | | | | | | | | | | 208
16384 | | | | | | | | | | 0

5603 bart create -I\0

value -------------- Distribution -------------- count
  512 | | | | | | | | | | 0
 1024 | | | | | | | | | | 127
 2048 | | | | | | | | | | 64
 4096 | | | | | | | | | | 70
 8192 | | | | | | | | | | 334
16384 | | | | | | | | | | 83
32768 | | | | | | | | | | 669
65536 | | | | | | | | | | 0
```
5. Disk Utilization, cont.

- Look for seek distance of the disk events. Run `seeksize.d` to understand if the I/O is sequential or not:

```
5603  bart create -I\0

<table>
<thead>
<tr>
<th>value</th>
<th>Distribution</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>914</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>128</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>256</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>512</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>2048</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>4096</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td>8192</td>
<td></td>
<td>170</td>
</tr>
<tr>
<td>16384</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>32768</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>65536</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>131072</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>262144</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>524288</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1048576</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2097152</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4194304</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>8388608</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>16777216</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>33554432</td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>67108864</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
```
5. Disk Utilization, cont.

- Other important DTT utilities used to measure and analyse disk I/O events
- `rwsnoop`: snoops the read/write operations
- `rwtop`: used to display the top read/write operations by process id
- `opensnoop`: used to snoop what files are being open and by who. Very easy to discover what processes are opening what files
5. Disk Utilization, cont.

- `rwtop` and `opensnoop`:

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>PPID</th>
<th>CMD</th>
<th>D</th>
<th>BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1954</td>
<td>1952</td>
<td>gnome-session</td>
<td>R</td>
<td>16</td>
</tr>
<tr>
<td>100</td>
<td>2194</td>
<td>2193</td>
<td>BitcX-1.1-final</td>
<td>R</td>
<td>59</td>
</tr>
<tr>
<td>100</td>
<td>5411</td>
<td>5405</td>
<td>firefox-bin</td>
<td>R</td>
<td>63</td>
</tr>
<tr>
<td>100</td>
<td>5411</td>
<td>5405</td>
<td>firefox-bin</td>
<td>W</td>
<td>63</td>
</tr>
<tr>
<td>100</td>
<td>5650</td>
<td>4816</td>
<td>gimp-2.0</td>
<td>R</td>
<td>64</td>
</tr>
<tr>
<td>100</td>
<td>5454</td>
<td>5443</td>
<td>soffice.bin</td>
<td>W</td>
<td>80</td>
</tr>
<tr>
<td>100</td>
<td>2101</td>
<td></td>
<td>nautilus</td>
<td>W</td>
<td>216</td>
</tr>
<tr>
<td>100</td>
<td>1982</td>
<td></td>
<td>xscreensaver</td>
<td>W</td>
<td>248</td>
</tr>
<tr>
<td>100</td>
<td>2125</td>
<td></td>
<td>clock-applet</td>
<td>W</td>
<td>320</td>
</tr>
<tr>
<td>100</td>
<td>5454</td>
<td>5443</td>
<td>soffice.bin</td>
<td>R</td>
<td>320</td>
</tr>
<tr>
<td>100</td>
<td>2129</td>
<td></td>
<td>gnome-netstatus</td>
<td>W</td>
<td>416</td>
</tr>
<tr>
<td>100</td>
<td>2099</td>
<td></td>
<td>gnome-panel</td>
<td>R</td>
<td>552</td>
</tr>
<tr>
<td>100</td>
<td>2101</td>
<td></td>
<td>nautilus</td>
<td>R</td>
<td>640</td>
</tr>
<tr>
<td>100</td>
<td>2194</td>
<td>2193</td>
<td>BitcX-1.1-final</td>
<td>W</td>
<td>681</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>init</td>
<td>W</td>
<td>824</td>
</tr>
<tr>
<td>100</td>
<td>1968</td>
<td></td>
<td>gconfd-2</td>
<td>R</td>
<td>832</td>
</tr>
<tr>
<td>100</td>
<td>2099</td>
<td></td>
<td>gnome-panel</td>
<td>W</td>
<td>920</td>
</tr>
</tbody>
</table>

