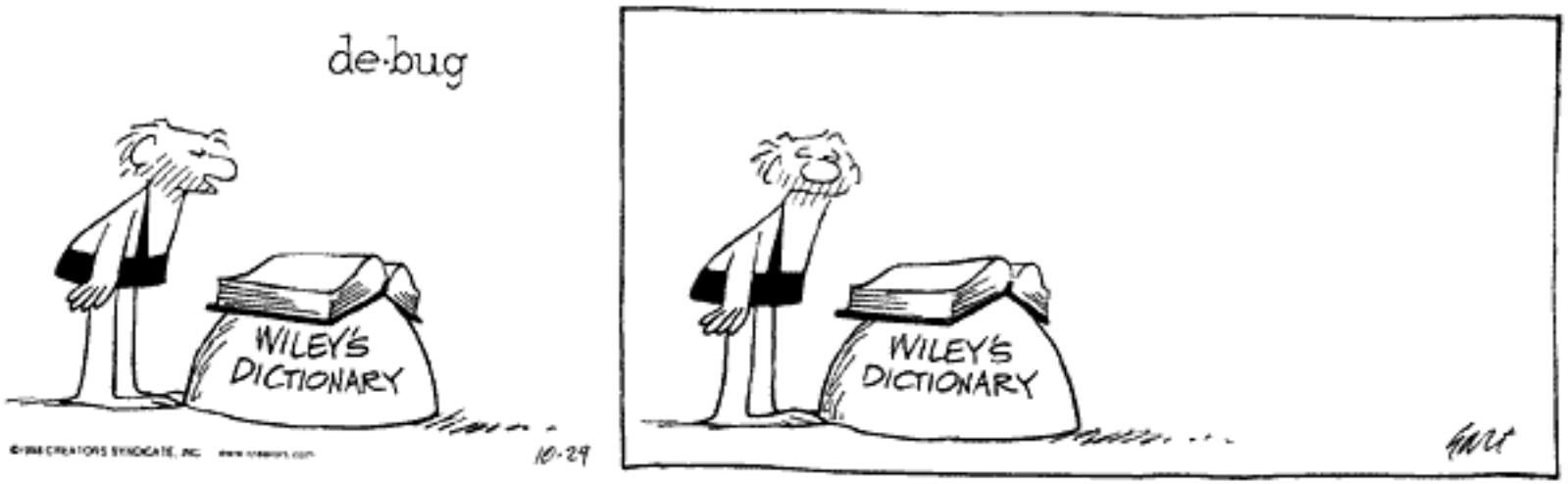
A Brief Tour of the Modular Debugger

Mike Shapiro
(mws@eng.sun.com)
Solaris Kernel Group

History of Kernel Debugging



- UNIX Version 6:

```
panic(s)
char *s;
{
        panicstr = s;
        update();
        printf("panic: %s\n", s);
        for (;;)
                idle();
}
```

Kernel Debugging Taxonomy

- `if (no_advanced_debugging) printf(9f)`
- `ASSERT(i_am_a_debug_kernel != 0);`
- In-situ kernel debugger (`kadb`)
- In-situ firmware debugger (SPARC obp)
- Run-time tracing (trap trace, kmem allocator)
- Post-mortem debuggers (`adb`, `crash`)

Problems

- Software (and bug) complexity is increasing rapidly, without corresponding advances in debugging tools
- There are more Solaris developers (and OEMs and ISVs) delivering OS code than ever, but there is still no way to easily share debugging technology
- Maintenance burden of existing tools is increasing; old tools are not designed to handle future needs
- Debugging support shows up late (or never)

Principles of OS Debugging

- The most important bugs occur at customer sites, cause downtime for mission-critical machines, and cannot be deterministically reproduced
- Customers should not and will not perform experiments and debugging tasks for us
- We must be able to perform root-cause analysis from a single crash dump and deliver an appropriate fix
- Debugging support for new subsystems is required at the same time as project integration

Debugging Roadmap

7

- Make sure we always get the dump!
- libkvm enhancements to facilitate analysis
- dumpadm(1M) to configure dump parameters

8

- Modular Debugger (MDB)
- Kernel panic sequence enhancements
- coreadm(1M) to administer user core files
- proc(1) enhancements for user core files

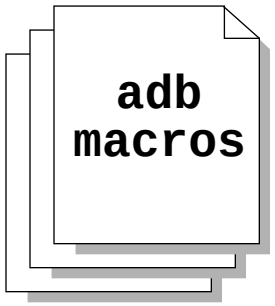
9

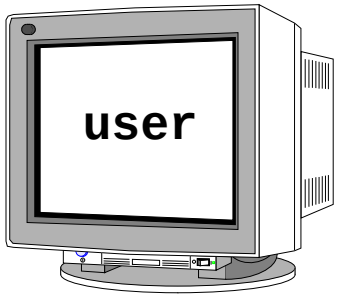
- Dynamic tracing and instrumentation
- MDB enhancements

