DTrace & DTraceToolkit, 0.96

Stefan Parvu
System Administrator
stefan.parvu@sun.com
stefanparvu14@yahoo.com

License

Agenda - Day 1

- Observability, Performance and Debugging in Solaris
  - Solaris observability and debug tools
- The DTrace Framework
  - An introduction to DTrace framework
  - Language concepts
  - DTrace methods
- DTrace Internals
- DTrace One Liners
  - several examples of using DTrace

Agenda - Day 2

- The Toolkit: DTraceToolkit
  - An introduction to DTraceToolkit
  - How to think and use the toolkit
  - Real Examples
- DTrace and Java
- DTrace Community
  - The DTT Team
  - OpenSolaris and DTrace books
- Future

Agenda - Day 1

- Observability, Performance and Debugging in Solaris
  - Solaris observability and debug tools
- The DTrace Framework
  - An introduction to DTrace framework
  - Language concepts
  - DTrace methods
- DTrace Internals
- DTrace One Liners
  - several examples of using DTrace

Observability, Performance and Debugging in Solaris

- You need tools to observe and debug different situations like: analysing the system performance, debugging an application or understanding the system utilisation or saturation, debugging a system/kernel crash
- A big number of observability and debug utilities under Solaris
- Several areas: Process Control, Process Statistics, Process Debugging, Kernel Debugging and Statistics and System Statistics

Observability, Performance and Debugging in Solaris, cont.

Process Control

pgrep, pkill
pstop, prun, prctl
preap, pwait, dtrace

Process Statistics

pargs, pflags
pldd, psig
pstack, pmap
pfiles, ptree
prstat, ptree, ptime, dtrace

Process Debugging

truss
mdb
plockstat
dtrace

Kernel Debugging and Statistics

mdb
kmdb
lockstat
kstat
dtrace

System Statistics

vmstat, iostat, sar, mpstat, cpustat, busstat, kstat, nfsstat, netstat, dtrace

Observability, Performance and Debugging in Solaris, cont.

- Some utilities are process based, some are only inspecting certain parts of the system: disks, virtual memory, kernel
- Under Solaris 10 new and enhanced tools:
  - pfiles: adds support for filenames
  - pstack: the Java frames are visible
  - intrstat: reports intrerrupt statistics, based on DTrace
  - plockstat: reports user level lock statistics, based on DTrace
  - dtrace: a new dynamic tracing framework

Observability, Performance and Debugging in Solaris, cont.

- DTrace framework one of the most important and innovative things for observability and debug
- New way to debug and observe the entire system and understand the big picture
- Does not replace or retire other system utilities: pstack, pmap, libumem, truss, mdb, ...
- Use the right tool for the right job
- Sometimes you find new debug tools or improved versions under Solaris Express builds

Observability, Performance and Debugging in Solaris, cont.

- **Solaris 10** – Sun's supported distribution, free to be used in commercial and non-commercial environments. Support costs
- **Solaris Express** – Sun's official release of next Solaris, tested and released ~ every one, two months
- **Solaris Express Community Release** – Internal latest builds based on OpenSolaris, codename: Nevada, released ~ every 2,3 weeks
- **OpenSolaris** – The source code for ON (OS/Net) consolidation: the kernel (similar as kernel.org) and user-land utilities/tools

Observability, Performance and Debugging in Solaris, cont.

- Visit http://cvs.opensolaris.org/source/
- The entire source code for ON(OS and Net) and the userland utilities
- Examine and read the source code of certain utilities used for debug or monitoring: vmstat, iostat, mpstat, etc...
- DTrace framework is available as source code

Agenda - Day 1

- Observability, Performance and Debugging in Solaris
  - Solaris observability and debug tools
- **The DTrace Framework**
  - An introduction to DTrace
  - Language concepts
  - DTrace methods
- DTrace Internals
- DTrace One Liners
  - several examples of using DTrace

DTrace Framework

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

DTrace Introduction

- A new powerful framework for real-time analysis and observability. System and process centric
- Hard to debug transient problems with: truss(1), pstack(1), prstat(1M)
- Only mdb(1) designed for systemic problems but only for postmortem analysis
- Designed for live production systems: a totally safe way to inspect live data on production systems

DTrace Introduction, cont.

- Safe and comprehensive: over 30.000 data monitoring points, inspect kernel and user space level
- Reduced costs: solutions usually found in minutes or hours not days or months
- Flexibility: DTrace lets you create your own custom programs to dynamically instrument the system
- No need to instrument your applications, no need to stop or restart them

DTrace Framework

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

Probes

- Programmable sensors placed all over your Solaris system
- A probe fires when the event happens
- The anatomy of a generic D script or the general form of a probe clause:

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

Probes, cont.

- A D program consists of one or more probe clauses
- Every probe has two names: a unique ID and a string name
- When the probe fires certain actions are executed only if the predicate expression is true
- Any directives found outside probe clauses are defined as declarations

Probes, cont.

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

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

- END: fires when the trace finishes

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

Probes, cont.

- A simple “Hello World” example

```
BEGIN
{
 trace("hello, world");
 exit(0);
}
```

Remember the generic D script ?

probe description
/predicate/
{
actions
}

Probes, cont.

- A point of instrumentation, made available by a provider, which has a name
- A four-tuple uniquely identifies every probe

  (provider:module:function:name)
- Module and Function: places where you want to look
- Name: represents an entry point in that function (eg.: *entry* or *return*)

Probes, 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:::`

Providers

- A methodology for instrumenting the system
- Makes available all know probes
- Providers are offering 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 certain providers: syscall, lockstat, fbt, io, mib

Providers, cont.

- **syscall**
  - one of the most important provider
  - holds the entire communication from userland to kernel space
  - every system call on the system
- **proc**
  - handles: process, LWP creation and termination, signaling
- **sched**
  - CPU scheduling: why threads are sleeping, running
  - used usually to compute the CPU time, which threads are run by which CPU and for how long

Providers, cont.

- **io**
  - a better look on iostat, regarding the I/O system
  - disk input and output requests
  - I/O by device, process, size, filename
- **mib**
  - counters for management information bases
  - IP, IPv6, ICMP, IPSec
- **profile**
  - time based probing at specific interval of times
  - low overhead
  - profile-<interval> and tick-<interval>

Providers, cont.

- **fbt**
  - Function Boundary Tracing
  - entry and return points of Solaris kernel function
- **vminfo**
  - probes specific to VM kernel system
  - VM kernel statistics used in vmstat(1)
- **lockstat**
  - locks statistic
  - a better understanding of locking condition and behavior

Providers, 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**:::

Actions

- Are taken when a probe fires
- Actions are indicated by following a probe specification with "*{ action }*"
- Used to record data to a DTrace buffer
- Different types of actions:
  - data recording
  - destructive
  - special
- By default, data recording actions record data to the principal DTrace buffer

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(@agg1)`

Actions, cont.