```
2006 Jun 4 15:38:03, load: 0.33, app_r: 2883 KB, app_w: 2842 KB
```

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>COMMD</th>
<th>FD</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>5</td>
<td>/var/adm/utmpx</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>6</td>
<td>/var/adm/utmpx</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/1840/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>0</td>
<td>252</td>
<td>utmpd</td>
<td>7</td>
<td>/proc/2150/psinfo</td>
</tr>
<tr>
<td>100</td>
<td>5685</td>
<td>find</td>
<td>-1</td>
<td>/var/ld/ld.config</td>
</tr>
<tr>
<td>100</td>
<td>5685</td>
<td>find</td>
<td>3</td>
<td>/lib/libsec.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5685</td>
<td>find</td>
<td>3</td>
<td>/lib/libc.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5685</td>
<td>find</td>
<td>3</td>
<td>/lib/libavl.so.1</td>
</tr>
<tr>
<td>100</td>
<td>1968</td>
<td>gconfd-2</td>
<td>44</td>
<td>/export/home/sparvu/.gconfd/saved_state.tmp</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>-1</td>
<td>/var/ld/ld.config</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>3</td>
<td>/lib/libsec.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>3</td>
<td>/lib/libavd.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>3</td>
<td>/lib/libc.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>3</td>
<td>/lib/libavl.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5686</td>
<td>find</td>
<td>-1</td>
<td>/var/ld/ld.config</td>
</tr>
<tr>
<td>100</td>
<td>5686</td>
<td>find</td>
<td>3</td>
<td>/lib/libsec.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5686</td>
<td>find</td>
<td>3</td>
<td>/lib/libc.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5686</td>
<td>find</td>
<td>3</td>
<td>/lib/libavl.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>bart</td>
<td>3</td>
<td>/etc/default/init</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>bart</td>
<td>3</td>
<td>/usr/share/lib/zoninfo/Europe/Helsinki</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>sort</td>
<td>-1</td>
<td>/var/ld/ld.config</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>sort</td>
<td>3</td>
<td>/lib/libc.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>sort</td>
<td>-1</td>
<td>/proc/self/auxw</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>sort</td>
<td>3</td>
<td>/lib/ld/64/ld.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>sort</td>
<td>3</td>
<td>/lib/libavd.so.1</td>
</tr>
<tr>
<td>100</td>
<td>5688</td>
<td>sort</td>
<td>3</td>
<td>/dev/null</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>3</td>
<td>/opt/sfw/lib/firefox/LICENSE</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>3</td>
<td>/opt/sfw/lib/firefox/README.txt</td>
</tr>
<tr>
<td>100</td>
<td>5687</td>
<td>bart</td>
<td>3</td>
<td>/opt/sfw/lib/firefox/browserconfig.properties</td>
</tr>
</tbody>
</table>
DTrace & Java
DTrace and Java

- DTrace can be used to debug and observe Java applications
- Easy to start: use `jstack()`, to display the Java activity as a stack backtrace. `jstack()` based on `ustack()`
- Useful to understand the I/O and scheduling caused by your Java application
- Java 5: VM agents, shared libraries which are dynamically loaded when the VM starts
- Java 6, Mustang, introduces two new providers: hotspot and hotspot_jni
DTrace and Java, cont.

- `jstack()`
- The simplest form to record a stack trace from a Java application
  - Not `jstackstrsize` default of 512 may need to be increased
    
    ```
    dtrace -x jstackstrsize=1k -n syscall: ...
    ```

- Delivered already with DTrace framework:
  ```
  $ dtrace -n 'syscall:::entry/pid==xxx/
  {jstack(40);}'

  $ dtrace -n 'syscall:::entry/pid==xxx/
  {@[jstack(40)] = count();}'
  ```
jstack() Action

- jstack action prints mixed mode stack trace
- Both java frames and native (C/C++) frames are shown
- Only JVM versions 5.0_01 and later are supported
- jstack shows hex numbers for JVM versions before 5.0_01

