OpenSolaris Troubleshooting
The Unofficial Tourist Guide

Peter Harvey

Solaris RPE

Introduction

To misquote LP Hartly

> *“Diagnosing problems on unfamiliar operating systems is a foreign country: they do things differently there.”*

- What this talk is:
  - Getting started troubleshooting on OpenSolaris
  - Sharing tools and techniques used inside Sun
  - Thinking
- What this talk is not:
  - Crash dump analysis
  - Performance tuning
  - A DTrace tutorial

Overview

- Principle of observability
- Be prepared
- Problem types
- OpenSolaris/UNIX tools
- Using OpenSolaris source
- Example

Theory

Practise

Principle of Observability

- Popularised in the early days of DTrace
- States the importance of being able to observe just what something is doing:

> *"It is thus incumbent upon providers of software layers to further provide infrastructure to dynamically observe these layers and their interactions – only through observability can we mitigate the complexity inherent in the layering of software." -- Bryan Cantrill, August 2002*

Observability in practise

- Built-in observability
  - > Verbose/debug options (eg rpc.nisd -v)
  - > Environment variables (eg NIS_OPTIONS)
  - > Dedicated tools (eg nfsstat)
- Retrofit observability
  - > truss, proc tools, debuggers, DTrace
  - > Interpose libraries (eg libumem)
- Retrofit vs built-in
  - > Consistent interface but no intimate knowledge

Observability isn't everything

- Observability provides data, it doesn't provide the answer nor helps you ask the right question
- Observability is in the eye of the beholder, understanding varies
  - > eg I can't read an ECG, but a cardiologist can
- Appropriate observation is all
  - > eg Doctors usually ask what the problem is first and examine later if necessary

Observability is not a panacea

- **pan**·a·ce·a - A remedy for all diseases, evils, or difficulties; a cure-all
- Brian Wong wrote (re bugs closed as “DTrace should be able to provide this information”):

> *“DTrace should be able to provide this information, but has anyone verified that it can? Furthermore, this issue is not resolved until an end user can discover this information without opening the DTrace manual. A suitable resolution would be to provide a DTrace script, with man page, and pointers to the script from existing man pages and manuals, such as vmstat(1), iostat(1) and other administrative manuals.”*

Be prepared

- Make sure sufficient data is captured
  - Configure syslogd(1M) to capture debug output
    - Add “*.debug /var/log/syslog.debug” to /etc/syslog.conf
      - Remember to use TAB for whitespace
    - Touch /var/log/syslog.debug
    - Send HUP to syslogd (# svcadm refresh system-log)
  - Configure collection of core files
    - # mkdir /var/cores
    - # coreadm -e <everything>
    - # coreadm -g /var/cores/core.%u.%f.%t.%z
  - Configure crash dumps (dumpadm(1M))
- Keep a change log

Problem types

- Many different types
  - Unexpected error messages
  - Application or system stops responding
  - Application or system won't start
  - Core dumps, crash dumps
  - Exits without errors or cores
  - Unexpected behaviour
  - Performance
- Different problems, same questions
  - What? Where? When?
  - Be clear about the problem first

Typical process problems

- If a process stops responding
  - Check whether it has stopped or is spinning
    - Use prstat (1M), truss(1)
  - For a process that has stopped, check pstack(1)
  - For a process that's spinning
    - truss(1) for a while
    - Periodically run pstack(1)
      - Look for common functions in the stack
- Core dumps
  - pstack(1), use libumem(3LIB) if in the memory allocator
- Exits without error
  - truss -t \!all -T exit

Special note on hangs

- First aid principles apply – Check level of response!
- People
  - > “Annie, can you hear me Annie?”
  - > “Annie, open your eyes!”
  - > Gentle shake of shoulders, “Annie!?”
- Systems
  - > Responds to ping? Remote login?
  - > Console access?
  - > Stop-A (SPARC), F1-A (x86), Console break, percussive maintenance :-)

