Introduction to Debugging the FreeBSD Kernel

May 17, 2008

John Baldwin
jhb@FreeBSD.org

Introduction

- Existing Documentation
- DDB
- kgdb
- Debugging Strategies

Existing Documentation

- *Kernel Debugging* chapter of FreeBSD Developer's Handbook
  - Compiling a Debug Kernel
  - Invoking DDB, kgdb
- ddb(4) Manual Page
- GDB Documentation

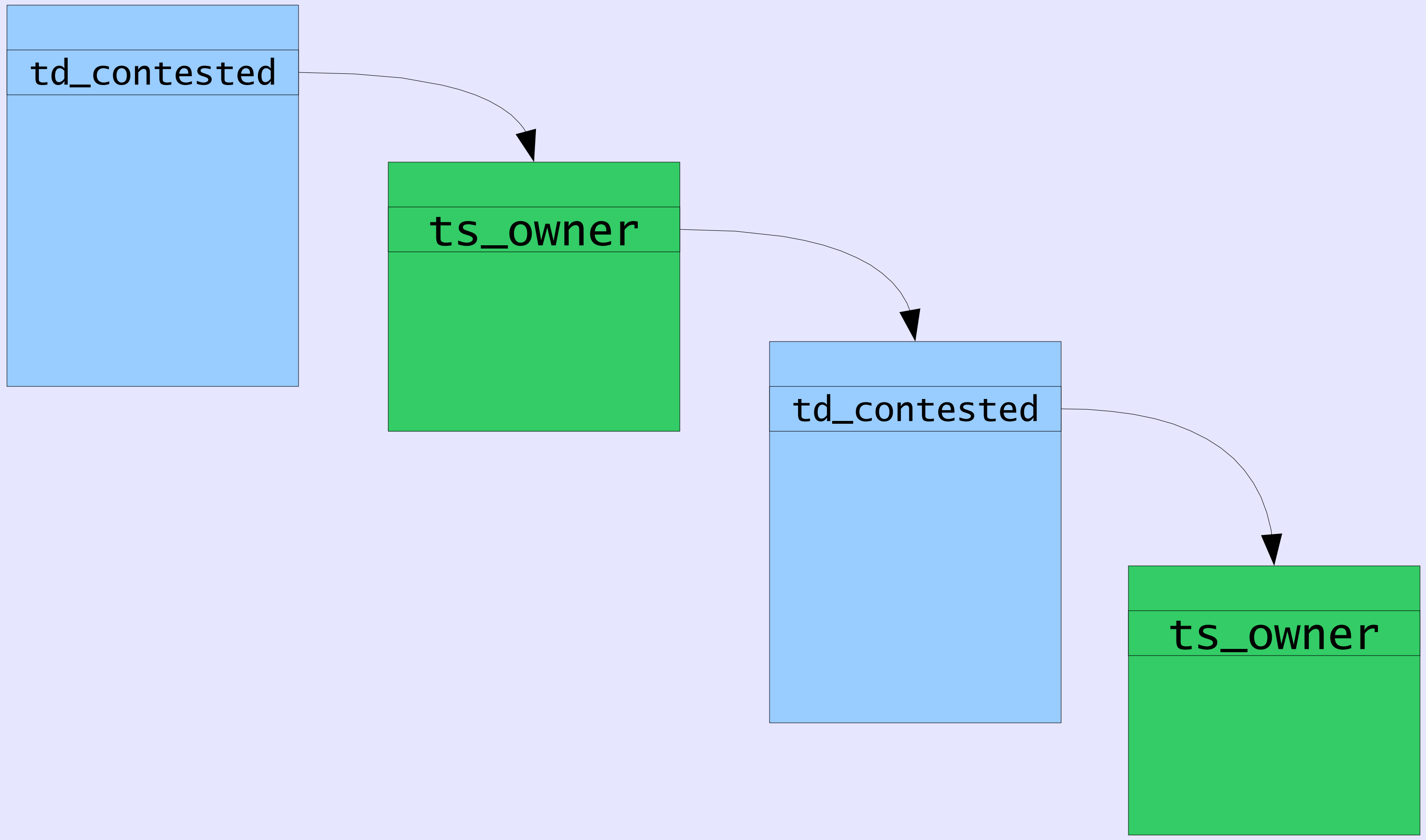
DDB

- Investigating Deadlocks
  - “ps”, “show thread”, and “show turnstile”
  - “show lockchain” and “show sleepchain”
- Adding New Commands

