Debugging Kernel Problems

Greg Lehey
grog@FreeBSD.org
grog@MySQL.com
grog@NetBSD.org
Basel, 25 November 2005

Introduction

The Complete FreeBSD ends with the text:

The seven eyes of Ningauble the Wizard floated back to his hood as he reported to Fafhrd: "I have seen much, yet cannot explain all. The Gray Mouser is exactly twenty-five feet below the deepest cellar in the palace of Gilpkerio Kistomerces. Even though twenty-four parts in twenty-five of him are dead, he is alive."

Introduction

The Complete FreeBSD ends with the text:

The seven eyes of Ningauble the Wizard floated back to his hood as he reported to Fafhrd: "I have seen much, yet cannot explain all. The Gray Mouser is exactly twenty-five feet below the deepest cellar in the palace of Gilpkerio Kistomerces. Even though twenty-four parts in twenty-five of him are dead, he is alive."

"Now about Lankhmar. She's been invaded, her walls breached everywhere and desperate fighting is going on in the streets, by a fierce host which out-numbers Lankhmar's inhabitants by fifty to one—and equipped with all modern weapons. Yet you can save the city."

Introduction

The Complete FreeBSD ends with the text:

The seven eyes of Ningauble the Wizard floated back to his hood as he reported to Fafhrd: "I have seen much, yet cannot explain all. The Gray Mouser is exactly twenty-five feet below the deepest cellar in the palace of Gilpkerio Kistomerces. Even though twenty-four parts in twenty-five of him are dead, he is alive."

"Now about Lankhmar. She's been invaded, her walls breached everywhere and desperate fighting is going on in the streets, by a fierce host which out-numbers Lankhmar's inhabitants by fifty to one—and equipped with all modern weapons. Yet you can save the city."

"How?" demanded Fafhrd.

Introduction

The Complete FreeBSD ends with the text:

The seven eyes of Ningauble the Wizard floated back to his hood as he reported to Fafhrd: "I have seen much, yet cannot explain all. The Gray Mouser is exactly twenty-five feet below the deepest cellar in the palace of Gilpkerio Kistomerces. Even though twenty-four parts in twenty-five of him are dead, he is alive."

"Now about Lankhmar. She's been invaded, her walls breached everywhere and desperate fighting is going on in the streets, by a fierce host which out-numbers Lankhmar's inhabitants by fifty to one—and equipped with all modern weapons. Yet you can save the city."

"How?" demanded Fafhrd.

Ningauble shrugged. "You're a hero. You should know."
(Fritz Leiber, from "The Swords of Lankhmar")

Introduction

The Complete FreeBSD ends with the text:

The seven eyes of Ningauble the Wizard floated back to his hood as he reported to Fafhrd: "I have seen much, yet cannot explain all. The Gray Mouser is exactly twenty-five feet below the deepest cellar in the palace of Gilpkerio Kistomerces. Even though twenty-four parts in twenty-five of him are dead, he is alive."

"Now about Lankhmar. She's been invaded, her walls breached everywhere and desperate fighting is going on in the streets, by a fierce host which out-numbers Lankhmar's inhabitants by fifty to one—and equipped with all modern weapons. Yet you can save the city."

"How?" demanded Fafhrd.

Ningauble shrugged. "You're a hero. You should know."
(Fritz Leiber, from "The Swords of Lankhmar")

This tutorial aims to make you a saviour. It will fail, but while failing we should have fun.

Overview

- How and why kernels fail.
- Userland tools for debugging a running system.
- The assembler-level view of a C program.
- Introduction to *gdb*.
- Preparing for debugging a kernel.
- Analysing panic dumps
- Debugging a running system.
- Writing *gdb* macros.