- 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(), used for Java

    The stack depth frames is different than in ustack

Actions, cont.

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

Process Destructive	**Results**
stop()	Stops the process which has executed the probe
raise()	Used to signal a process at a precise point during execution
copyout, copyoutstr()	
system()	
Kernel Destructive	**Results**
breakpoint()	Stops the system abd transfers the control to the kernel debugger
panic()	Triggers a panic. Used to force a crash dump
chill()	A sophisticated routine to inject a short delay. Used for timings measurements

Actions, cont.

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

  - different 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

- Are D expressions
- 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`**`/execname=="firefox-bin"/`**` { @[probefunc] = count(); }`
  - `sysinfo:::readch{ trace(execname); exit(0); }`
  - `sysinfo:::writech`
  - `io:::`

Managing Probes

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

```
$ dtrace -l | wc -l
39570
```

  - 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
```

Managing Probes

- List all read function probes:

```
$ dtrace -l -f read
```

- Enabling probes
  - Activate a probe by not using '-l' option
  - Using the default action: indicates that the probe has been enabled and lists: the CPU, the probe number and name
  - 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
```

DTrace Framework

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

The D language

- A simple dynamically interpreted language what dtrace(1M) uses
- It is like a C language with some constructs similar with awk(1):
  - Supports ANSI C operators and has support for strings
  - D expressions are based on built-in variables: pid, execname, timestamp, curthread
- 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

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
  - created automatically
  - global variables

```
BEGIN
{
    x = 123;
}
```

Global variable x

```
int x;
BEGIN
{
    x = 123;
}
```

Explicit variable declaration, not needed. You can do this outside probe clause

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.

- **Thread-local variables**
  - Variable storage to each OS thread
  - When you want to enable a probe and you want each thread to be marked
  - Using the “self” identifier
  - A simple example which associates a thread-local variable called read for each thread

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

The D language, cont.

- Thread-local variables used sometimes to compute the time spent in some system calls
- A simple example:

```
syscall::read:entry
{
 self -> t = timestamp;
}


sycall::read:return
/self -> t != 0/
{
 printf(“%d/%d spent %d secs in read\n”, pid, tid, timestamp -
  t);
}
```

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
  - Can be referenced and assigned by applying the -> operator to the special identifier this

```
BEGIN
{
 this->secs = timestamp / 1000000000;
 ...
}
```

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
- **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
- Run the scripts as any other shell script on the system. Make sure the script has the executable bits on

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

syscall:::entry
/execname=="firefox-bin"/
{
 @[probefunc] = count();
}
```

DTrace Framework

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

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:

  @name[keys] = aggfunc(args)

- There is no need to use other tools like: awk(1), perl(1)
- The general definition of aggregating function:

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

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 ?

```
dtrace -n syscall:::entry
```

- Difficult to read, start aggregating...

```
dtrace -n 'syscall:::entry{@[execname] = count();}'
```

- Filter on read system call

```
dtrace -n
  'syscall::read*:entry{@[execname]=count();}'
```

- Add the file descriptor information

```
dtrace -n
  'syscall::read*:entry{@[execname,arg0]=count();}'
```

Aggregations, cont.

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

```
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)

DTrace Framework

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

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

```
BEGIN
{
 x = (int *)NULL;
 y=*x;
 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

Item	Predefined Size	Consecutive storage order	Form
Scalar Array	Yes	Yes	int a[4]
Associative Array	No	No	a[123,"abc"]

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 comparation using the relational operators (<, >, <=, >=, ==, !=)
  - Example: execname == "firefox-bin"
- Comparation 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

Structs and Unions

- Similar with ANSI-C struct and union
- Struct: easy way to create a new data type

```
struct identifier {
 data_type member1;
 data_type member2;
};

struct identifier variable;
```

- Union: similar with struct, exception is that data members occupy the same region of storage

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 Framework

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

Speculative tracing

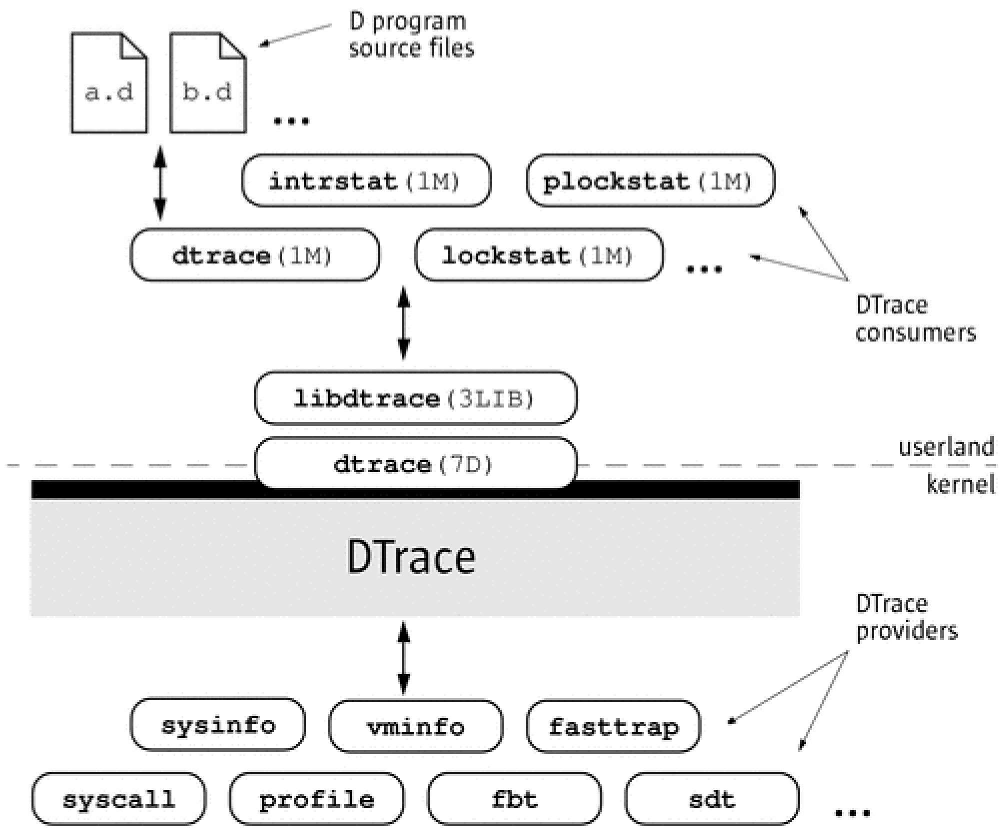
- Tentatively trace data and later commit or not to a trace buffer
- When you cannot use a predicate condition and don't know a probe event
- When you have an error event and would like to know the history behind it and why that error occurred
- Functions:
  - speculation()
  - speculate()
  - commit()
  - discard()

Agenda - Day 1

- Observability, Performance and Debugging in Solaris
  - Solaris observability and debug tools
- The DTrace Framework
  - An introduction to DTrace
  - Language concepts
  - DTrace methods
- **DTrace Internals**
- DTrace One Liners
  - several examples of using DTrace

DTrace Internals

- Compilation: D programs are compiled into a safe intermediate form that is used for execution when your probes fire which is validated by DTrace



DTrace Internals, cont.

- Programming mistakes: DTrace will report your errors to you and disable your instrumentation
- Execution environment: DTrace also handles any run-time errors: dividing by zero, dereferencing invalid memory, and so on, and reports them to you
- Safe: you can never construct a bad script that would cause DTrace to damage the Solaris kernel or one of the processes running on your system (do not confuse sometimes bugs in certain kernel subsystems which gets triggered by DTrace !)

DTrace Internals, cont.

- Safety one of the top priorities
  - 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 result 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.” http://en.wikipedia.org/wiki/Halting_problem

DTrace Internals, cont.

- Only root allowed to run DTrace by default
- To run DTrace you must have certain privileges:

