Debugging System Hangs on Solaris

Oliver Yang

Jun 25, 2007

Sun Microsystem

http://oliveryang.net

Agenda

- What is a system hang?
- Debugging system hangs with kmdb/mdb
- Case Analysis
- References
- Appendix

Agenda

- What is a system hang?

What is a system hang?

- System...
  - has no response
  - is no longer usable

- What conditions cause hangs?
  - Deadlock
  - Resources exhaustion
  - Hardware problems

When system hangs happen...

- What you should do?
  - > Try a ping - reachable?
  - > Access network services - ssh/rsh/telnet... ?
  - > See console messages - any err/warn/fail ?
  - > System console status - no response?
  - > Try to force a system crash dump
  - > Check system logs and test journals after system boot

Agenda

- What is a system hang?
- **Debugging system hangs with kmdb/mdb**
  - > Loading kmdb
  - > Forcing a crash dump
  - > Live debugging
  - > Crash dump analysis
  - > Other crash dump analysis tools

Loading kmdb

- Boot-time Loading
  - x86 - grub

    ```
    kernel /platform/i86pc/multiboot -k
    ```

  - SPARC - OBP

    ```
    boot -k
    ```

- Runtime Loading
  - mdb -K

Forcing a crash dump

- General hangs
  - > Drop into kmdb or OBP

    Keyboard - Stop+A or F1+A

    Remote Console - Send a BREAK
  - > $<systemdump (all platforms, if kmdb loaded)
  - > sync (OBP, SPARC only)
- Hard hangs
  - > You can't enter kmdb when running into hard hangs
  - > Enable deadman timer in /etc/system

    set snooping=1

Deadman panic

- Why panic?
  - Setting in /etc/system file

    set snooping=1
  - Deadman timer will trigger a panic if clock interrupt was inactive about 5000 ticks (1tick=10ms)
  - We can change the default timer to other vaules(eg, 90s)

    set snoop_interval = 90000000
- What we should do?
  - Find out why clock interrupt become inactive

Live debugging

- For special circumstance...
  - Can't save crash dump
  - System hangs occurred during system boot.
- System hang happened during boot
  - Boot kmdb with the -kd options
  - Set necessary variables for debugging:

    ```
    moddebug/W 0x80000000
    snooping/W 0x1
    kmem_flags/W 0xf
    ```
  - Set break point

    Using fullly qualified symbol name - ::bp bge`bge_attach

Crash dump analysis

- Using mdb/kmdb and reading relevant source code
  - To identify the set of kernel threads in deadlock
  - To investigate how system hangs took place
- Checking crash dump files...
  - System status checking
  - Kernel threads checking
    - CPU and dispatch queue
    - User processes and kernel threads status
    - Check the stack trace of suspicious threads:

      - Function name related to mutex(9F)/rwlock(9F)/condvar(9F)/semaphore(9F)/biowait(9F)

      - Running into an infinite loop

mdb - frequently used ::dcmds

- System status checking
  - System messages - ::msgbuf
  - Clock interrupt - ::cycinfo
  - Physical memory - ::memstat
  - Cache/vmem allocation - ::kmastat
  - Checking any necessary global variables

    kmem_flags/X

    snooping/X

    ...

mdb - frequently used ::dcmds

- Kernel threads checking
  - ::cpuinfo
  - ::threadlist
  - ::thread
  - ::findstack
  - ::mutex
  - ::rwlock
  - ::wchaninfo
  - ::whatthread(Nevada only) or ::kgrep

Other crash dump analysis tools

- ACT - Automated Crash Tool
  - > A complete list of threads with function arguments
  - > Detailed system setting and resource summary
  - > Deadlock detection - mutex and rwlocks only
  - > Threads blocked in either getblk() or biowait()
- SCAT - Solaris Crash Analysis Tool
- Download from http://sunsolve.sun.com

Agenda

- What is a system hang?
- Debugging system hangs with kmdb/mdb
- **Case Analysis**
  - > Reverse locking order
  - > Infinite loop
  - > The constraints of current context

Reverse locking order

- Multiple threads deadlocks on multiple locks
  - To avoid the dead lock
    - Must always lock in the same order
    - Must always release in reverse order of locking
  - Two threads acquiring two locks with reverse order
    - Thread 1 ---> acquire Lock A ---> sleep and wait for Lock B
    - Thread 2 ---> acquire Lock B ---> sleep and wait for Lock A
  - See an example:
    - http://blog.ccw.com.cn/blog-htm-do-showone-itemid-12139-type-blog.html

Infinite loop - kmdb debugging

- Threads running into an infinite loop

```
>::cpuinfo -v
 ID ADDR       FLG NRUN BSPL PRI RNRN KRNRN SWITCH THREAD      PROC
  0 0000180c000   1d    0      0    0     yes      no    t-301817  30002e6e380  ifconfig
                       |
      RUNNING <--+
     QUIESCED
      EXISTS
      ENABLE
> 30002e6e380::findstack -v
stack pointer for thread 30002e6e380: 2a101d04391
 000002a101d04431 i_mod_hash_find_nosync+0x34(3000090bb40, 600042e5b58, 2a101d04e40, 1, 3000090bbb8, 98)
 000002a101d044e1 mod_hash_find+0x18(3000090bb40, 600042e5b58, 2a101d04e40, 53 , 30002e6e384, 0)
 000002a101d04591 mac_open+0xf8(600042e5b58, 1, 600042e5c60, 70051800, 70057c00 , 0)
 000002a101d04751 dls_mac_hold+0x24(600042e5b58, 2a101d05350, 2a101d051c8, 1, 600042e5cc0, 0)
```

