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

opensolaris

Eric Saxe & Stephen Lau
Solaris Kernel Development
Sun Microsystems, Inc.

OpenSolaris in Brief

- Launched June 14, 2005
- Sources & binaries (ISOs) available from: http://opensolaris.org
- OpenSolaris is **really Open Source**
  - **Licensed under CDDL & other Open Source licenses**
  - **Allows royalty-free use, modification, & derived works**
  - **OSI approved**
- **OpenSolaris is openly developed**
  - Non-Sun contributors of code, documentation, & more
  - Strategic & tactical collaboration on technology
  - 138 bugs filed, 54 bugfixes or new features submitted!

Why on earth...?

- It's the Right Thing To Do
  - OpenSolaris is increasing reach of Solaris into markets previously closed to us
  - OpenSolaris is increasing mindshare among you, today's students – you are tomorrow's IT professionals
  - OpenSolaris will allow ports of Solaris to new machine and CPU architectures
  - OpenSolaris is providing larger markets for our partners: ISVs, OEMs and solution providers

So... is it still Solaris?

- Yes!
- Codebase is Solaris “Nevada” (post-Solaris 10)
- Similar to Fedora/Red Hat relationship
- The technologies and code in OpenSolaris will make up the next version of Solaris
- Different OpenSolaris-based distributions will be free to use/discard whichever parts of OpenSolaris they want.

What's it got?

- 33,457 source files, 10.9 million lines of code
  - Most source under the CDDL, some source under other (pre-existing) Open Source licenses
  - Add-on redistributable binaries for which source isn't yet available: 523 files on x86
- ON (OS/Networking) consolidation:
  - Kernel, drivers, system libraries, commands, daemons
  - The good stuff: Dtrace, Zones, ZFS, SMF, FMA, & more
- JDS/GNOME (Java Desktop System)
- Xorg
- Network Storage

What's it getting?

- Everything Solaris is getting, and more
  - Moving from bi-weekly code syncs to nightly to realtime
- And even some stuff Solaris doesn't have yet!
  - Bleeding edge bits/projects
  - Xen, BrandZ, Wifi, etc.
- Better Open Source/freeware integration
  - Blastwave, SunFreeware, Companion CD
- Community projects/distributions:
  - Schilix, Belenix, Nexenta Debian GNU/OpenSolaris
  - OpenSolaris on PowerPC, DTrace for FreeBSD

So why should you care?

- <showoff>
  Our code is good. Really good.
  </showoff>

  * (okay, so some of it isn't... so help us make it better)
- Observability
  - Spend less time asking “wtf?” and more time coding
- Better than a textbook
- Direct access to the experts

Where and how?

- http://opensolaris.org
- 3 ways to play:
  - Live CD
  - Full Install
  - Source

Live CD: I'm just curious

- Nexenta
  - http://www.gnusolaris.org
- SchiliX
  - http://schillix.berlios.de
- Belenix
  - http://belenix.sarovar.org/belenix_home.html

Full install: I'm ready to jump in!

- Solaris Express: Community Release (SXCR)
  - http://opensolaris.org/os/downloads
  - You'll be running what we're running
  - Solaris “Nevada”, the full thing
- Nexenta
  - http://www.gnusolaris.org

Source: I'm hardcore.

- First: install a recent SXCR
- Download 5 components from http://opensolaris.org/os/downloads
  - opensolaris-src-2005XXXX.tar.bz2
  - opensolaris-closed-bins-2005XXXX.tar.bz2
  - SUNWonbld-2005XXXX.tar.bz2
  - opensolaris-build-extras-2005XXXX.tar.bz2
  - Compiler (Sun Studio 10) ... it's free!

Preparing your build environment

- Unpack the compiler (/opt/SUNWspro)
- Unpack & install the build tools
  - Download the SUNWonbld to /tmp

```
bunzip2 -c SUNWonbld-2005XXXX.i386.tar.bz2 | tar xf -
su -
pkgadd -d onbld SUNWonbld
```

- On a SPARC machine?

```
cd /opt
bunzip2 -c opensolaris-build-extras.tar.bz2 | tar xf -
```

Getting the source ready...