MDB: The Modular Debugger

- New debugger designed to facilitate advanced analysis of kernel crash dumps during Solaris 7 development
- First-class programming API to allow developers to implement their own debugger commands and analysis tools without having to recompile or modify MDB itself
- Backward compatibility with existing adb syntax and macro files; include useful crash commands
- Enhanced usability features
- Robust design for current and future architectures

MDB Architecture





unix module

nfs module

MDB Language

MDB Module API

Debugger Engine

disassembler

libkvm target

/proc target

Sun microsystems

MDB Layering

```
> ::softint
ADDR                PEND  PIL        ARG   HNLDR
0000000010432e48    0     1          0     softlevel1
0000000010432e68    0     13         0     poke_cpu_intr

> ::walk softint
10432e48
10432e68

> 10432e48::softint
ADDR                PEND  PIL        ARG   HNLDR
0000000010432e48    0     1          0     softlevel1

> ::walk softint | ::softint
ADDR                PEND  PIL        ARG   HNLDR
0000000010432e48    0     1          0     softlevel1
0000000010432e68    0     13         0     poke_cpu_intr
```

MDB Programming I

```
int
softint(uintptr_t addr, uint_t flags,
        int argc, const mdb_arg_t *argv)
{
        struct intr_vector iv;

        /* based on flags, print header or invoke walker ... */

        if (mdb_vread(&iv, sizeof (iv), addr) == -1) {
                mdb_warn("couldn't read intr_vector at %p", addr);
                return (DCMD_ERR);
        }

        mdb_printf("%0?p %4d %4d %?p %a\n", addr,
                iv.iv_pending, iv.iv_pil, iv.iv_arg, iv.iv_handler);
        return (DCMD_OK);
}
```

MDB Programming II

```
/* based on flags, print header or invoke walker ... */

if (!(flags & DCMD_ADDRSPEC)) {
      if (mdb_walk_dcmd("softint", "softint", argc, argv)<0) {
            mdb_warn("can't walk softint");
            return (DCMD_ERR);
      }
      return (DCMD_OK);
}

if (DCMD_HDRSPEC(flags)) {
      mdb_printf("%?s %4s %4s %?s %s\n",
            "ADDR", "PEND", "PIL", "ARG", "HNLDR");
}
```

Example 1: Memory Leaks

- **`::findleaks`** - responsible for detecting 13 separate bugs in the last two builds of 5.7, 85 more in 5.8

- Latest version is extremely fast, can find circular leaks

```
> ::findleaks
CACHE     LEAKED   BUFCTL              TIMESTAMP CALLER
f540c6e8       5 fd31f8a0       2324730701010 lm_dup+0x10
f540c6e8       1 fa87f540        210119896850 lm_dup+0x10
f540c6e8       1 fc671420       1442224346250 lm_dup+0x10
f540f068      14 fc40f4a0        254759126720 lm_dup+0x10
f540f068       4 fa1fc560        215239318510 lm_dup+0x10
...

> fd31f8a0$<bufctl_audit
...
```

Example 2: Data Corruption

- **`::whatis`** - use kmem caches and other information to determine a data type from a pointer