DDB “ps”

```
db> ps
  pid  ppid  pgrp   uid   state   wmesg         wchan        cmd
  954     0     0     0  LL      (threaded)                  crash2
100144                   L      *abc     0xffffff0001288dc0 [crash2: 3]
100143                   L      *jkl     0xffffff0001288c80 [crash2: 2]
100142                   L      *ghi     0xffffff0001288be0 [crash2: 1]
100055                   L      *def     0xffffff0001288d20 [crash2: 0]
  812     0     0     0  SL       -      0xffffffff80673a20 [nfsiod 0]
  771   769   771 26840  Ss+     ttyin   0xffffff00011b9810 tcsh
  769   767   767 26840  S       select  0xffffff00018ca0d0 sshd
  767   705   767     0  Ss      sbwait  0xffffff00016ed94c sshd
...
   10     0     0     0  RL      (threaded)                  idle
100005                   Run     CPU 0                       [idle: cpu0]
100004                   Run     CPU 1                       [idle: cpu1]
100003                   Run     CPU 2                       [idle: cpu2]
100002                   Run     CPU 3                       [idle: cpu3]
```

Threads and Turnstiles



DDB “show thread” and “show turnstile”

```
db> show thread 100055
Thread 100055 at 0xffffff00013869c0:
 proc (pid 954): 0xffffff0001354000
 name: crash2: 0
 stack: 0xffffffffae213000-0xffffffffae216fff
 flags: 0x4  pflags: 0x200000
 state: INHIBITED: {LOCK}
 lock: def  turnstile: 0xffffff0001288d20
 priority: 224
db> show turnstile 0xffffff0001288d20
Lock: 0xffffffffae3c6fc0 - (sleep mutex) def
Lock Owner: 0xffffff000155c680 (tid 100142, pid 954, "crash2: 1")
Shared Waiters:
        empty
Exclusive Waiters:
        0xffffff00013869c0 (tid 100055, pid 954, "crash2: 0")
Pending Threads:
        empty
```

DDB “show lockchain”

```
db> show lockchain 100055
thread 100055 (pid 954, crash2: 0) blocked on lock 0xffffffffae3c6fc0
 (sleep mutex) "def"
thread 100142 (pid 954, crash2: 1) blocked on lock 0xffffffffae3c7000
 (sleep mutex) "ghi"
thread 100143 (pid 954, crash2: 2) blocked on lock 0xffffffffae3c7040
 (sleep mutex) "jkl"
thread 100144 (pid 954, crash2: 3) blocked on lock 0xffffffffae3c6f80
 (sleep mutex) "abc"
thread 100055 (pid 954, crash2: 0) blocked on lock 0xffffffffae3c6fc0
 (sleep mutex) "def"
thread 100142 (pid 954, crash2: 1) blocked on lock 0xffffffffae3c7000
 (sleep mutex) "ghi"
...
```

DDB “show sleepchain”

```
db> ps
  pid  ppid  pgrp   uid   state   wmesg         wchan          cmd
  811     0     0     0  SL      (threaded)                    crash2
100139                   D       fee      0xffffffffae3a9180 [crash2: 3]
100138                   D       four     0xffffffffae3a9140 [crash2: 2]
100137                   D       fo       0xffffffffae3a9240 [crash2: 1]
100136                   D       two      0xffffffffae3a90c0 [crash2: 0]
...
db> show lock fee
 class: lockmgr
 name: fee
 lock type: fee
 state: EXCL (count 1) 0xffffff00013079c0 (tid 100136, pid 811, "crash2: 0")
 waiters: 1
db> show sleepchain 100139
thread 100139 (pid 811, crash2: 3) blocked on lk "fee" EXCL (count 1)
thread 100136 (pid 811, crash2: 0) blocked on sx "two" XLOCK
thread 100137 (pid 811, crash2: 1) blocked on lk "fo" EXCL (count 1)
thread 100138 (pid 811, crash2: 2) blocked on sx "four" XLOCK
thread 100139 (pid 811, crash2: 3) blocked on lk "fee" EXCL (count 1)
...
```