- Unpack the source

```
mkdir /export/opensolaris/b27
cd /export/opensolaris/b27
bunzip2 -c opensolaris-src.tar.bz2 | tar xf -
bunzip2 -c opensolaris-closed-bins.tar.bz2 | tar xf -
cp usr/src/tools/env/opensolaris.sh .
```

- Edit opensolaris.sh
  - Change `GATE` to be the top-level dir (e.g.: b27)
  - Change `CODEMGR_WS` to top-level path (e.g.: /export/opensolaris)
  - Change `STAFFER` to be your login

Building the bits!

- To build the entire ON tree:

```
/opt/onbld/bin/nightly ./opensolaris.sh
```

- To build an individual component:

```
/opt/onbld/bin/bldenv -d ./opensolaris.sh
```

```
cd usr/src/cmd/passwd
```

  > edit passwd.c to mail you people's passwords when they change them

```
dmake all
```

- To build the kernel:

```
cd usr/src/uts
```

```
dmake all
```

Booting the bits!

- If you've changed only the kernel

```
cd usr/src/uts

dmake install

Install -G kernel.myname -k i86pc
```

  - copy tar file to test system, untar in /, and

```
reboot -- kernel.myname/unix -k
```

  - Away you go... (hopefully)
- If you've changed libraries
  - BFU (aka *Blindingly Fast Upgrade, Big F******g Upgrade, Bonwick-Faulkner Upgrade*)
  - Ensures all bits being booted and used are self-consistent

Where to go for help?

- Stuck on the build? (you read the ReleaseNotes, right?)
  - Ask on opensolaris-help@opensolaris.org
- Questions about the code or found a bug?
  - Ask on opensolaris-code@opensolaris.org
- Got a fix for a bug?
  - Request a sponsor on request-sponsor@opensolaris.org
- Got random questions?
  - irc.freenode.net channel #opensolaris
- Solaris is teh suxor! Linux rulez!
  - Flame and debate on opensolaris-discuss@opensolaris.org

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

opensolaris™

Eric Saxe (esaxe@eng.sun.com)
Stephen Lau (stevel@sun.com)
Solaris Kernel Development

DTrace Overview

Eric Saxe, Stephen Lau
Solaris Kernel Development
Sun Microsystems

Why Dynamic Tracing?

- Well-defined techniques for debugging *fatal*, *non-reproducible* failure:
  - Obtain core file or crash dump
  - Debug problem *postmortem* using mdb(1), dbx(1)
- Techniques for debugging *transient* failures are much more ad hoc
  - Typical techniques push traditional tools (e.g. truss(1), mdb(1)) beyond their design centers
  - Many transient problems cannot be debugged at all using extant techniques

Transient failure

- Any unacceptable behavior that does not result in fatal failure of the system
- May be a clear failure:
  - “read(2) is returning EIO on a device that isn't reporting any errors.”
  - “One of our threads is missing a condition variable wakeup.”
- Or more symptomatic:
  - “We were expecting to accommodate 100 users per CPU, and we're able to get no more than 60.”
  - “Every morning from about 9am to about 10:30am, the system is a dog.”

Debugging transient failure

- Historically, we have debugged transient failure using process-centric tools: truss(1), pstack(1), prstat(1), etc.
- These tools were not designed to debug *systemic* problems
- But the tools designed for systemic problems (i.e., mdb(1)) are designed for postmortem analysis...

Postmortem techniques

- One technique is to use postmortem analysis to debug transient problems by *inducing* fatal failure during period of transient failure
- Better than nothing, but not by much:
  - Requires inducing fatal failure, which nearly always results in more downtime than the transient failure
  - Requires a keen intuition to be able to suss out a dynamic problem from a static snapshot of state

Invasive techniques

- When existing tools cannot be used to root-cause transient failure, more invasive techniques must be used.
- Typically, custom instrumentation is developed for the failing program or kernel.
- The problem must then be reproduced using the instrumented binaries.

Invasive techniques, cont.

- Requires either:
  - running instrumented binaries in production

  or
  - reproducing a transient problem in a development environment
- Neither of these is desirable!
- Invasive techniques are slow, error prone, and often ineffective. We must develop a better way...

Dynamic instrumentation

- Want to be able to *dynamically* modify a running system to record *arbitrary* data
- Must have zero probe effect when not used
- Must be able to do this on *production* systems.
- *Must be* completely *safe* – there should be no way to induce fatal failure.