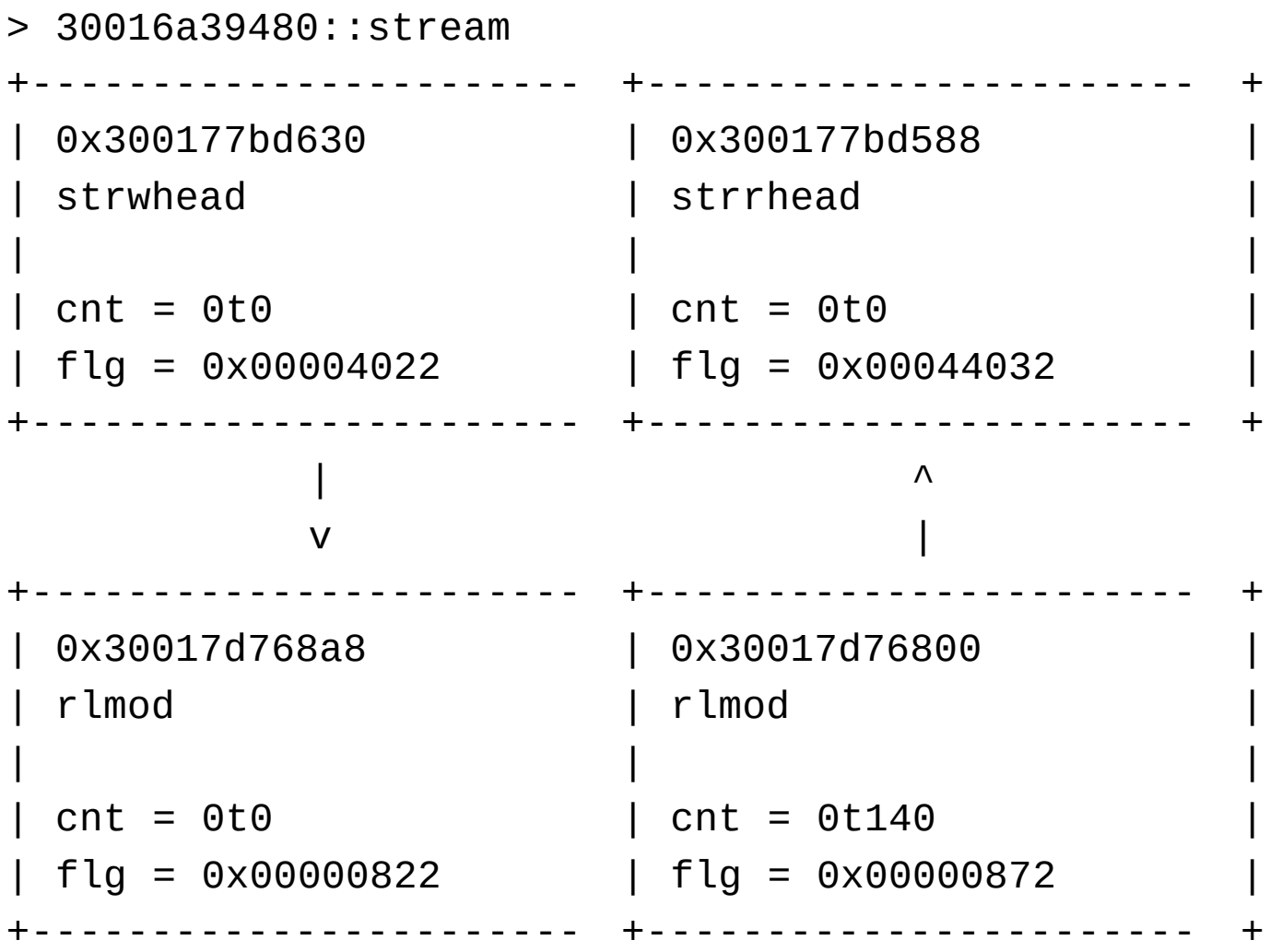
```
> 300013054a4::whatis
300013054a0 is 300013054a0+4, allocated from streams_dblk_32

> 300013054a0::kgrep
300012f8288
300013de7a8
300013de808
300013de868
300013de8c8

> 300013de7a8::whatis
300013de7a8 is 300013de780+28, allocated from streams_mblk
```

Example 3: Dispatcher

- Display dispatch queue and interrupt threads:

```
> 1::cpuinfo -v
ID ADDR     FLG  NRUN  BSPL  PRI  RNRN  KRNRN  SWITCH  THREAD        PROC
 1 104ae000 1b   8     9     0    no    no     t-0     3000797a920   xemacs
                    |     |
                    |     +--> PIL THREAD
                    |            10 000002a100075d60
                    |
                    +-->  PRI THREAD            PROC
                           169 000002a100075d60 sched
                            60 000002a100791d60 sched
                            60 000002a1027dfd60 sched
                            60 000002a1014cfd60 sched
                            60 000002a1017a3d60 sched
                            60 000002a101eddd60 sched
```

Example 4: STREAMS



Case 1: A Tale of Two File Descriptors

```
configuring network interfaces: spwr0.
Hostname: fastfood
The system is coming up.  Please wait.
checking ufs filesystems
/dev/rdsk/c0t1d0s7: is stable.
/dev/rdsk/c0t0d0s7: is stable.
/dev/rdsk/c0t1d0s0: is clean.
/dev/rdsk/c0t0d0s6: is clean.
NIS domainname is sunsoft.eng.sun.com
starting router discovery.
starting rpc services: rpcbind keyserv ypbind done.
Setting netmask of spwr0 to 255.255.255.0
Setting default interface for multicast: add net 224.0.0.0: gateway fastfood
syslog service starting.
Print services started.
volume management starting.
mibiisa: if_init failed
The system is ready.

fastfood console login:
```

main() and if_init()

```
/* ... close all file descriptors ... */

if ((snmp_socket = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
      SYSLOG0("Can't get socket");
      return (-1);
}

if (if_init() == -1) {
      SYSLOG0("if_init failed");
      return (-1);
}
```

```
int if_init()
{
      if (mibopen() == 0)
            return (-1);
      /* else do some other stuff and return 0 */
```

mibopen( )

```
static int mibfd = -2;
int mibopen()
{
        mutex_lock(&mibget_mut);
        if (mibfd == -2 ) {
                mibfd = open("/dev/ip", 2);
                if (mibfd == -1) {
                        perror("ip open");
                        close(mibfd);
                        mutex_unlock(&mibget_mut);
                        return (-1);
                }
                /* ... */
        }
        mutex_unlock(&mibget_mut);
        return (mibfd);
}
```