How and why kernels fail

- Kernels fail because of bugs.
- They can *hang*.
- They can destroy themselves (overwriting code).
- They can continue running incorrectly, corrupting data or breaching network protocols.
- Failed consistency checks result in a specific `panic`:

  ```
  panic: Free vnode isn't
  ```

- Exception conditions result in a less specific `panic`:

  ```
  panic: Page fault in kernel mode
  ```

Userland programs: dmesg

- In-core copy of kernel message buffer.
- Limited in size.
- Normally copied to */var/log/messages* file.
- Can be accessed with *gdb*.

dmesg: example

```
FreeBSD 4.5-PRERELEASE #3: Sat Jan  5 13:25:02 CST 2002
    grog@echunga.lemis.com:/src/FreeBSD/4-STABLE-ECHUNGA/src/sys/compile/ECHUNGA
Timecounter "i8254"  frequency 1193182 Hz
Timecounter "TSC"  frequency 751708714 Hz
CPU: AMD Athlon(tm) Processor (751.71-MHz 686-class CPU)
  Origin = "AuthenticAMD"  Id = 0x621  Stepping = 1
  Features=0x183f9ff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,SEP,MTRR,PGE,MCA,CMOV,
PAT,PSE36,MMX,FXSR>
  AMD Features=0xc0400000<AMIE,DSP,3DNow!>
...
pci0: <unknown card> (vendor=0x1039, dev=0x0009) at 1.1
...
cd1 at ahc0 bus 0 target 1 lun 0
cd1: <TEAC CD-ROM CD-532S 1.0A> Removable CD-ROM SCSI-2 device
cd1: 20.000MB/s transfers (20.000MHz, offset 15)
cd1: Attempt to query device size failed: NOT READY, Medium not present
...
WARNING: / was not properly unmounted
```

dmesg: normal operation

```
sio1: 1 more silo overflow (total 1607)
sio1: 1 more silo overflow (total 1608)
nfsd send error 64
...
nfs server wantadilla:/src: not responding
nfs server wantadilla:/: not responding
nfs server wantadilla:/src: is alive again
nfs server wantadilla:/: is alive again
arp info overwritten for 192.109.197.82 by 00:00:21:ca:6e:f1
```

/var/log/ files

- */var/log/* is the conventional place to put log files.
- Many are maintained by *syslogd*.
- Can maintain a single file for multiple machines.
- *syslogd* messages contain two fields describing the message.
- *facility* field states what created the message.
- *level* field states how important the message is.

/etc/syslog.conf/

```
# $FreeBSD: src/etc/syslog.conf,v 1.13 2000/02/08 21:57:28 rwatson Exp $
#
#     Spaces are NOT valid field separators in this file.
#     Consult the syslog.conf(5) manpage.
*.*                                     @echunga
*.err;kern.debug;auth.notice;mail.crit          /dev/console
*.notice;kern.debug;lpr.info;mail.crit;news.err      /var/log/messages
security.*                              /var/log/security
mail.info                               /var/log/maillog
lpr.info                                /var/log/lpd-errs
cron.*                                  /var/log/cron
*.err                                   root
*.notice;news.err                       root
*.alert                                       root
*.emerg                                       *
# uncomment this to enable logging of all log messages to /var/log/all.log
#*.*                                    /var/log/all.log
# uncomment this to enable logging to a remote loghost named loghost
#*.*                                    @loghost
# uncomment these if you're running inn
# news.crit                             /var/log/news/news.crit
# news.err                              /var/log/news/news.err
# news.notice                           /var/log/news/news.notice
!startslip
*.*                                     /var/log/slip.log
!ppp
*.*                                     /var/log/ppp.log
```

Userland programs

- *ps*
- *top*
- *vmstat*
- *iostat*
- *systat*
- *ktrace*
- *netstat*

ps

- Displays process state.
- Some fields show what the process is doing:
- `WCHAN` shows where the process is sleeping.
- `STAT` shows current process status.
- flags (`F`) show process flags.
- All is revealed in the man page.

ps example

```
$ ps lax
  UID   PID  PPID CPU PRI NI    VSZ   RSS MWCHAN STAT  TT       TIME COMMAND
    0     0     0   0 -16  0      0    12 sched  DLs   ??    0:15.62  (swapper)
 1004     0 60226   0 -84  0      0     0 -      ZW    ??    0:00.00  (galeon-bin)
    0     1     0   0   8  0    708    84 wait   ILs   ??    0:14.58 /sbin/init --
    0     7     0   0 171  0      0    12 -      RL    ??   80:46.00  (pagezero)
    0     8     0   2   4  0      0    12 sbwait DL    ??    1:44.51  (bufdaemon)
    0    11     0 150 -16  0      0    12 -      RL    ??  52617:10.66  (idle)
    0    12     0   0 -44  0      0    12 -      WL    ??   39:11.32  (swi1: net)
    0    13     0   0 -48  0      0    12 -      WL    ??   43:42.81  (swi6: tty)
    0    20     0   0 -64  0      0    12 -      WL    ??    0:00.00  (irq11: ahc0)
    0    21     0  34 -68  0      0    12 Giant  LL    ??  116:10.44  (irq12: rl0)
```