```
$ ppriv -l | grep dtrace
dtrace_kernel
dtrace_proc
dtrace_user
```

- Enable using usermod utility
  - `# usermod -K defaultpriv=basic,dtrace_kernel,\ dtrace_proc,dtrace_user username`

Agenda - Day 1

- Observability, Performance and Debugging in Solaris
  - Solaris observability and debug tools
- The DTrace Framework
  - An introduction to DTrace
  - Language concepts
  - DTrace methods
- DTrace Internals
- **DTrace One Liners**
  - several examples of using DTrace

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

```
$ dtrace -n 'syscall::read:entry{@[execname]
  = quantize(arg2);}'
```

- How big a write is

```
$ dtrace -n
  'syscall::write:entry{@[execname] =
  quantize(arg2);}'
```

- Disk size by process

```
$ 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);}'
```

Top vs Prstat

- A real case using DTrace to analyse how much CPU top and prstat are using. Which one is better to run in your production servers ?
- Both tools used to monitor the performance of the system, in particular the process activity
- Remember: top is not part of Solaris. Install top from http://blastwave.org
- prstat(1M) reports active process statistics, similar with top but smarter. Part of Solaris 8+
- All measurements have been registered using a Dual Core AMD X2 4800+ !

Top vs Prstat, cont.

- Urban legends !?
  - Is *top* really a CPU hog application ?
  - There are many stories on net about SysAdmins which are never using *top* because they consider the application being CPU hog
  - Stories about using *top* on production env with performance penalties
  - Other folks are saying *top* is smarter, lighter and works similar as in other OSes: Linux, BSD* etc, no performance issues using *top*
  - A lot of confusion over this topic
- DTrace will help us to measure how much CPU *top* really uses comparing with prstat

Top vs Prstat, cont.

- top

```
load averages:  0.16,  0.09,  0.07                                  21:06:39
97 processes:  95 sleeping, 1 zombie, 1 on cpu
CPU states: 98.7% idle,  1.0% user,  0.3% kernel,  0.0% iowait,  0.0% swap
Memory: 2048M real, 1111M free, 713M swap in use, 2884M swap free

   PID USERNAME LWP PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
  2484 sparvu     5  49    0  304M  133M sleep   13:46  3.17% soffice.bin
   392 root       1  59    0  158M  136M sleep   12:22  2.75% Xorg
  2055 root       1  59    0 5752K 4676K sleep    2:02  0.23% intrd
  2150 sparvu     2  59    0   68M   23M sleep    1:22  0.11% gnome-terminal
  2059 sparvu     1  59    0   48M   13M sleep    0:09  0.09% metacity
  2127 sparvu     1  59    0   51M   14M sleep    0:09  0.08% wnck-applet
  2219 sparvu     3  49    0  112M   65M sleep    2:34  0.07% firefox-bin
  2516 sparvu     1  59    0   42M 6376K sleep    0:11  0.06% at-spi-registry
  2708 sparvu     1  59    0 1568K 1196K cpu/0    0:00  0.05% top
  2099 sparvu     1  59    0   53M   16M sleep    0:06  0.04% gnome-panel
  2129 sparvu     1  59    0   50M   13M sleep    0:15  0.03% gnome-netstatus
  2114 root      31  29   10  103M   37M sleep    0:10  0.02% webservd
  2147 webservd  75  59    0  147M   55M sleep    0:09  0.01% webservd
  1810 noaccess  28  59    0  202M   91M sleep    0:15  0.01% java
  2288 sparvu     1  59    0 2072K 1360K sleep    0:05  0.01% mpstat
  2244 sparvu     1  59    0 5240K 3360K sleep    0:04  0.01% ssh
  2131 sparvu     1  59    0   47M   10M sleep    0:03  0.01% mixer_applet2
  2101 sparvu     5  59    0   60M   24M sleep    0:04  0.01% nautilus
  2005 sparvu     1  59    0 8132K 3808K sleep    0:01  0.01% gnome-smproxy
  2328 sparvu     1  59    0   67M   29M sleep    0:39  0.00% gnome-pdf-viewe
  2326 sparvu     1  49    0   51M   15M sleep    0:01  0.00% gpdf
  1954 sparvu     1  59    0   54M   17M sleep    0:01  0.00% gnome-session
  2125 sparvu     1  59    0   48M   11M sleep    0:01  0.00% clock-applet
  2016 sparvu     1  59    0   47M 9160K sleep    0:01  0.00% gnome-settings-
  1982 sparvu     1  59    0 5984K 3108K sleep    0:01  0.00% xscreensaver
  1936 sparvu     1  59    0 2216K  900K sleep    0:01  0.00% dsdm
  2133 sparvu     1  59    0   47M 9404K sleep    0:00  0.00% notification-ar
   347 root       7  59    0 2760K 2140K sleep    0:06  0.00% vold
     9 root      16  59    0 9668K 8724K sleep    0:03  0.00% svc.configd
  1968 sparvu     1  59    0   12M 9956K sleep    0:01  0.00% gconfd-2
  2146 root       2  59    0   26M 8456K sleep    0:01  0.00% webservd
```

Top vs Prstat, cont.

- prstat