Case 1 Strategy

- Modify daemon to force system panic when error condition occurs
- Enable kernel memory tracing facility to record memory activity relating to freeing a file table entry
- Drop several machines into reboot loops waiting for condition to reproduce

Kernel Memory Debugging

```
> e21feea8$<file
0xe21feeb0: flag        count       vnode
            3           3           e20f7184
0xe21feeb8: offset      cred        audit_data
            0           e1387f58    0

> e21feea8/20X
0xe21feea8: 0           baddcafe    badd0003    e20f7184
            0           0           e1387f58    0
            3           baddcafe    feedface    feedface
            e21ff000    430f38ed    0           baddcafe
            badd0003    e20eea40    0           0
```

File Descriptor 0's bufctl_audit

```
> e21ff000$<bufctl_audit
0xe21ff000:       next            addr            slab
                  e184b5a0        e21feea8        e2189a20
0xe21ff00c:       cache           timestamp       thread
                  e13826e8        9121733a4       e1e536a0
0xe21ff01c:       lastlog         contents        stackdepth
                  e07b1000        0               7
0xe21ff028:
                  kmem_cache_alloc_debug+0xe7
                  kmem_cache_alloc_global+0x1a6
                  kmem_cache_alloc+0x1db
                  falloc+0x2e
                  copen+0x37
                  open+0x13
                  watch_do_syscall+0xbe
```

Memory Allocations by e1e536a0

```
> e1e536a0::allocdby
BUFCTL         TIMESTAMP CALLER
e2245d20     9129cf7dc forklwp+0x1c5
e2222560     9129b17dc lwp_create+0x12e
e13d5bc0     9129a02b6 thread_create+0x21
e22411a0     9129951c2 anon_alloc+0x13
e202ff20     91297712c segkp_get_internal+0x69
e1f4c8c0     91296b3ae setuctxt+0x74
e20067a0     91295714c pid_assign+0xf
e210e080     912950d92 getproc+0x16
e2242ea0     9123cf45e anon_alloc+0x13
...
e2222920     9121abeac allocq+0x10
e21ff000     9121733a4 falloc+0x2e
e2200f80     9120b43dc falloc+0x2e
...
```

Other Threads on the Scene

```
> e1e99790::walk thread
e1e536a0    (Thread 1)
e210fb80    (Thread 2)
e210fa00
e210f880

> e210fb80::freedby
BUFCTL       TIMESTAMP CALLER
e2200e00     9120c60dc closef+0xee

> e2200e00$<bufctl_audit
0xe2200e1c:       lastlog          contents         stackdepth
                  e07435c0         e0aaf910         6
0xe2200e28:
                 kmem_cache_free_debug+0x126
                 kmem_cache_free+0x24
                 closef+0xee
```

The Past Life of fd 0

```
> e0aaf910$<file
0xe0aaf918:       flag              count             vnode
                  3                 0                 e1d977fc
0xe0aaf920:       offset            cred              audit_data
                  0                 e1387f58          0

> e1d977fc::vnode2path
/devices/pseudo/mm@0:zero
```