OpenSolaris/UNIX tools

- OpenSolaris
  - truss(1)
  - proc(1) tools
  - snoop(1M)
  - dtrace(1M)
  - dbx(1)/mdb(1)
  - memory allocators
    - libumem(3lib)
    - watchmalloc(3malloc)
- UNIX generic
  - Wireshark (ethereal)
  - lsof
  - Command line
    - eg grep, cut, sort, sed, awk, perl
  - Editors
    - eg Xemacs

truss(1)

- Everyone has their favourite options
  - Mine is `-aef -rall -wall -vall`
  - Timestamps are often useful (-d)
- Can attach to running programs
- Can trace user level functions (-u)
- Can stop the process on specific conditions
  - System calls (-S), signals (-T), user-level functions (-U)
  - Use prun(1) to continue
- Alternatives
  - apptrace(1), sotruss(1) <- actually a script, take a look!

Anatomy of truss(1M) output

```
$ truss -tstat ls -ld .zshrc
...
stat64(".zshrc", 0xFFBFD9E0)                    = 0
-rw-r--r--   1 peteh    staff        392 Apr 18 13:17 .zshrc

$ truss -vall -tstat ls -ld .zshrc
...
stat64(".zshrc", 0xFFBFD9E0)                    = 0
    d=0x05B80E8A i=2557576 m=0100644 l=1  u=27324 g=10    sz=392
        at = Jun 19 12:45:18 BST 2007  [ 1182253518 ]
        mt = Apr 18 13:17:10 BST 2007  [ 1176898630 ]
        ct = Apr 18 13:17:10 BST 2007  [ 1176898630 ]
    bsz=8192  blks=2     fs=nfs

$ truss -xall -tstat ls -ld .zshrc
stat64(0xFFBFEEF7, 0xFFBFD9E0)                  = 0
     0xFFBFEEF7: ".zshrc"
```

- From the truss(1) man page: -x [!]syscall, ...
  - “This is for unredeemed hackers who must see the raw bits to be happy.”

proc(1) tools - *wonderful*

- Too many to list, my favourites
  - pstack (where are you?)
    - Use dem(1) or c++filt(1) to demangle names
  - ptree (what are you related to?)
  - pargs (use '-e' to see the environment)
  - pgrep (use -fl for better matching, more output)
  - pldd (which libraries? are they as expected?)
  - pfiles (what open files?)
- Can be used on running process and sometimes core files too

Anatomy of pfiles(1) output

```
# pfiles 1420
1420:   /usr/lib/ssh/sshd
  Current rlimit: 256 file descriptors
   0: S_IFCHR mode:0666 dev:302,0 ino:6815752 uid:0 gid:3 rdev:13,2
      O_RDWR|O_LARGEFILE
      /devices/pseudo/mm@0:null
   1: S_IFCHR mode:0666 dev:302,0 ino:6815752 uid:0 gid:3 rdev:13,2
      O_RDWR|O_LARGEFILE
      /devices/pseudo/mm@0:null
   2: S_IFCHR mode:0666 dev:302,0 ino:6815752 uid:0 gid:3 rdev:13,2
      O_RDWR|O_LARGEFILE
      /devices/pseudo/mm@0:null
...
   6: S_IFSOCK mode:0666 dev:308,0 ino:40260 uid:0 gid:0 size:0
      O_RDWR|O_NONBLOCK
        SOCK_STREAM
        SO_REUSEADDR,SO_KEEPALIVE,SO_SNDBUF(49152),
        SO_RCVBUF(49640),IP_NEXTHOP(0.0.193.232)
        sockname: AF_INET6 ::ffff:129.156.210.4  port: 22
        peername: AF_INET6 ::ffff:129.156.173.17  port: 40473
   7: S_IFDOOR mode:0644 dev:311,0 ino:51 uid:0 gid:0 size:0
      O_RDONLY FD_CLOEXEC  door to keyserv[735]
      /var/run/rpc_door/rpc_100029.2
...
#
```

snoop(1M)

- Captures network traffic
- Can be configured to capture just relevant data
- Currently captures single interfaces
  - > Has to be run multiple times
  - > For client/server problems run it both ends as well
- Capture binary not text output
  - > Text can be extracted later
- The capture files can be split/merged
  - > split: snoop -i infile -o outfile expression
  - > merge: mergecap (part of SFW)

dbx(1) and mdb(1)