```
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
   392 root      156M  138M sleep   59    0   0:12:34 2.0% Xorg/1
  2484 sparvu    309M  142M sleep   49    0   0:13:51 1.8% soffice.bin/5
  2055 root     5752K 4676K sleep   59    0   0:02:04 0.2% intrd/1
  2150 sparvu     68M   23M sleep   59    0   0:01:23 0.1% gnome-terminal/2
  2724 sparvu     72M   38M sleep   49    0   0:00:05 0.1% gimp-2.0/1
  2059 sparvu     48M   14M sleep   59    0   0:00:11 0.1% metacity/1
  2219 sparvu    112M   65M sleep   49    0   0:02:35 0.1% firefox-bin/3
  2127 sparvu     51M   14M sleep   59    0   0:00:10 0.1% wnck-applet/1
  2516 sparvu     42M 6376K sleep   59    0   0:00:12 0.0% at-spi-registry/1
  2129 sparvu     50M   13M sleep   59    0   0:00:15 0.0% gnome-netstatus/1
  2737 sparvu   3400K 2852K cpu0    59    0   0:00:00 0.0% prstat/1
  2099 sparvu     53M   16M sleep   59    0   0:00:06 0.0% gnome-panel/1
  2147 webservd  147M   55M sleep   59    0   0:00:09 0.0% webservd/75
  2114 root      103M   37M sleep   29   10   0:00:10 0.0% webservd/31
  2288 sparvu   2072K 1360K sleep   59    0   0:00:05 0.0% mpstat/1
  1810 noaccess  202M   91M sleep   59    0   0:00:15 0.0% java/28
  2131 sparvu     47M   10M sleep   59    0   0:00:03 0.0% mixer_applet2/1
  2005 sparvu   8140K 3832K sleep   59    0   0:00:02 0.0% gnome-smproxy/1
  2101 sparvu     60M   24M sleep   59    0   0:00:05 0.0% nautilus/5
  1982 sparvu   5984K 3108K sleep   59    0   0:00:01 0.0% xscreensaver/1
  2244 sparvu   5240K 3360K sleep   59    0   0:00:04 0.0% ssh/1
  2243 sparvu   4696K 2948K sleep   59    0   0:00:00 0.0% ssh/1
   384 root     3412K 1672K sleep   59    0   0:00:00 0.0% syslogd/13
   461 lp       2936K 1236K sleep   59    0   0:00:00 0.0% httpd/1
   422 root     3048K 1048K sleep   59    0   0:00:00 0.0% lpsched/1
   457 root     2204K 1136K sleep   59    0   0:00:00 0.0% fbconsole/1
   360 root     4660K 1924K sleep   59    0   0:00:00 0.0% automountd/2
  2213 sparvu   1124K  904K sleep   59    0   0:00:00 0.0% run-mozilla.sh/1
   382 root       40M 2388K sleep   59    0   0:00:00 0.0% gdm-binary/1
   480 root     1736K  704K sleep   59    0   0:00:00 0.0% smcboot/1
   478 root     1740K 1100K sleep   59    0   0:00:00 0.0% smcboot/1
   237 daemon   2372K 1584K sleep   59    0   0:00:00 0.0% statd/1
   479 root     1736K  736K sleep   59    0   0:00:00 0.0% smcboot/1
   106 root     2188K 1404K sleep   59    0   0:00:00 0.0% syseventd/14
Total: 100 processes, 359 lwps, load averages: 0.14, 0.12, 0.09
```

Top vs Prstat, cont.

- Run top and prstat: interval 1sec x 10 times
- Record the number of system calls for *top* and *prstat* during the execution
- Use an aggregation like below:
  - `dtrace -n 'syscall:::entry/execname == "prstat"/ { @num[probefunc] = count(); }'`
  - `dtrace -n 'syscall:::entry/execname == "top"/ { @num[probefunc] = count(); }'`
- Compare the results

Top vs Prstat, cont.

- Total system calls for top

chdir	1
fsat	1
getgid	1
getpid	1
getrlimit	1
lseek	1
rexit	1
schedctl	1
setegid	1
seteuid	1
sigpending	1
sysi86	1
fcntl	2
setcontext	2
fstat64	3
open64	3
sigaction	4
stat64	4
memcntl	5
lwp_sigmask	6
munmap	7
doorfs	8
resolvepath	8
pollsys	9
xstat	9
gtime	10
uadmin	10
getuid	11
sysconfig	12
write	12
llseek	19
getdents64	20
brk	28
mmap	29
ioctl	64
close	1062
open	1062
read	1068

Top vs Prstat, cont.

- Total system calls for prstat

access	1
exece	1
fsat	1
fxstat	1
rexit	1
setrlimit	1
sysconfig	1
sysi86	1
fstat	2
getpid	2
systeminfo	2
xstat	2
fcntl	3
getrlimit	3
memcntl	3
read	3
stat	3
setcontext	4
munmap	6
resolvepath	7
pollsys	9
getloadavg	10
sigaction	10
ioctl	11
lseek	19
getdents	20
mmap	21
brk	26
write	32
close	108
open	112
doorfs	680
pread	1020

Top vs Prstat, cont.

- At this moment we know that *top* uses more system calls than prstat
- *top* seems to open, read and close many files several times. prstat does not do that
- What files *top* opens so many times ? What about prstat ?
  - `dtrace -n 'syscall::open:entry/execname == "top"/ { @name[copyinstr(arg0)] = count();}'`
  - `dtrace -n 'syscall::open:entry/execname == "prstat"/ { @name[copyinstr(arg0)] = count();}'`
- Compare the results

Top vs Prstat, cont.

- Opened files by top

```
2450/psinfo                                                       10
2471/psinfo                                                       10
2473/psinfo                                                       10
2484/psinfo                                                       10
2485/psinfo                                                       10
251/psinfo                                                        10
2516/psinfo                                                       10
252/psinfo                                                        10
267/psinfo                                                        10
268/psinfo                                                        10
2716/psinfo                                                       10
2724/psinfo                                                       10
2725/psinfo                                                       10
2749/psinfo                                                       10
2788/psinfo                                                       10
2789/psinfo                                                       10
3/psinfo                                                          10
347/psinfo                                                        10
360/psinfo                                                        10
382/psinfo                                                        10
384/psinfo                                                        10
386/psinfo                                                        10
392/psinfo                                                        10
421/psinfo                                                        10
422/psinfo                                                        10
427/psinfo                                                        10
457/psinfo                                                        10
460/psinfo                                                        10
461/psinfo                                                        10
478/psinfo                                                        10
479/psinfo                                                        10
480/psinfo                                                        10
7/psinfo                                                          10
9/psinfo                                                          10
90/psinfo     _                                                   10
```

Top vs Prstat, cont.

- Opened files by prstat