Infinite loop - look into the code

- **The infinite loop between L239 and L250**

```
196 int
197 mac_open(const char *macname, uint_t ddi_instance, mac_handle_t *mhp)
198 {
...
239 again:
240         rw_enter(&i_mac_impl_lock,  RW_WRITER);
...
248         if (mip->mi_destroying) {
249                 rw_exit(&i_mac_impl_lock);
250                 goto again;
251         }
```

The constraints of current context

- Know the constraints of current context...
  - Learn the constraints of current context. For example, high-level interrupt, interrupts threads, soft-interrupts, timeout(9F), etc...
  - The lock usage should follow the constraints of current context, sometimes you have to use taskq(9F) when you can't block in current context
- The constraints of interrupt context
  - Only spin mutex used in high-level interrupt(PIL>10)
  - Interrupt thread(PIL<10) is a special context, be careful to use following functions in an interrupt context

    mutex(9F)/rwlock(9F)/condvar(9F)/semaphore(9F)/biowait(9F)

The constraints of current context

- Self-Deadlock in an interrupt context
  - An simple example:
    - Drivers must call biodone(9F) when the transfer is complete to notify the thread blocked by biowait(9F). biodone(9F) is usually called in the interrupt routine.
    - Call biowait(9F) in an interrupt thread
    - But biodone(9F) only is called by the same interrupt thread

The constraints of current context

- The implicit constraints of timeout(9F)
  - See my blog: http://blog.csdn.net/yayong
  - The clock interrupt thread trigger a taskq thread. That taskq thread ran into callout_execute() by holding the mutex of callout table, then called the driver timeout handler registered by timeout(9F) routine
  - The driver timeout handler blocked on a rwlock waiting for the rwlock owner releasing the rwlock
  - At the same time, the owner of that rwlock was blocked on cv_timedwait, and it has to wait for following clock interrupt to wake up it by calling callout_execute()
  - But the following callout_execute() can't be done unless the previous callout_execute() completed.

Agenda

- What is a system hang?
- Debugging system hangs with kmdb/mdb
- Case Analysis
- **References**
  - > Books
  - > Blogs

Books

- Solaris x86 Crash dump Analysis
  - By Frank.Hofmann (2003-2005)
  - Free download with Creative Commons Deed
  - http://opensolaris.org/os/community/documentation/files/book.pdf
- Solaris Internals
  - By Jim Mauro, Richard McDougall and Brendan Gregg
  - 2nd Edition (July 10, 2006, ISBN 0131482092)
  - Chinese edition have been published
  - http://www.solarisinternals.com/wiki/index.php/Solaris_Internals

Blogs

- Kernel debugging part 1 kmdb
  - http://blogs.sun.com/eschrock/entry/kernel_debugging_part_1_kmdb
- Debugging Solaris scheduling problems
  - http://blogs.sun.com/esaxe/entry/debugging_solaris_scheduling_problems_and
- A R/W deadlock of aggregation in GLD code
  - http://blog.ccw.com.cn/blog-htm-do-showone-itemid-12139-type-blog.html
- Solaris learning journal(6)
  - http://blog.csdn.net/yayong/archive/2007/03/04/1520604.aspx

Agenda

- What is a system hang?
- Debugging system hangs with kmdb/mdb
- Case Analysis
- References
- **Appendix**
  - Crash dump basics
  - Modular Debugger – mdb(1)

Crash dump basics

- It's similar with application core dumps, but...
  - Dump system wide pages whereas a application core dump just contains the 1 specific process's pages
  - It can be debugged by mdb whereas application code dumps can be debugged by dbx/gdb/mdb
  - Managed by dumpadm(1M) whereas application code dumps are managed by coreadm(1M)
- dumpadm(1M)
  - Dump content - kernel pages by default
  - Dump device - /dev/dsk/c0t0d0s1 (swap)
  - Savecore directory - /var/crash/<hostname>

Crash dump basics

- savecore(1M)
  - When the system is rebooted, savecore can be run to retrieve the image from the dump device and archive it to a disk file
- About crash dump files...
  - unix.X - Symbol tables
  - vmunix.X – Memory dump
  - bounds - contains the sequence number to use for the next execution of savecore
  - Check with mdb X
    - eg. mdb 0

Modular Debugger - mdb(1)

- mdb(1) basics
  - commands (dcmd)
    - ::dcmds for a list
    - expression::dcmd - eg: cbd7bad8::ps
    - ::help ::dcmd - ::help ::ps
  - walkers
    - ::walkers for a list
    - expression::walk <walker_name> - e.g. ::walk cpu
  - macros
    - $M for a list
    - $<threadlist

Modular Debugger - mdb(1)

- Symbols and typed data
  - address::print (for symbol)
  - address::print <type>
    - eg. <address>::print cpu_t
    - eg. ::sizeof cpu_t
- Pipelines
  - expression, dcmd or walk can be piped
    - ::walk <walk_name> | ::dcmd
    - e.g. ::walk cpu | ::print cpu_t
  - dcmd or walk can be piped with shell (mdb only)
    - eg. ::ps ! grep bash

Q &A

Oliver.Yang

Oliver.Yang@sun.com

http://oliveryang.net