- dbx
  - Part of Sun's Studio developer tools
  - Download available from OpenSolaris
    - http://opensolaris.org/os/community/tools/sun_studio_tools/
  - Very handy for MT programs
  - Less handy unless '-g' used for compilation
- mdb
  - Superb, provided you know what you're doing
  - Needed for libumem (eg ::findleaks)

dbx(1), mdb(1) and libraries

- Library matching used to be essential when debugging cores from other systems
- Fixed in Solaris 10 (bug 4884589) and later
  - Core files now include all text, see coreadm(1M)
- For those debugging pre-Solaris 10
  - dbx extended *pathmap* in Workshop 6.0u2
    - See: help core mismatch
  - Scripted example: pkg_app
    - http://www.sun.com/bigadmin/scripts/indexSjs.html

Memory allocators

- libumem(3lib)
  - Can be used with existing binaries
  - Resolves memory problems in hours, not weeks
    - Memory leaks (::findleaks in mdb)
    - Memory mangling (see umem_debug(3malloc))
  - Typically performs better than standard malloc(3c)
  - Developed from the kernel slab allocator
- watchmalloc(3malloc)
  - Useful for catching things libumem doesn't as it can set watchpoints to protect memory
    - Actually, libumem can do this, be we don't document it!
    - http://blogs.sun.com/peteh/entry/hidden_features_of_libumem_firewalls

Text tools

- Often forgotten!
- Using standard UNIX text tools
  - grep, cut, sort, sed, awk, perl, Xemacs, ...
- Very handy for processing large files
  - messages and other log files
  - ASCII snoop output
  - correlating data from a series of files
  - extracting data for graphing
    - xgraph, xplot, StarOffice
- Incremental regexp in Xemacs is great

Using debug options in SMF

- Classic UNIX
  - Restart daemon foo with '-v'
- OpenSolaris has SMF - Some suggestions ...
  - Disable the service and run the command directly
    - # svcadm disable -t fooservice
    - Run fooservice with debug options
    - # svcadm enable fooservice when you're done
  - Modify the service method (see /lib/svc/method)
  - Modify the service manifest
    - disable fooservice, copy the manifest and edit the methods
    - import the modified fooservice, enable fooservice

Setting environment variables in SMF

- Many services can be debugged using environment variables
- svccfg(1M) can set them
- Example:
  - > # svccfg -s system/service setenv LD_PRELOAD libumem.so
  - > # svccfg -s system/service setenv UMEM_DEBUG default

Using OpenSolaris

- Examining the source code
  - OpenGrok
    - http://src.opensolaris.org/source/
  - cscope
    - Part of Sun Studio tools
    - Modified version in ON – usr/src/tools/cscope-fast
      - Creates reverse indexes with the '-q' option
      - Much faster than vanilla cscope
    - Build the indexes using the xref command
    - Integrated with editors
      - vim
      - xemacs
    - http://cscope.sourceforge.net/

From core to source

- Get a stack trace
  - Don't assume function arguments are valid
    - SPARC 'in' registers get reused
    - x86 (32-bit) is usually reliable as arguments are passed on the stack
    - x64 (64-bit) uses registers by default, arguments are not saved
      - mdb doesn't show any arguments at all
      - Compiler option to save arguments:
        - Sun (-save_args)
        - gcc (-msave_args)
- Start somewhere familiar

Example: bug 6562696, rpcbind core

- Started with pstack:

```
core 'core.rpcbind.228.1179802250' of 228:      /usr/sbin/rpcbind
 ff3d1b60 elf_bndr (0, 2a0, 1480c, 9, 4, 4) + c
 ff3b839c elf_rtbndr (1480c, 0, 0, 41, 0, ff32f948) + 10
 0002c008 ???????? (3, ffbfdc14, 2d360, 2d000, 0, 0)
 0001480c rpcbproc_callit_com (2dab8, 2d9a0, 5, 2, 13000, 134) + 10
 ff2cfa9c _svc_prog_dispatch (2d9a0, 2da80, 2dab8, 2, 13288, 2) + 184
 ff2cf878 svc_getreq_common (2d9a0, c, 1, ff327014, ff32f380, 2dab8) + e4
 ff2cf774 svc_getreq_poll (ffbfddc0, 1, ffffffff, 536bc, ff1619dc,
ff322da4) + 9c
 00015b7c my_svc_run (1, 5, ffffffff, 2d604, 2d5ec, ffbfddc0) + 128
 00017320 main     (1a000, 1a000, 1a000, 2dca0, 2328, 2d400) + 348
 000129ac _start   (0, 0, 0, 0, 0, 0) + 108
```