```
/proc/2485/psinfo                           1
/proc/251/psinfo                            1
/proc/2516/psinfo                           1
/proc/252/psinfo                            1
/proc/267/psinfo                            1
/proc/268/psinfo                            1
/proc/2716/psinfo                           1
/proc/2724/psinfo                           1
/proc/2725/psinfo                           1
/proc/2749/psinfo                           1
/proc/2796/psinfo                           1
/proc/2797/psinfo                           1
/proc/347/psinfo                            1
/proc/360/psinfo                            1
/proc/382/psinfo                            1
/proc/384/psinfo                            1
/proc/386/psinfo                            1
/proc/392/psinfo                            1
/proc/421/psinfo                            1
/proc/422/psinfo                            1
/proc/427/psinfo                            1
/proc/457/psinfo                            1
/proc/460/psinfo                            1
/proc/461/psinfo                            1
/proc/478/psinfo                            1
/proc/479/psinfo                            1
/proc/480/psinfo                            1
/proc/7/psinfo                              1
/proc/9/psinfo                              1
/proc/90/psinfo                             1
/var/ld/64/ld.config                        1
/var/ld/ld.config                           1
/usr/share/lib/terminfo//x/xterm            2
```

Top vs Prstat, cont.

- Partial results
- Interesting findings: *top* seems to open for each update every psinfo file from /proc. This happens 10 times (remember we started 1 sec x 10 times)
- *prstat* uses a smarter method, pread. No need to open 10 times each psinfo file, the file is opened once
- Does the difference matter ? So what if *top* uses more syscalls or opens more frequent the psinfo files ?

Top vs Prstat, cont.

- Measuring the CPU overhead between *prstat* and *top*

```
#!/usr/sbin/dtrace -s
syscall:::entry
/execname == $$1/
{
 self->start = vtimestamp;
}
syscall:::return
/self->start/
{
 this->time = vtimestamp - self->start;
 @Time[probefunc] = sum(this->time);
 @Time["TOTAL:"] = sum(this->time);
 self->start = 0;
}
```

Top vs Prstat, cont.

- The CPU overhead for top, ~ 60ms

```
open64                                     37270
stat64                                     40315
memcntl                                    76578
mmap                                       89981
pollsys                                   146723
write                                     332862
ioctl                                     591361
getdents64                               2856879
close                                    2983845
read                                    21667295
open                                    29061813
TOTAL:                                  58105511
```

- The CPU overhead for prstat, ~ 30ms

```
memcntl                                   201860
exece                                     321309
close                                     355196
write                                     708567
open                                      883755
doorfs                                   1085825
getdents                                 2870547
pread                                   21986669
TOTAL:                                  28924120
```

Top vs Prstat, cont.

- Conclusions
  - Using DTrace we were able to understand how *prstat* and *top* works
  - This was not intended to be a benchmark exercise !
  - For each screen update *top* opens, reads and closes a psinfo file for every process. *prstat* only does a read
  - On very fast machines the difference is small, however it very much depends how many processes are running. Try to experiment with different number of processes on different hardware
  - Try to discover other similar monitoring applications like *top*

Agenda - Day 2

- The Toolkit: DTraceToolkit
  - An introduction to DTraceToolkit
  - How to think and use the toolkit
  - Real Examples
- DTrace and Java
- DTrace Community
  - The DTT Team
  - OpenSolaris and DTrace books
- Future

The toolkit: DTraceToolkit

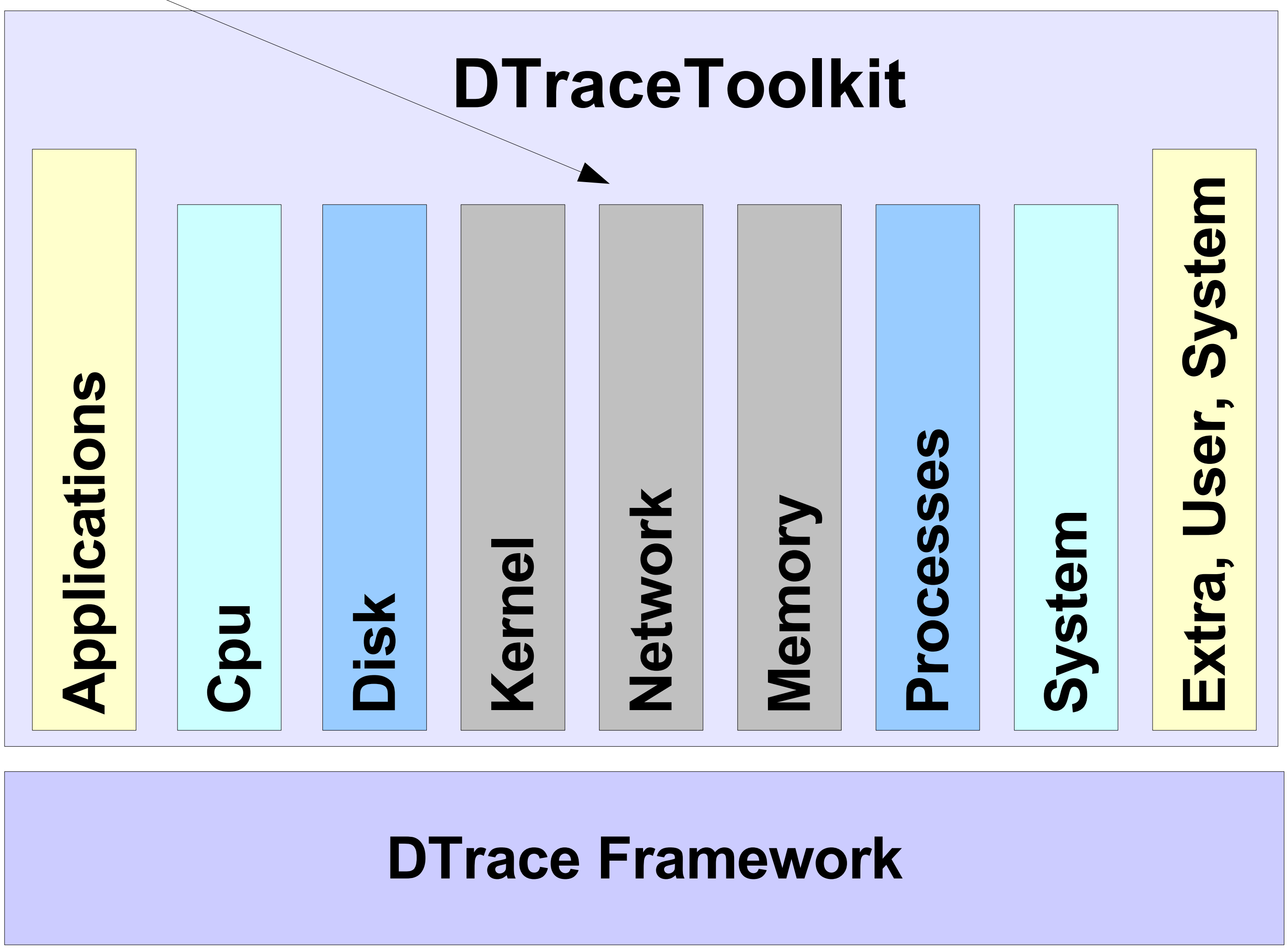
- **Introduction**
- Installation and Setup
- Toolkit elements
- Categories
- Free your mind
- Examples

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

Introduction, cont.

Script Categories: collection of D scripts



The toolkit: DTraceToolkit

- 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)

The toolkit: DTraceToolkit

- 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

The toolkit: DTraceToolkit

- 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
  - **dnlcstat**: 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 od 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

The toolkit: DTraceToolkit

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

The toolkit: DTraceToolkit

- 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.