```bash
#!/usr/sbin/dtrace -s
syscall::pollsys:entry
/ pid == $1 / {
    jstack(50,8192);
}
```

- first optional argument limits the number of frames shown
- second optional argument changes the string size
- jstackstrsize pragma / -x to increase buffer for all jstack()'s
jstack()

libc.so.1`_pollsys+0x7
libc.so.1`pselect+0x19e
libc.so.1`select+0x69
libXt.so.4`IoWait+0x36
libXt.so.4`_XtWaitForSomething+0x1a9
libXt.so.4`_XtAppPending+0x188
libmawt.so`0xd43d3928
libmawt.so`0xd43d37d6
libmawt.so`Java_sun_awt_motif_MToolkit_run+0x34
sun/awt/motif/MToolkit.run
java/lang/Thread.run
StubRoutines (1)
libjvm.so`__lCJavaCallsLcall_helper6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x187
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x14
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x2468_v+0x14
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x28
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x2be
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x37d
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x66
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x6d
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0xd0
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x51
libjvm.so`__lCJavaCallsLcall_wrapr6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v+0x105
libjvm.so`__lCG_start6Fpv_0+0xd2
libc.so.1`_thr_setup+0x51
libc.so.1`_lwp_start

397

libc.so.1`stat64+0x7
java/io/UnixFileSystem.getBooleanAttributes0*
0x20245c8b
932
The dvm Provider

- java.net project to add DTrace support in
  - 1.4.2 (libdvmpi.so)
  - 1.5 (libdvmti.so)
  - https://solaris10-dtrace-vm-agents.dev.java.net/
- Download shared libs
- Add location of libs to LD_LIBRARY_PATH variable
- Set JAVA_TOOL_OPTIONS to -Xrundvmti:all
- Name of provider - “dvm”
The dvm Provider: Probes

- dvm probes and their signatures
  - `vm-init()`, `vm-death()`
  - `thread-start(char *thread_name)`, `thread-end()`
  - `class-load(char *class_name)`
  - `class-unload(char *class_name)`
  - `gc-start()`, `gc-finish()`
  - `gc-stats(long used_objects, long used_object_space)`
  - `object-alloc(char *class_name, long size)`
  - `object-free(char *class_name)`
  - `method-entry(char *class_name, char *method_name, char *method_signature)`
  - `method__return(char *class_name, char *method_name, char *method_signature)`
The dvm Provider: alloc and free

- Object allocation/deallocation

```c
#!/usr/sbin/dtrace -qs
dvm$target:::object-alloc
{
    printf("%s allocated %d size objects\n", 
        copyinstr(arg0), arg1);
}

dvm$target:::object-free
{
    printf("%s freed %d size objects\n", 
        copyinstr(arg0), arg1);
}

# ./java_alloc.d -p `pgrep -n java`
```
The dvm Provider: Methods

- Count methods called

```
#!/usr/sbin/dtrace -s

dvm$target:::method-entry
{
    @[copyinstr(arg0),copyinstr(arg1)] = count();
}

# ./java_method_count.d -p `pgrep -n java`
```
The dvm provider: Time Spent

- Time spent in methods

```bash
#!/usr/sbin/dtrace -s
dvm$target:::method-entry
{
    self->ts[copyinstr(arg0),copyinstr(arg1)] = vtimestamp;
}

dvm$target:::method-return
{
    @ts[copyinstr(arg0),copyinstr(arg1)] = sum(vtimestamp - self->ts[copyinstr(arg0),
                          copyinstr(arg1)]);
}

# ./java_method.d -p `pgrep -n java`
```
DTrace and Java, cont.

• VM Agents

  – Some probes have a significant probe effect, and require enabling when the JVM is started
  
  -XX:+ExtendedDtraceProbes

  jinfo -XX:+ExtendedDtraceProbes
DTrace and Java, cont.

• Java 6, Mustang
  – Added two new providers: hotspot and hotspot_jni
  – Using these providers it is now possible to collect data from your Java applications
  – Hotspot_jni: probes related with Java Native Interface
  – Hotspot provider:
    - VM Probes: Initialization and Shutdown
    - Thread statistics Probes
    - Class loading and unloading Probes
    - Garbage Collection Probes
    - Method Compilation Probes
DTrace in JDK 6

- hotspot provider implements all dvm probes plus extensions:
  - Method compilation (method-compile-begin/end)
  - Compiled method load/unload(compiled-method-load/unload)
  - JNI method probes.
- DTrace probes as entry and return from each JNI method.
- Strings are now unterminated UTF-8 data. Always use associated length value with copyin restr().
Method Compilation Probes

```c
hotspot$1:::method-compile-begin {
    self->str = (char*) copyin(arg2, arg3+1);
    self->str[arg3] = '\0';
    self->classname = (string)self->str;
    self->str = (char*) copyin(arg4, arg5+1);
    self->str[arg5] = '\0';
    self->methodname = (string)self->str;
    printf("Compile begin %s.%s\n",
            self->classname, self->methodname);
}
```
Exception Stack Trace