- Digression – check that time stamp

```
$ echo "0t1179802250=Y" | mdb
                2007 May 22 03:50:50
$
```

Example: bug 6562696, rpcbind core

- Same thing, but with mdb
  > Library modules make it clearer

```
$ mdb /usr/sbin/rpcbind ./core.rpcbind.228.1179802250
Loading modules: [ libc.so.1 libuutil.so.1 ld.so.1 ]
> $c
ld.so.1`elf_bndr+0xc(0, 2a0, 1480c, 9, 4, 4)
ld.so.1`elf_rtbndr+0x10(1480c, 0, 0, 41, 0, ff32f948)
0x2c008(3, ffbfdc14, 2d360, 2d000, 0, 0)
rpcbproc_callit_com+0x10(2dab8, 2d9a0, 5, 2, 13000, 134)
libnsl.so.1`_svc_prog_dispatch+0x184(2d9a0, 2da80, 2dab8, 2, 13288, 2)
libnsl.so.1`svc_getreq_common+0xe4(2d9a0, c, 1, ff327014, ff32f380, 2dab8)
libnsl.so.1`svc_getreq_poll+0x9c(ffbfddc0, 1, ffffffff, 536bc, ff1619dc,
ff322da4)
my_svc_run+0x128(1, 5, ffffffff, 2d604, 2d5ec, ffbfddc0)
main+0x348(1a000, 1a000, 1a000, 2dca0, 2328, 2d400)
_start+0x108(0, 0, 0, 0, 0, 0)
>
```

Example: bug 6562696, rpcbind core

- What is rpcbproc_callit_com doing??

```
> rpcbproc_callit_com+0x10::dis
rpcbproc_callit_com:                sethi     %hi(0x20000), %g1
rpcbproc_callit_com+4:              xor       %g1, -0x118, %g1
rpcbproc_callit_com+8:              save      %sp, %g1, %sp
rpcbproc_callit_com+0xc:            ld        [%i1], %o0
rpcbproc_callit_com+0x10:           call      +0x17a98      <PLT:t_getinfo>
rpcbproc_callit_com+0x14:           add       %fp, -0x1c, %o1
> t_getinfo::dis
libnsl.so.1`t_getinfo:              mov       %o7, %g1
libnsl.so.1`t_getinfo+4:            mov       1, %o2
libnsl.so.1`t_getinfo+8:            call      -0x60ac
<libnsl.so.1`_tx_getinfo>
libnsl.so.1`t_getinfo+0xc:          mov       %g1, %o7
> rpcbproc_callit_com+0x10+0x17a98=X
                2c2a4
> 2c2a4::dis
PLT:t_getinfo:                      sethi     %hi(0xa8000), %g1
PLT:t_getinfo:                      ba,a      -0x2a4        <0x2c004>
PLT:t_getinfo:                      nop
>
```

Example: bug 6562696, rpcbind core

- cscope and vim demo

Example: bug 6562696, rpcbind core

- Where is this taking us?

```
> 0x2c004::dis
0x2c000:                        unimp     0x2c6e0
0x2c004:                        save      %sp, -0x40, %sp
0x2c008:                        call      -0xc73c7c  <ld.so.1`elf_rtbndr>
0x2c00c:                        nop
> ld.so.1`elf_rtbndr::dis
ld.so.1`elf_rtbndr:             mov       %i7, %o0
ld.so.1`elf_rtbndr+4:           save      %sp, -0x60, %sp
ld.so.1`elf_rtbndr+8:           srl       %g1, 0xa, %o1
ld.so.1`elf_rtbndr+0xc:         ld        [%i7 + 8], %o0
ld.so.1`elf_rtbndr+0x10:        call      +0x197b8     <ld.so.1`elf_bndr>
ld.so.1`elf_rtbndr+0x14:        mov       %i0, %o2
> ld.so.1`elf_bndr::dis -n 10
ld.so.1`elf_bndr:               save      %sp, -0x98, %sp
ld.so.1`elf_bndr+4:             mov       %i0, %l1
ld.so.1`elf_bndr+8:             mov       %o7, %i0
ld.so.1`elf_bndr+0xc:           clr       [%fp - 4]
>
```