Introducing DTrace

- Dynamic tracing framework introduced in Solaris 10. Available in OpenSolaris.
- Available on stock systems
- Utilizes a dynamically interpreted language allowing arbitrary actions and predicates
- Can instrument all levels of the system, (user applications and the kernel).

Probes and Providers

- DTrace allows you to add instrumentation wherever *probes* can be enabled.
- Probes are made available to the DTrace framework by *providers*.
- A provider represents a methodology for instrumenting the system in a particular way.

Providers

- DTrace has quite a few built in providers, e.g.:
  - The *function boundary tracing (FBT)* provider can dynamically instrument every function entry and return in the kernel
  - The *pid* provider can dynamically instrument user processes, down to instruction level granularity.
  - The *syscall* provider can dynamically instrument the system call table
  - The *lockstat* provider can dynamically instrument the kernel synchronization primitives
  - The *profile* provider can add a configureable-rate profile interrupt of to the system
  - ... and more.

Invoking DTrace

- A DTrace invocation consists of:
  - specifying a set of *probes* to enable
  - for each probe, specifying an *action* to take when the probe fires
  - for each probe, optionally specifying a *predicate,* which allows a probe's action to be taken only when certain conditions are met
- Actions and predicates are expressed in the D programming language.

Invoking DTrace

- As a simple example, tracing the pid of every process named “date” that does an open(2):

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

syscall::open:entry
/execname == “date”/
{
        trace(pid);
}
```

Probe name

- Entry point of open(2)

Predicate

- Process is named “date”

Action

- Print the process ID

Another Example

- Trace all kernel functions calls (and returns) once we hit our open(2) probe:

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

syscall::open:entry
/execname == “date”/
{
        self->traceme = 1;
}
fbt:::
/self->traceme == 1/
{
        trace(timestamp);
}
syscall::open:return
/self->traceme == 1/
{
        self->traceme = 0;
}
```

Set thread local variable “traceme” if thread's process is named “date”.

At all function entry/exit points, trace a timestamp if thread local variable “traceme” is set

Clear thread local variable “traceme” if currently set

Data Management Primitives

- DTrace provides functions for aggregating data, to assist in identifying larger trends.
- For example, to count all systems calls, and aggregate by process name, and system call name:

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

syscall:::
{
        @foo[execname, probefunc] = count();
}
```

Data Management Primitives

- Data may also be quantized allowing for distribution analysis
- Measure and quantize the amount of time spent in malloc(3c):

```
#!/usr/sbin/dtrace -s
pid$1:libc.so.1:malloc:entry
{
        self->s = vtimestamp;
}
pid$1:libc.so.1:malloc:return
/self->s/
{
        @foo["malloc times"]=quantize(vtimestamp - self->s);
        self->s = 0;
}
```

Data Management Primitives

- For mozilla-bin:

```
# ./ex6.d `pgrep mozilla-bin`
dtrace: script './ex6.d' matched 2 probes
^C

  malloc times
           value  ------------- Distribution ------------- count
             256 |                                         0
             512 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@    163276
            1024 |@@@                                      12310
            2048 |                                         1352
            4096 |                                         456
            8192 |                                         89
           16384 |                                         39
           32768 |                                         25
           65536 |                                         2
          131072 |                                         0
```

Exploring DTrace

- This presentation has not even scratched the surface!
- Other places to look to learn more:
  - The BigAdmin DTrace discussion forum:
    http://www.sun.com/bigadmin/content/dtrace
    There you will find the Solaris Dynamic Tracing Guide available...over 400 pages on DTrace!
  - Many example DTrace scripts can be found in
    /usr/demo/dtrace/
- Please join us in the DTrace community at:
  http://opensolaris.org/os/community/dtrace

The DTrace Revolution

- DTrace tightens the diagnosis loop: *hypothesis* → *instrumentation* → *data gathering* → *analysis* → *hypothesis*
- Tightened loop effects a revolution in the way we diagnose transient failure
- Focus can shift from *instrumentation* stage to *hypothesis* stage:
  - Much *less* labor intensive, less error prone
  - Much *more* brain intensive
  - *Much* more effective! (And a *lot* more fun)

DTrace Overview

Eric Saxe <eric.saxe@sun.com>
Stephen Lau <stevel@sun.com>