- *swapper* is in short term wait (D), locked in core (L), and is session leader (s).
- *init* is waiting longer than 20 seconds (I state).

top

Example: slow `make world`

```
last pid: 79931;  load averages:  2.16,  2.35,  2.21   up 0+01:25:07  18:07:46
75 processes:  4 running, 51 sleeping, 20 waiting
CPU states: 18.5% user,  0.0% nice, 81.5% system,  0.0% interrupt,  0.0% idle
Mem: 17M Active, 374M Inact, 69M Wired, 22M Cache, 60M Buf, 16M Free
Swap: 512M Total, 512M Free

  PID USERNAME  PRI NICE   SIZE    RES STATE    TIME   WCPU    CPU COMMAND
   10 root      -16    0     0K    12K RUN     18:11  1.07%  1.07% idle
79828 root      125    0   864K   756K select   0:00  3.75%  0.83% make
    6 root       20    0     0K    12K syncer   0:35  0.20%  0.20% syncer
   19 root      -68 -187     0K    12K WAIT     0:12  0.00%  0.00% irq9: rl0
   12 root      -48 -167     0K    12K WAIT     0:08  0.00%  0.00% swi6: tty:sio
  303 root       96    0  1052K   688K select   0:05  0.00%  0.00% rlogind
```

- Load average is an indication of how busy the system is.
- Some pathological cases: thundering hordes syndrome.

vmstat

- Shows kernel activity, in particular Virtual Memory.
- Can be repeated to show current activity.

vmstat example (FreeBSD Intel)

```
procs      memory      page                    disks     faults      cpu
r b w     avm    fre   flt  re  pi  po  fr  sr ad0 ad2   in   sy  cs us sy id
1 3 0  307300  12672    15   0   0   0  90  71   0   0  542 1108 299  2  4 94
2 3 0  307300  12672     5   0   0   0   0   0   0   0 59695  534 128  1 87 11
0 3 0  307300  12672     3   0   0   0   0   0   1   0 43298  623 158  1 52 47
0 3 0  307300  12672     3   0   0   0   0   0   0   0  271  516  79  1  2 98
0 3 0  307300  12672     3   0   0   0   0   0   1   0  277  458  68  1  2 98
3 3 0  307300  12672     3   0   0   0   0   0   0   0 47332  546 128  1 68 31
4 3 0  308444  12668     3   0   0   0   1   0   0   0 84256  653 220  1 99  0
0 3 0  307832  12668     3   0   0   0   0   0   0   0 4371 12831 6280 58 28 14
0 3 0  307832  12668     3   0   0   0   0   0   0   0  234  453  67  1  1 98
0 3 0  307832  12668     4   0   0   0   0   0   0   0  242  476  74  2  1 98
0 3 0  307832  12668     3   0   0   0   0   0   0   0  274  461  70  2  1 98
1 3 0  307832  12668     3   0   0   0   0   0   0   0 73967  536 194  1 99  0
0 3 0  307832  12668     3   0   0   0   0   0   0   0  253  475  73  2 12 86
2 3 0  222468  12668     3   0   0   0   0   0   0   0 47362  540 121  1 69 30
0 3 0  222812  12668     3   0   0   0   0   0   0   0 88438  968 260  1 91  8
0 3 0  222812  12668     3   0   0   0   0   0   0   0  242  472  74  2  1 98
```

iostat

- Shows statistics about I/O activity.
- Can be repeated to show current activity.
- Can specify which devices or device categories to observe.