Memory Allocations/Frees by Thread 2

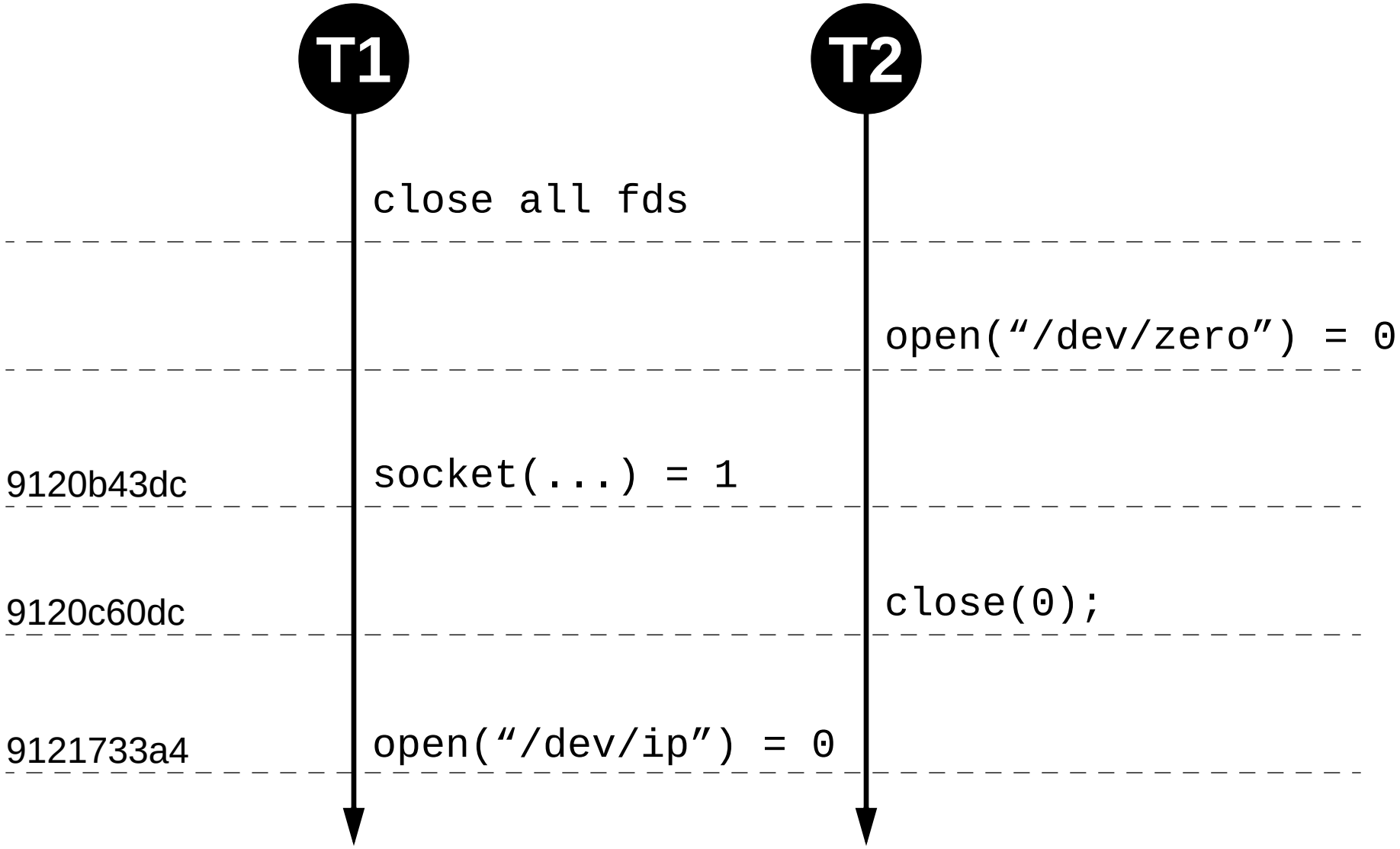
```
> e210fb80::allocdby
BUFCTL      TIMESTAMP CALLER
e1f02860    9121bb3f2 lwp_create+0x18b
e2222500    9121b6b2c lwp_create+0x12e
e202fec0    9121ad784 tnf_thread_create+0x11
e210ec80    9121a41fc thread_create+0xe1
e13d58c0    91219a5c6 thread_create+0x21
e2242e40    91218bc56 anon_alloc+0x13
e2242de0    91217ae56 anon_alloc+0x13
e202fe00    912171cca segkp_get_internal+0x69
e2242d80    91210a7dc anon_alloc+0x13
e219bc60    9120fb16a anon_create+0x5d
e219bc00    9120f68fe anon_create+0x1d
e2072c20    9120eef5a anonmap_alloc+0x11
e20fb1a0    9120d26e8 segvn_vpage+0x58
e20731e0    9120b6182 segvn_create+0x2af
e2029ea0    9120a874e seg_alloc+0x75
```

```
> e210fb80::freedby
TIMESTAMP   CALLER
9120c60dc   closef+0xee
```

Back to Thread 1

```
> e1e536a0::allocdby
...
e21ff000    9121733a4 falloc+0x2e
e2200f80    9120b43dc falloc+0x2e
...
> e2200f80$<bufctl_audit
0xe2200f80:       next              addr              slab
                  e184b540          e21feee0          e2189a20
0xe2200f8c:       cache             timestamp         thread
                  e13826e8          9120b43dc         e1e536a0
0xe2200f9c:       lastlog           contents          stackdepth
                  e07b0a60          e0aa5388          5
0xe2200fa8:
                  kmem_cache_alloc_debug+0xe7
                  kmem_cache_alloc+0x1bc
                  falloc+0x2e
                  sockfs‘so_socket+0x124
```

Timeline


T1
T2
close all fds
open(“/dev/zero”) = 0
9120b43dc
socket(...) = 1
9120c60dc
close(0);
9121733a4
open(“/dev/ip”) = 0

Case 1 Solution

- Thread `e210fb80` opens `/dev/zero` to map a stack; `/dev/zero` is assigned fd `0`
- Thread `e1e536a0` performs `socket()`; gets back fd `1`
- Thread `e210fb80` closes fd `0`
- Thread `e1e536a0` calls `mibopen()`; `/dev/ip` gets fd `0`
- The broken `mibopen()` function returns `0` but has actually succeeded; `if_init()` failure is reported