```
 kthr      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 2883592 1077152 28 281 0 0  0  0  0  0  0  0  0 1563 2570 1591  2  1 97
 0 0 0 2883592 1077152 28 278 0 0  0  0  0  0  0  0  0 1535 2537 1546  2  1 97
 0 0 0 2883592 1077152 28 281 0 0  0  0  0  0  0  0  0 1852 3655 2168  2  2 96
 0 0 0 2883592 1077152 28 296 0 0  0  0  0  0  0  0  0 1902 4950 2421  4  2 94
 0 0 0 2883592 1077152 28 304 0 0  0  0  0 48  0  0  0 2175 6404 2979  9  2 89
 0 0 0 2883584 1077144 28 278 0 0  0  0  0  2  0  0  0 1903 5431 2568  6  2 91
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 2017 6956 2830  7  2 91
 0 0 0 2883584 1077144 28 278 0 0  0  0  0  0  0  0  0 1901 6855 2650  6  2 91
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 2114 9656 3387  8  3 89
 0 0 0 2883584 1077144 28 282 0 0  0  0  0  0  0  0  0 1774 5176 2292  4  2 94
 0 0 0 2883584 1077144 27 276 0 0  0  0  0  0  0  0  0 1651 2964 1742  2  2 96
 0 0 0 2883584 1077144 28 282 0 0  0  0  0  0  0  0  0 1546 2696 1552  2  1 97
 0 0 0 2883584 1077144 28 278 0 0  0  0  0  0  0  0  0 1900 4065 2287  3  2 95
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1644 3741 1883  4  1 95
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1982 5698 2650 11  2 87
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1844 3867 2223  3  1 95
 0 0 0 2883584 1077144 28 281 0 0  0  0  0  0  0  0  0 1555 2506 1542  1  1 97
 0 0 0 2883584 1077144 28 278 0 0  0  0  0  0  0  0  0 1475 2497 1495  2  1 96
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1501 2527 1542  2  1 97
 kthr      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 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1517 2531 1539  2  1 97
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1494 2510 1464  2  1 97
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1517 2576 1585  2  1 97
 0 0 0 2883584 1077144 28 281 0 0  0  0  0  0  0  0  0 1813 3656 2180  2  2 96
 0 0 0 2883584 1077144 29 281 0 0  0  0  0  0  0  0  0 1472 2475 1482  2  1 97
 0 0 0 2883584 1077144 28 278 0 0  0  0  0  0  0  0  0 1465 2508 1468  2  1 96
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1500 2491 1564  2  1 97
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1510 2526 1541  2  1 96
 0 0 0 2883584 1077144 28 279 0 0  0  0  0  0  0  0  0 1480 2534 1477  2  1 97
```

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.

```
2006 May 25 15:20:43, load average: 0.15, 0.12, 0.10   syscalls: 2552

  PROCESS                          COUNT
  httpd                                3
  xscreensaver                         9
  mixer_applet2                       10
  nscd                                10
  gnome-netstatus-                    12
  intrd                               15
  java                                18
  gnome-panel                         31
  webservd                            43
  tput                                49
  vmstat                              51
  gnome-terminal                      62
  dtrace                              65
  soffice.bin                         69
  at-spi-registryd                    72
  sh                                 122
  clear                              136
  Xorg                               151
  firefox-bin                        151
  realplay.bin                      1473
```

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 Utilisation

- 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*:

```
kthr      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  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  0 2847 51987 5458 14 44 43
0 0 0 2762448 973124 5429 73284 0 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  1  0  0  0 2710 51428 5335 13 45 42
0 0 0 2762432 973084 5446 73548 0 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  0 2791 51366 5433 14 43 42
0 0 0 2762576 973128 5459 73745 0 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  0 2821 51881 5429 14 43 44
0 0 0 2762468 973032 5419 73135 0 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  0 2806 51692 5438 13 43 43
0 0 0 2762540 973092 5431 73348 0 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  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  0 2765 51299 5336 14 43 43
0 0 0 2762448 973128 5438 73422 0 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  0 2835 52062 5700 15 43 42
0 0 0 2762432 973084 5428 73341 0 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  0 2644 52488 6327 28 41 31
```

2.High CPU Utilisation, cont.

- The output from *mpstat 1*:

```
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    1  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*:

```
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  8120 sparvu   1204K  716K run      0    4   0:01:51 5.9% ksh/1
  2961 root      197M  174M sleep   59    0   0:46:03 3.6% Xorg/1
  3169 sparvu     70M   32M sleep   59    0   0:05:40 2.5% gnome-terminal/2
  6971 sparvu     63M   14M sleep   59    0   0:04:11 1.3% realplay.bin/1
  6765 sparvu    129M   77M sleep   59    0   0:04:26 0.3% firefox-bin/3
  1922 root     5752K 4544K sleep   59    0   0:20:49 0.2% intrd/1
  7068 sparvu    249M  134M sleep   49    0   0:02:43 0.1% soffice.bin/5
  3291 sparvu     44M 7836K sleep   59    0   0:01:53 0.1% at-spi-registry/1
  3154 sparvu     50M   12M sleep   59    0   0:01:32 0.0% gnome-netstatus/1
  1967 root      102M   36M sleep   29   10   0:01:14 0.0% webservd/31
  1984 webservd  142M   54M sleep   59    0   0:01:11 0.0% webservd/76
 23319 sparvu   3416K 2872K cpu0    59    0   0:00:00 0.0% prstat/1
  1810 noaccess  177M   65M sleep   59    0   0:00:56 0.0% java/28
  3133 sparvu     49M   14M sleep   59    0   0:00:46 0.0% metacity/1
  3152 sparvu     51M   14M sleep   59    0   0:00:31 0.0% wnck-applet/1
 27399 sparvu     71M   36M sleep   37    4   0:00:13 0.0% gimp-2.0/1
  3156 sparvu     48M   11M sleep   59    0   0:00:25 0.0% mixer_applet2/1
  6983 sparvu   1204K  908K sleep   59    0   0:00:00 0.0% ksh/1
  3137 sparvu     56M   19M sleep   59    0   0:00:31 0.0% gnome-panel/1
  3111 sparvu   6052K 3036K sleep   59    0   0:00:06 0.0% xscreensaver/1
  3116 sparvu   8148K 3740K sleep   59    0   0:00:05 0.0% gnome-smproxy/1
   360 root     4660K 1848K sleep   59    0   0:00:00 0.0% automountd/2
   480 root     1736K  544K sleep   59    0   0:00:00 0.0% smcboot/1
   478 root     1740K  944K sleep   59    0   0:00:00 0.0% smcboot/1
 NPROC USERNAME  SIZE   RSS MEMORY      TIME  CPU
    57 sparvu   1491M  543M    27%   0:26:49  10%
    39 root      560M  307M    15%   1:08:55 3.8%
     1 webservd  142M   54M   2.6%   0:01:11 0.0%
     1 noaccess  177M   65M   3.2%   0:00:56 0.0%
     1 smmsp    6872K 1480K   0.1%   0:00:00 0.0%
     1 lp       2936K 1084K   0.1%   0:00:00 0.0%
     4 daemon     11M 6004K   0.3%   0:00:00 0.0%
Total: 104 processes, 371 lwps, 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

   PROCESS                                     COUNT
   gnome-vfs-daemon                                3
   httpd                                           3
   mixer_applet2                                   8
   xscreensaver                                    9
   gnome-netstatus-                               10
   intrd                                          15
   java                                           20
   gnome-panel                                    31
   mpstat                                         35
   tput                                           49
   webservd                                       49
   dtrace                                         62
   firefox-bin                                    84
   soffice.bin                                   108
   sh                                            122
   clear                                         136
   Xorg                                          628
   gnome-terminal                               2455
   ksh                                          9727
   date                                        32778
```

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  325    8   48    2    0   552    1   0   0  99
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   0    0   517  380  371    2   76    9    0   839    2   0   0  98
  1    0   0    0   152    5  406    4   69    7    0   817    2   1   0  97
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   0    4   506  384  279    6   52    4    0   306    1   0   0  99
  1    0   0    1   154   10  312    6   50    1    0   272    0   0   0 100
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   0    0   684  443  431   13   60   13    0   702   10   1   0  89
  1    0   0    0   288    7  714   19   63    2    0   906    5   1   0  94
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  171  93 5915  4832  736 2227  318  117  392    0 143341   13  37   0  50
  1  573  62 3507  7098    5 4971  648  128  247    0 54178   23  19   0  58
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   2 3089  4004  410 3263  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
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  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:

```
Tracing... Hit Ctrl-C to end.
^C
   PID CMD                              XCALLS
 11257 ksh                                   1
 11258 ksh                                   1
 11259 ksh                                   1
 11260 ksh                                   1
 11255 ksh                                   2
 11256 ksh                                   2
  2540 ksh                                   3
  2163 gnome-panel                           7
 11254 dtrace                               15
  1922 intrd                                27
  2372 mpstat                               27
     0 sched                                46
 11255 find                              27329
```

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*

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

4.Network Connections, cont.

- 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
```

4.Network Connections, cont.

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

```
UID    PID CMD          TYPE  PORT IP_SOURCE
  0    266 inetd         tcp    23 192.168.1.3
  0    422 sshd          tcp    22 192.168.1.3
 80   1984 webservd      tcp    80 192.168.1.3
  0    422 sshd          tcp    22 192.168.1.3
  0    422 sshd          tcp    22 192.168.1.3
  0    266 inetd         tcp    21 192.168.1.3
```

4.Network Connections, cont.

- **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 Utilisation

- 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 Utilisation, 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 Utilisation, 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
^C
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 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 5640 find 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
  1     71              bdev_strategy:start 5640 find 1024
```

5.Disk Utilisation, cont.

- Run *iotop*:

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

  UID    PID   PPID CMD              DEVICE  MAJ MIN D   %I/O
  100   1968      1 gconfd-2         cmdk0   102   7 W      0
    0    121      1 nscd             cmdk0   102   0 R      1
  100   5568      1 gnome-panel-scre cmdk0   102   1 R      1
  100   1968      1 gconfd-2         cmdk0   102   7 R      1
    0    392    386 Xorg             cmdk0   102   0 R      2
  100   5555   4816 bart             cmdk0   102   7 R     16
  100   5568      1 gnome-panel-scre cmdk0   102   7 R     21
  100   5568      1 gnome-panel-scre cmdk0   102   0 R     54
```

5.Disk Utilisation, cont.

- Run now *iosnoop*:

```
UID    PID D     BLOCK   SIZE       COMM PATHNAME
100   5603 R   3475216   8192       bart /opt/openoffice.org2.0/program/libres680si.so
100   5603 R   3475232   8192       bart /opt/openoffice.org2.0/program/libres680si.so
100   5603 R   3475248  16384       bart /opt/openoffice.org2.0/program/libres680si.so
100   5603 R   3462668   2048       bart /opt/openoffice.org2.0/program/libres680si.so
100   5603 R  56037520   8192       bart <none>
100   5603 R  56038128   8192       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038168   8192       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038192   4096       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038272   8192       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038296   4096       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038336  20480       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038392   8192       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038416  16384       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038528  45056       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038688  36864       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038792  53248       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038952   4096       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038968   4096       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56038984   4096       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56039040  57344       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56039152   4096       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56039224   4096       bart /opt/openoffice.org2.0/program/libsb680si.so
100   5603 R  56039288   8192       bart /opt/openoffice.org2.0/program/libsb680si.so
```

5.Disk Utilisation, cont.

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

```
 PID  CMD
5602  find /opt\0

       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 Utilisation, 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