Adding new DDB Commands

- Declaring Commands
- DDB Console I/O
- Using DDB's Symbol Table

Declaring a DDB Command

- DB_COMMAND()
- Function Arguments
  - addr
  - have_addr
  - count
  - modif

```
DB_COMMAND(foo, db_foo_cmd)
{
  struct foo *foop;
  int i;

  if (have_addr)
    foop = (struct foo *)addr;
  else
    foop = &default_foo;

  /* Default count. */
  if (count == -1)
    count = 1;
  for (i = 0; i < count; i++)
    do_something(foop);
}
```

DDB I/O

- Use db_printf() Instead of printf()
- Global Variable db_pager_quit
- Use db_disable_pager() to Disable Pager

```
DB_SHOW_COMMAND(foos, db_show_foos_cmd)
{
  struct foo *foop;
  int verbose;

  verbose = index(modif, 'v') != NULL;
  TAILQ_FOREACH(foop, &allfoos, f_list) {
    if (verbose)
      db_printf("%p: ", foop);
    db_printf("%s (%d)\n" foop->f_name,
       foop->f_count);
    if (db_pager_quit)
      break;
  }
}
```

Using DDB's Symbol Tables

- Use db_search_symbol() to find the nearest symbol to an address.
- Use db_symbol_values() to get the name and value.

```
#if defined(DDB)
  const char *name;
  c_db_sym_t sym;
  db_expr_t  offset;

  sym = db_search_symbol(
    (vm_offset_t)(*sipp)->func,
    DB_STGY_PROC, &offset);
  db_symbol_values(sym, &name,
    NULL);
  if (name != NULL)
    printf("   %s(%p)... ", name,
      (*sipp)->udata);
  else
#endif
    printf("   %p(%p)... ",
      (*sipp)->func,
      (*sipp)->udata);
```

kgdb

- Debugging Kernel Modules
- Extending kgdb with User-Defined Commands

kgdb and Kernel Modules

- Each module has to have symbols loaded individually
- kgdb's integrated kernel module support
  - “add-kld” command loads symbols for a single module
  - kgdb treats kernel modules as shared libraries
- The asf(8) utility can be used with older kgdb binaries or kernels without debug symbols

kgdb's Integrated KLD Support

```
> sudo kgdb -q
Loaded symbols for /boot/kernel/iwi_bss.ko
Loaded symbols for /boot/kernel/logo_saver.ko
...
(kgdb) info sharedlibrary
From        To          Syms Read   Shared Object Library
0xc3e8e5a0  0xc3e8e63b  Yes         /boot/kernel/iwi_bss.ko
0xc41037a0  0xc4103c28  Yes         /boot/kernel/logo_saver.ko
(kgdb) info files
Symbols from "/boot/kernel/kernel".
kernel core dump file:
        `/dev/mem', file type FreeBSD kernel vmcore.
Local exec file:
        `/boot/kernel/kernel', file type elf32-i386-freebsd.
        Entry point: 0xc04513c0
        ...
        0xc3e8e5a0 - 0xc3e8e63b is .text in /boot/kernel/iwi_bss.ko
        0xc3e8e63b - 0xc3e8e724 is .rodata in /boot/kernel/iwi_bss.ko
        0xc3e8f000 - 0xc3ebdb04 is .data in /boot/kernel/iwi_bss.ko
        0xc3ebdb04 - 0xc3ebdb7c is .dynamic in /boot/kernel/iwi_bss.ko
        0xc3ebdb7c - 0xc3ebdb88 is .got in /boot/kernel/iwi_bss.ko
        0xc3ebdb88 - 0xc3ebdb8c is .bss in /boot/kernel/iwi_bss.ko
        ...
```

kgdb Scripting Gotchas

- Limited control flow
- Arguments
  - No argument count
  - Not local variables with local scope
- String literals
- No way to abort execution of a user-defined command