Case 2: Sendmail dies in two SIGALRM fire

```
jurassic# ls /var/core/jurassic
core.sendmail.414918

jurassic# pstack /var/core/jurassic/core.sendmail.414918
core 'core.sendmail.414918' of 414918: /usr/lib/sendmail -bd
 00063600 sfgets   (0, 0, 0, 0, f0878, d1800) + 80

jurassic# mdb /var/core/jurassic/core.sendmail.414918
Loading modules: []
> $c
sfgets+0x80(0, 0, 0, 0, f0878, d1800)
> <pc/i
sfgets+0x80:    st        %o0, [%fp - 0xc]
> <fp=K
                1
> <o7=p
                libc.so.1'_libc_siglongjmp+0x1c
```

sfgets() and readtimeout()

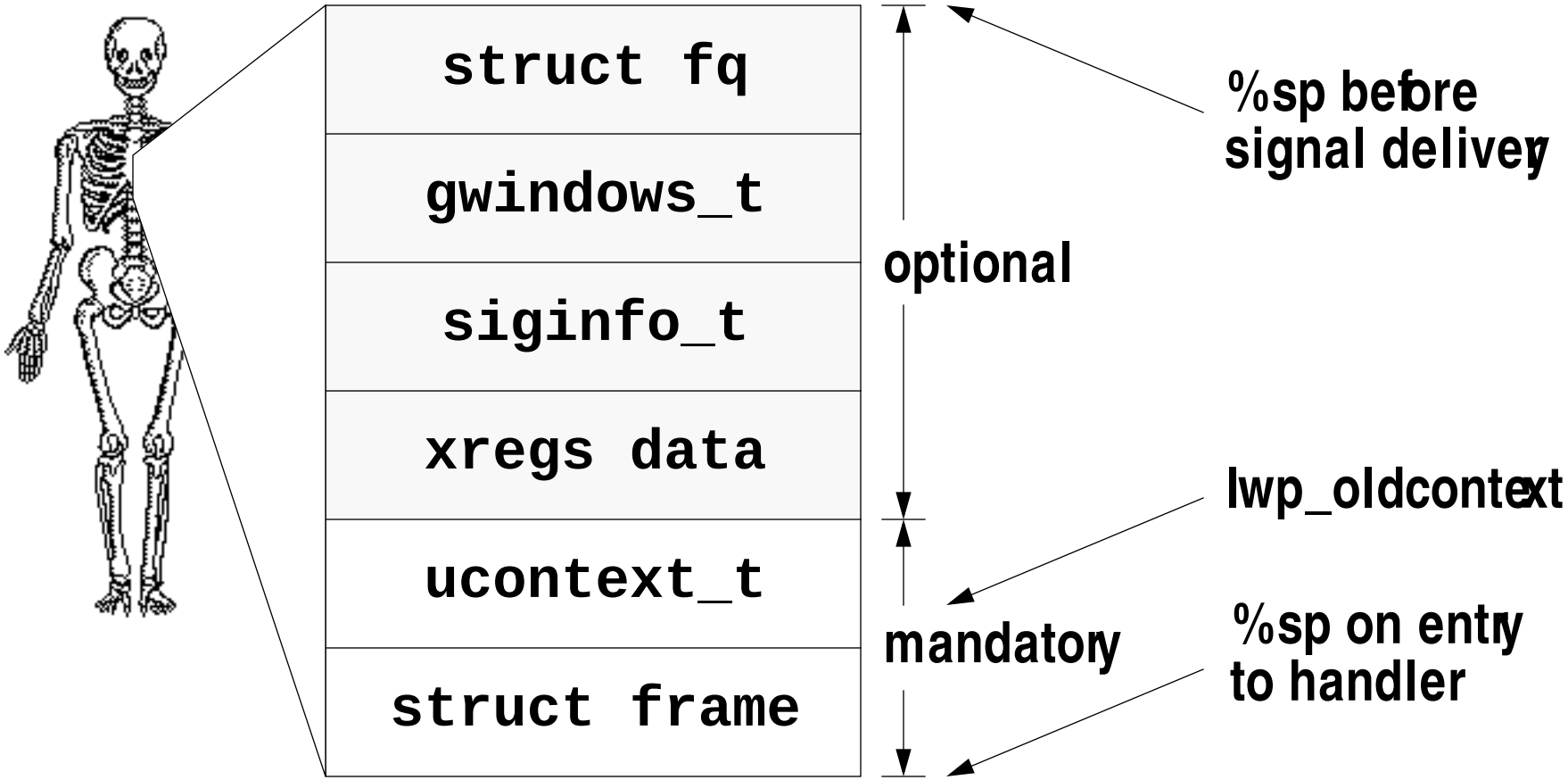
```
static sigjmp_buf CtxReadTimeout;
char *
sfgets(buf, siz, fp, timeout, during)
{
      /* ... */
      if (sigsetjmp(CtxReadTimeout, 1) != 0) {
            sm_syslog(LOG_NOTICE, CurEnv->e_id, “timeout “
                  “waiting for input from %.100s “,
                  CurHostName ? CurHostName : "local”);
      /* ... perform read on socket ... */
}

static void
readtimeout(time_t timeout)
{
      siglongjmp(CtxReadTimeout, 1);
}
```