```java
hotspot$1:::method-entry {
    self->ptr = (char*)copyin(arg1, arg2+1);
    self->ptr[arg2] = '\0';
    self->classname = (string)self->ptr;
    self->ptr = (char*)copyin(arg3, arg4+1);
    self->ptr[arg4] = '\0';
    self->methodname = (string)self->ptr;
}

hotspot$1:::method-entry

/self->classname == "java/lang/Throwable" &&
    self->methodname == "<init>"
{
    jstack();
}
```
JDK 6 DTrace Usage

• Certain probes are expensive
  – Turned off by default
  – object-alloc
  – method-entry, method-return
  – monitor probes
    • monitor-wait, monitor-contended-enter, etc

• Requires you to start your application with the flag
  -XX:+ExtendedDTraceProbes

• Use -XX:
  +DTrace{Alloc,Method,Monitor}Probes if possible
JDK6 hotspot_jni Provider

- Probes for Java Native Interface (JNI)
- Located at entry/return points of all JNI functions
- Probe arguments are same as corresponding JNI function arguments (for _entry probes)
- For XXX_return probes, probe argument is return value

Examples:

```
hotspot_jni$1:::GetPrimitiveArrayCritical_entry
hotspot_jni$1:::GetPrimitiveArrayCritical_return
```
JDK 1.6 and DTrace

• Check out

    /usr/jdk/jdk1.6.0_06/sample/dtrace

class_loading_stat.d The script collects statistics about loaded and unloaded Java classes and dump current state to stdout every N seconds.
gc_time_stat.d The script measures the duration of a time spent in GC. The duration is measured for every memory pool every N seconds.
hotspot_calls_tree.d The script prints calls tree of fired 'hotspot' probes.

method_compile_stat.d The script prints statistics about N methods with largest/smallest compilation time every M seconds.
method_invocation_stat.d The script collects statistics about Java method invocations.
method_invocation_stat_filter.d The script collects statistics about Java method invocations. You can specify package, class or method name to trace.
method_invocation_tree.d The script prints tree of Java and JNI method invocations.
monitors.d The script traces monitor related probes.
object_allocation_stat.d The script collects statistics about N object allocations every M seconds.
DTrace Community, cont.

• Solaris Internals 2nd
  – an update to Solaris Internals, for Solaris 10 and OpenSolaris. It covers Virtual Memory, File systems, Zones, Resource Management, Process Rights etc (all the good stuff in S10). This book is about 1100 pages

• New Solaris Performance and Tools!
  – aimed at Administrators to learn about performance and debugging. It's basically the book to read to understand and learn DTrace, MDB and the Solaris Performance tools, and a methodology for performance observability and debugging. This book is about 550 pages
DTrace Community, cont.

● Build around OpenSolaris community

● Available under www.opensolaris.org
  – The main page:
  http://www.opensolaris.org/os/community/dtrace/
  IRC on irc.freenode.net channels: #opensolaris, #dtrace

● The leaders:
  – Bryan M. Cantrill
  – Adam H. Leventhal
  – Mike Shapiro
  – Brendan Gregg

● Working with other communities
DTrace Community, cont.

- Jim Mauro and Richard McDougall: Solaris Internals
  - www.solarisinternals.com

- Lots of folks:
  - http://www.opensolaris.org/os/community/dtrace/observers/

- How can you help? Use, Improve and Evangelize
Future

- Visualization tools
- Integration with Java 6
- New providers: Apache, Sun Java System Webserver
- DTrace and Zones: support already in Solaris Express builds
- Better documentation and more scripts
- DTrace and other operating systems:
  - FreeBSD: porting already done!
  - Linux: using SystemTap still experimental!
DTrace for Database Administrators
- Learn how to use DTrace
- Easy to use and experiment using DTrace Toolkit
- Understand how the entire database engine works
- Special glasses: I/O monitoring

DTracing Oracle!!!

Real Case Examples
Free your mind

- A new mentality when debugging and observe with DTrace
- See the entire system
- Discover certain locations you want to investigate and look
- Place probes there, where are you interested
- Wait and see when the probes are executing
- Observe these locations by discovering who, how and when are accessed
- Gather the results by building a report
Free your mind, cont.

- Using DTrace does not mean you should not use anymore: vmstat, iostat, mpstat, etc.
- Try to understand every monitoring tool
- You don't have to do everything using DTrace...e.g.: memory leaks use the best tool: libumem, dbx
- Solaris has a very rich support for monitoring and observability. Try to understand each tool and what is good for: memory, disk, network, cpu, tracing, process monitoring and debug, kernel debug
Coming Soon!

DTrace
DYNAMIC TRACING IN SOLARIS, MAC OS X AND FREEBSD

Jim Mauro, Brendan Gregg, Chad Mynhier, Tariq Magdon-Ismail
Thank you! (and to Jim Mauro et al.)

Harry J Foxwell, PhD
harry.foxwell@oracle.com

“open” artwork and icons by chandan: http://blogs.sun.com/chandan