Debugging Strategies

- Kernel Crash
  - page fault: corrupt data structure
  - kmem_map: possible resource exhaustion
  - Check for bad hardware for “weird” panics
- System Hangs
  - Check console messages; resource exhaustion?
  - Use DDB to inspect system state; “ps”, etc.
  - Get a crash dump for offline analysis

Q&A

- Paper and slides are available online
  - http://www.FreeBSD.org/~jhb/papers/bsdcan/2008/
- Some kgdb scripts for 4.x and 6.x are also available
  - http://www.FreeBSD.org/~jhb/gdb/
- Questions?

Kernel Crash Messages

- Panic String
  - Simple Description
  - grep'able
- Memory Access Fault
  - Faulting Address
  - Program Counter
  - Current Process

Sample amd64 Page Fault

```
Fatal trap 12: page fault while in kernel mode
cpuid = 0; apic id = 00
fault virtual address    = 0x4
fault code               = supervisor read, page not present
instruction pointer      = 0x8:0xffffffff80359af8
stack pointer            = 0x10:0xffffffffa3cbb550
frame pointer            = 0x10:0xffffffffa3cbb570
code segment             = base 0x0, limit 0xfffff, type 0x1b
                         = DPL 0, pres 1, long 1, def32 0, gran 1
processor eflags         = interrupt enabled, resume, IOPL = 0
current process          = 31466 (netstat)
trap number              = 12
panic: page fault
```

DDB “show proc” and “show thread”

```
db> show proc 954
Process 954 (crash2) at 0xffffff0001354000:
 state: NORMAL
 uid: 0  gids: 0
 parent: pid 0 at 0xffffffff806538e0
 ABI: null
 threads: 4
100144                  L      *abc      0xffffff0001288dc0 [crash2: 3]
100143                  L      *jkl      0xffffff0001288c80 [crash2: 2]
100142                  L      *ghi      0xffffff0001288be0 [crash2: 1]
100055                  L      *def      0xffffff0001288d20 [crash2: 0]
db> show thread 100055
Thread 100055 at 0xffffff00013869c0:
 proc (pid 954): 0xffffff0001354000
 name: crash2: 0
 stack: 0xffffffffae213000-0xffffffffae216fff
 flags: 0x4  pflags: 0x200000
 state: INHIBITED: {LOCK}
 lock: def  turnstile: 0xffffff0001288d20
 priority: 224
```

DDB “show lock” and “show turnstile”

```
db> show lock def
 class: sleep mutex
 name: def
 flags: {DEF}
 state: {OWNED, CONTESTED}
 owner: 0xffffff000155c680 (tid 100142, pid 954, "crash2: 1")
db> show turnstile def
Lock: 0xffffffffae3c6fc0 - (sleep mutex) def
Lock Owner: 0xffffff000155c680 (tid 100142, pid 954, "crash2: 1")
Shared Waiters:
        empty
Exclusive Waiters:
        0xffffff00013869c0 (tid 100055, pid 954, "crash2: 0")
Pending Threads:
        empty
```

Threads and Processes in kgdb

- kgdb maps each kernel thread to a GDB thread
- The “info threads” and “thread” commands work just as in GDB
- kgdb adds “proc” and “tid” commands which accept kernel PIDs and TIDs

Examining Crash Dumps with System Utilities

- Several system utilities can use libkvm(3) to analyze crash dumps
- Use -M and -N arguments to specify kernel and vmcore
- ps(1), netstat(1), vmstat(8), etc.

Conclusion