Case 2 Strategy

- Use `sigjmp_buf` and registers to confirm current `%sp`
- Develop walkers to determine active signal handlers and corresponding saved `ucontext_t` structures
- Use results to locate stack prior to `siglongjmp()`
- Hope that we can develop and prove a hypothesis on what caused the erroneous `siglongjmp()` based on the state of the program prior to the SIGALRM

Anatomy of a Signal Handler

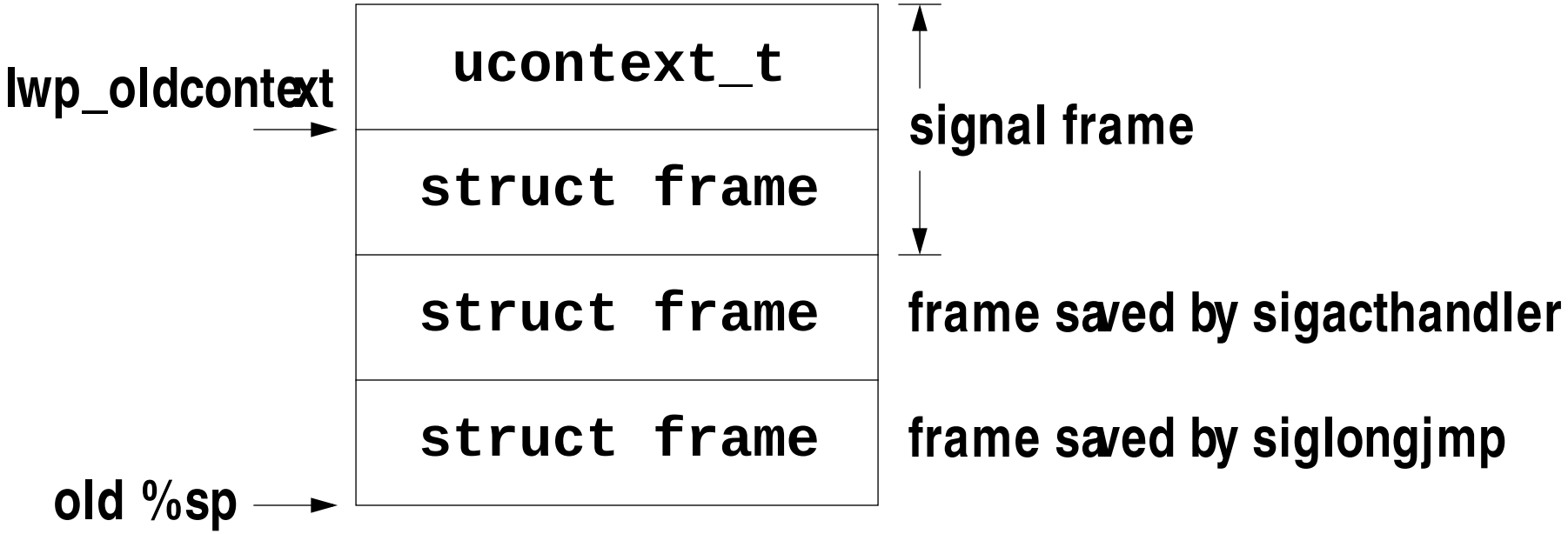
- All **`sigaction(3C)`** handlers are vectored through `libc'sigacthandler()`



Finding the Stack I

```
> 0xcc800+304::nmadd -s 0t76 CtxReadTimeout
added CtxReadTimeout, value=ccb04 size=4c
> ::nm -P
Value      Size       Type  Bind  Other Shndx    Name
0x000ccb04|0x0000004c|NOTY |GLOB |0x0  |UNDEF   |CtxReadTimeout

> CtxReadTimeout/3p
CtxReadTimeout:
CtxReadTimeout: 1                0xffbe8630      sfgets+0x50
                flags            saved %sp       saved %pc
> <sp=K
                ffbe8630
```

Finding the Stack II




```
> ::walk oldcontext
ffbe82c0

> ffbe82c0 - (0x60 * 3) = K
                ffbe81a0
```