           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@             914
               1 |                                         0
               2 |                                         18
               4 |@                                        26
               8 |@                                        26
              16 |@                                        26
              32 |                                         16
              64 |@                                        31
             128 |@                                        33
             256 |@                                        25
             512 |@                                        19
            1024 |                                         15
            2048 |@                                        28
            4096 |@                                        51
            8192 |@@@@@@                                   170
           16384 |                                         6
           32768 |                                         0
           65536 |                                         1
          131072 |                                         1
          262144 |                                         0
          524288 |                                         0
         1048576 |                                         2
         2097152 |                                         3
         4194304 |                                         2
         8388608 |                                         2
        16777216 |@                                        21
        33554432 |@                                        26
        67108864 |                                         0
```

5.Disk Utilisation, 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 Utilisation, cont.

- *rwtop* and *opensnoop*:

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

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

```
UID    PID COMM          FD PATH
  0    252 utmpd          5 /var/adm/utmpx
  0    252 utmpd          6 /var/adm/utmpx
  0    252 utmpd          7 /proc/1840/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
  0    252 utmpd          7 /proc/2150/psinfo
100   5685 find          -1 /var/ld/ld.config
100   5685 find           3 /lib/libsec.so.1
100   5685 find           3 /lib/libc.so.1
100   5685 find           3 /lib/libavl.so.1
100   1968 gconfd-2      44 /export/home/sparvu/.gconfd/saved_state.tmp
100   5687 bart          -1 /var/ld/ld.config
100   5687 bart           3 /lib/libsec.so.1
100   5687 bart           3 /lib/libmd5.so.1
100   5687 bart           3 /lib/libc.so.1
100   5687 bart           3 /lib/libavl.so.1
100   5686 find          -1 /var/ld/ld.config
100   5686 find           3 /lib/libsec.so.1
100   5686 find           3 /lib/libc.so.1
100   5686 find           3 /lib/libavl.so.1
100   5688 bart           3 /etc/default/init
100   5688 bart           3 /usr/share/lib/zoneinfo/Europe/Helsinki
100   5688 sort          -1 /var/ld/ld.config
100   5688 sort           3 /lib/libc.so.1
100   5688 sort           3 /proc/self/auxv
100   5688 sort          -1 /var/ld/64/ld.config
100   5688 sort           3 /lib/64/libc.so.1
100   5688 sort           3 /dev/null
100   5687 bart           3 /opt/sfw/lib/firefox/LICENSE
100   5687 bart           3 /opt/sfw/lib/firefox/README.txt
100   5687 bart           3 /opt/sfw/lib/firefox/browserconfig.properties
```

Agenda - Day 2

- The toolkit: DTraceToolkit
  - An introduction to DTraceToolkit
  - How to think and use the toolkit
  - Real Examples
- **DTrace and Java**
- DTrace Community
  - The DTT Team
  - OpenSolaris and DTrace books
- Future

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, hotspot_jni

DTrace and Java, cont.

- *jstack()*
- The simplest form to record a stack trace from a Java application
- Delivered already with DTrace framework:

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


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

DTrace and Java, cont.

```
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`__1cJJavaCallsLcall_helper6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v_+0x187
libjvm.so`__1cCosUos_exception_wrapper6FpFpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread__v2468_v_+0x14
libjvm.so`__1cJJavaCallsEcall6FpnJJavaValue_nMmethodHandle_pnRJavaCallArguments_pnGThread__v_+0x28
libjvm.so`__1cJJavaCallsMcall_virtual6FpnJJavaValue_nLKlassHandle_nMsymbolHandle_4pnRJavaCallArguments_pnGThread__v_+0xbe
libjvm.so`__1cJJavaCallsMcall_virtual6FpnJJavaValue_nGHandle_nLKlassHandle_nMsymbolHandle_5pnGThread__v_+0x6d
libjvm.so`__1cMthread_entry6FpnKJavaThread_pnGThread__v_+0xd0
libjvm.so`__1cKJavaThreadRthread_main_inner6M_v_+0x51
libjvm.so`__1cKJavaThreadDrun6M_v_+0x105
libjvm.so`__1cG_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
```

DTrace and Java, cont.

- VM Agents
  - Adding probes using a VM agent
  - Shared libraries which needs to be loaded when the VM starts
  - Java 1.4.2: using the dvmpi agent
  - Java 5: the dvmti agent
  - Download from: https://solaris10-dtrace-vm-agents.dev.java.net/

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: Initialisation and Shutdown

    Thread statistics Probes

    Class loading and unloading Probes

    Garbage Collection Probes

    Method Compilation Probes

Agenda - Day 2

- The toolkit: DTraceToolkit
  - An introduction to DTraceToolkit
  - How to think and use the toolkit
  - Real Examples
- DTrace and Java
- **DTrace Community**
  - The DTT Team
  - OpenSolaris and DTrace books
- Future

DTrace Community

- **DTT Team**
  - **Central Maintainer**

    Brendan Gregg, brendan.gregg@tpg.com.au
  - **Bug fixes**

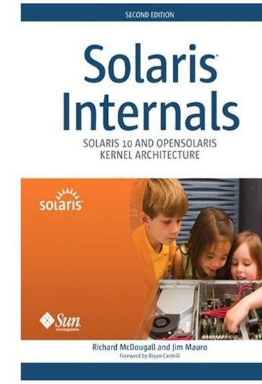
    Brendan Gregg, Stefan Parvu
  - **Testing and documentation**

    Brendan Gregg, James Dickens, Ryan Matteson, Stefan Parvu
  - **Ideas, technical advice**

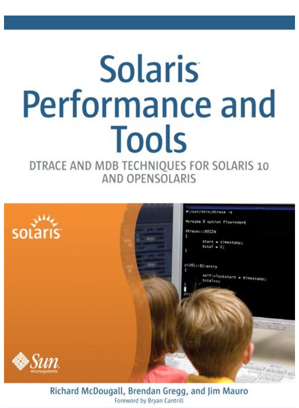
    Ben Rockwood, David Rubio, Nathan Kroenert
  - **Internals DTrace engineers**

    Adam Leventhal, Bryan Cantrill, Mike Shapiro

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

Agenda - Day 2

- The toolkit: DTraceToolkit
  - An introduction to DTraceToolkit
  - How to think and use the toolkit
  - Real Examples
- DTrace and Java
- DTrace Community
  - The DTT Team
  - OpenSolaris and DTrace books
- **Future**

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:
  - MacOSX Leopard
  - FreeBSD

Appendix, License

Public Documentation License Notice

開
放
的
열린
مفتوح
libre
मुक्त
ಮುಕ್ತ
livre
libero
ముక్త
开放的
açık
open
nyílt
פתוח
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

open

USE IMPROVE EVANGELIZE

Thank you!

Stefan Parvu
System Administrator
stefan.parvu@sun.com
stefanparvu14@yahoo.com
http://stefanparvu.blogspot.com

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