Example: bug 6562696, rpcbind core

- How did we fail to write to the frame pointer?

```
ld.so.1`elf_bndr+0xc:           clr       [%fp - 4]

> <fp/X
mdb: failed to read data from target: no mapping for address
0xffbdda78:
> $m
    BASE    LIMIT     SIZE NAME
   10000    1c000     c000 /usr/sbin/rpcbind
   2c000    2e000     2000 /usr/sbin/rpcbind
   2e000    b2000    84000 [ heap ]
...
ff3f6000 ff3f8000     2000 /lib/ld.so.1
ffbfc000 ffc00000     4000 [ stack ]
>
```

Example: bug 6562696, rpcbind core

- So what happened to the stack?

```
May 22 03:50:44 oaf350 genunix: [ID 470503 kern.warning]
  WARNING: Sorry, no swap space to grow stack for pid 228 (rpcbind)
```

- Digression ... what's that syslog ID?
  - > Message digest of the syslog string
  - > Useful to accurately identify strings in source
  - > See msgid(1M)

```
$ strings /platform/sun4u/kernel/sparcv9/genunix | msgid | grep 470503
470503 Sorry, no swap space to grow stack for pid %d (%s)
$
```

Example: bug 6562696, rpcbind core

- And one more thing ... how did we get a full stack trace?
  - > The stack couldn't be grown
  - > Where did mdb get the stack from?
- SPARC feature – gwindows
  - > See proc(4)
  - > Used to store registers when the stack can't be used

Crash dump analysis

- References
  - Solaris Modular Debugger Guide
    - http://docs.sun.com/
  - Frank Hofmann's “Solaris on x86 Platforms”
    - http://opensolaris.org/os/community/documentation/doc_index/dev/
  - PANIC! UNIX System Crash Dump Analysis Handbook
    - ISBN 0131493868
  - Solaris Internals
    - ISBN 0131482092
  - SOLARIS Systems Programming
    - ISBN 0201750392
  - SPARC Architecture Manuals

Performance tuning

- References
  - Solaris Performance and Tools
    - ISBN 0131568191
  - OpenSolaris Performance Community
  - http://blogs.sun.com/
- Tools
  - Far too many!
  - My usual favourites
    - vmstat(1M)
    - dtrace(1M)
    - lockstat(1M)

DTrace tutorial

- References
  - > OpenSolaris DTrace Community
  - > Solaris Performance and Tools (again)
  - > Future LOSUG talk? (hint)
- How I use DTrace
  - > Observing internals on live systems (lab and production)
  - > Data collection (inside and outside Sun)
  - > Iteratively
  - > With the manual open :-)
  - > With Google

Troubleshooting at Sun

- Sun has integrated Kepner-Tregoe processes across its service organisation and beyond
  - > We call it “Sun Global Resolution” - SGR
- SGR asks common sense questions
  - > WHAT is having the problem? WHAT problem?
  - > WHERE do you see the problem?
  - > WHEN do you see it?
  - > What is the EXTENT of the problem
- Reference “The New Rational Manager”
  - > ISBN 08715627

SGR – why do I need a process?

- Common approach, common language, consistency
- Good troubleshooters do this instinctively
  - Common themes
    - Separate and clarify the issues
    - Prioritise
    - Describe the problem accurately
    - Understand the boundaries of the problem
    - Develop possible causes
    - Consider the optimal way of verifying/fixing
- Software systems are harder to observe
  - Even more important to be methodical

OpenSolaris Troubleshooting

The Unofficial Tourist Guide

Peter.Harvey@Sun.COM

http://blogs.sun.com/peteh