Finding the Stack III

```
> 0xffbe81a0$C
00000000 0(0, 0, 0, 0, 0, 0)
ffbe81a0 0(d0800, 654c6, 6386c, 0, 0, 0)
ffbe8200 libc.so.1‘sigacthandler+0x28(e, 0, ffbe82c0)
ffbe8578 libc.so.1‘_wait+0xc(654d7, f07c8, d00c8)
ffbe85e0 endmailer+0x7c(188c6c, d0ae0, 0, d0ae0, a9c08, 5)
ffbe8640 smtpquit+0x88(f0878, d1800, 1, d0ae0, 188c6c, 54)
ffbe86a0 reply+0x3fc(cf400, cfc00, cfc00, 1, 0, 188c6c)
ffbe9700 smtpquit+0x64(f0878, d1800, 0, d0ae0, 188c6c, 78)
ffbe9760 mci_flush+0x58(2, 4, cfb28, 0, 1, 0)
ffbe97c0 finis+0xdc(cec00, cfabe, ffbe98e8, ff1ba000, 0, 0)
ffbe9828 libc.so.1‘sigacthandler+0x28(f, ffbe9ba0, ffbe98e8)
ffbe9c20 libc.so.1‘sigacthandler+0x18(1, ffbe9f98, ffbe9ce0)
ffbea018 libc.so.1‘_read+0xc(cfba0, 18a0f4, cfba0)
ffbea078 libc.so.1‘fgets+0x24c(ff1c164c, 0, cfba0)
ffbea0d8 sfgets+0x164(d0000, cfba0, cfba0, e10, c44a0, a9b8f)
```

Walking the uc_link List

- Using the stack and uc_link walker, we can correlate saved ucontext_t's and the corresponding siginfo:

```
> ::walk oldcontext | ::walk ucontext
ffbe82c0  <-- SIGALRM that caused siglongjmp of death      [1]
ffbe82b8  <-- ???                                          [2]
ffbe98e8  <-- siginfo ffbe9ba0 = SIGTERM from PID 424186   [3]
ffbe9ce0  <-- siginfo ffbe9f98 = SIGHUP from PID 424186    [4]
```

- What was the mystery signal?

sigbrokenjmp

```
int sigsetjmp(sigjmp_buf env, int savemask)
{
        ucontext_t uc;

        getcontext(&uc);
        /* save %pc, %sp, stack_t and signal mask into env */
        return (0);
}

int siglongjmp(sigjmp_buf env, int value)
{
        ucontext_t uc;

        getcontext(&uc);
        /* restore saved information from env into uc */
        setcontext(&uc);
}
```

A Second SIGALRM

```
        ...
        ffbe8640 smtpquit+0x88(f0878, d1800, 1, d0ae0, 188c6c, 54)
        ffbe86a0 reply+0x3fc(cf400, cfc00, cfc00, 1, 0, 188c6c)
        ffbe9700 smtpquit+0x64(f0878, d1800, 0, d0ae0, 188c6c, 78)
        ...

int
reply(m, mci, e, timeout, pfunc)
{
        /*  ... */
        p = sfgets(bufp, MAXLINE, mci->mci_in, timeout,
              SmtpPhase);
        mci->mci_lastuse = curtime();
        /* ... */
        if (p == NULL) {
              /* ... */
              smtpquit(m, mci, e);
```

Sendmail Event Queue

```
struct event {
        time_t ev_time;         /* time of the function call */
        void (*ev_func)(int); /* function to call */
        int ev_arg;             /* argument to ev_func */
        int ev_pid;             /* pid that set this event */
        struct event *ev_link;/* link to next item */
}

> EventQueue/K
0xd09ec:          0

> *FreeEventList/YpnXDK
0xf1cd0:          1999 Sep 15 15:04:10    readtimeout
                  0                 414918            f1b50
> f1b50/YpnXDK
0xf1b50:          1999 Sep 15 15:02:20    readtimeout
                  0                 414918            0
```

Mailer Connection Info

```
if (p == NULL) {
        if (errno == 0)
                errno = ECONNRESET;
        mci->mci_errno = errno;
        mci->mci_state = MCIS_ERROR;
        smtpquit(m, mci, e);
}
```

- We can retrieve the mci in question from the stack:

```
> 188c6c/6xnXXnppDnXXXXXnYp
0x188c6c:       2048    83      0       0       0       0
                6c              0
                0               0               414935
                c44a0           f0878           130920
                0               0
                1999 Sep 15 15:02:20    0
```

Case 2 Solution

- `sfgets()` was pending, SIGHUP interrupted
- SIGHUP handling interrupted by SIGTERM
- `finis()` -> `reply()` -> `sfgets()`
- `sfgets()` pending again, first SIGALRM interrupts
- `reply()` -> `smtpquit()`, re-using stack referred to by CtxReadTimeout's saved %sp
- SIGALRM from second `sfgets()` now delivered, causing another siglongjmp to the same saved %sp

Future Work: MDB

- User process control (breakpoints, watchpoints)
- Registers and stack traces after `::context` switch
- Debugging support for raw files and devices
- Extract user core dump from crash dump
- Target for examining raw dump device
- Kernel stabs support, EOL most adb macros
- Kmdb to replace kadb, better fsdb

MDB Resources

- Solaris Modular Debugger Guide (answerbook) available on-line at **`docs.sun.com`**
- **`mdb(1)`** man page (subset of answerbook)
- `/usr/demo/mdb` module developer kit