iostat example (OpenBSD SPARC)

```
      tty             sd0             rd0             rd1             cpu
 tin tout  KB/t t/s MB/s   KB/t t/s MB/s   KB/t t/s MB/s  us ni sy in id
   0    0  7.77   9 0.07   0.00   0 0.00   0.00   0 0.00  19  0  6  1 74
   0  222 56.00   1 0.05   0.00   0 0.00   0.00   0 0.00  69  0 29  2  0
   0   75  0.00   0 0.00   0.00   0 0.00   0.00   0 0.00  81  0 19  0  0
   0   76 32.00   1 0.03   0.00   0 0.00   0.00   0 0.00  84  0 16  0  0
   0   74  0.00   0 0.00   0.00   0 0.00   0.00   0 0.00  90  0  7  3  0
   0   74  0.00   0 0.00   0.00   0 0.00   0.00   0 0.00  95  0  5  0  0
   0   74  5.30  20 0.10   0.00   0 0.00   0.00   0 0.00  40  0 31  0 29
   0   73  6.40  51 0.32   0.00   0 0.00   0.00   0 0.00  12  0 10  3 75
   0   75  5.55  49 0.27   0.00   0 0.00   0.00   0 0.00  24  0 12  3 61
   0   73  4.91  54 0.26   0.00   0 0.00   0.00   0 0.00  21  0  9  1 69
   0   75  6.91  54 0.36   0.00   0 0.00   0.00   0 0.00  39  0  7  3 51
   0   72  9.80  49 0.46   0.00   0 0.00   0.00   0 0.00  31  0  6  4 59
   0   76 17.94  36 0.63   0.00   0 0.00   0.00   0 0.00  34  0 12  0 54
   0   75 19.20   5 0.09   0.00   0 0.00   0.00   0 0.00  93  0  5  1  1
   0   74 37.33   3 0.11   0.00   0 0.00   0.00   0 0.00  93  0  6  1  0
   0   75 56.00   1 0.06   0.00   0 0.00   0.00   0 0.00  82  0 17  1  0
   0   73  0.00   0 0.00   0.00   0 0.00   0.00   0 0.00  83  0 16  1  0
```

systat

- Shows a number of different parameters in graphical form.
- Includes *iostat*, *netstat* and *vmstat*.
- Ugly display.

systat example

```
                    /0   /1   /2   /3   /4   /5   /6   /7   /8   /9   /10
     Load Average   ||

          /0   /10  /20  /30  /40  /50  /60  /70  /80  /90  /100
cpu  user|XXXXXXXXXXXXXXXXXXXXX
     nice|
   system|XXXXX
interrupt|
     idle|XXXXXXXXXXXXXXXXXXXX

          /0   /10  /20  /30  /40  /50  /60  /70  /80  /90  /100
ad0    MB/sXXXX
       tps|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

systat vmstat, FreeBSD

```
   24 users    Load  0.85  0.25  0.15                  Sun Jan 20 14:40

Mem:KB    REAL            VIRTUAL                       VN PAGER  SWAP PAGER
        Tot   Share      Tot    Share    Free           in  out     in  out
Act  150180    3536   220116    10096   10404 count
All  252828    4808  3565340    15372         pages
                                                         zfod   Interrupts
Proc:r  p  d  s  w    Csw  Trp  Sys  Int  Sof  Flt        cow   62295 total
     2     1 24       147   14  63262294   26    6  56060 wire      1 ata0 irq14
                                                     162880 act         ata1 irq15
 1.5%Sys  98.5%Intr  0.0%User  0.0%Nice  0.0%Idl     24140 inact       ahc0 irq11
|    |    |    |    |    |    |    |    |           9748 cache    27 mux irq10
=+++++++++++++++++++++++++++++++++++++++++++         656 free      4 atkbd0 irq
                                                         daefr       psm0 irq12
Namei         Name-cache    Dir-cache                    prcfr    77 sio1 irq3
    Calls     hits    %     hits    %                    react       ppc0 irq7
                                                         pdwak    99 clk irq0
                                                         pdpgs   128 rtc irq8
Disks   ad0   ad2   cd0   cd1   sa0 pass0 pass1           intrn 61959 lpt0 irq7
KB/t   8.00  0.00  0.00  0.00  0.00  0.00  0.00     35712 buf
tps       1     0     0     0     0     0     0        27 dirtybuf
MB/s   0.01  0.00  0.00  0.00  0.00  0.00  0.00     17462 desiredvnodes
% busy    0     0     0     0     0     0     0     22916 numvnodes
                                                    17020 freevnodes
```

systat vmstat, NetBSD

```
    1 user     Load  2.74  1.91  1.60                  Thu Jan 17 14:31:09

          memory totals (in KB)             PAGING   SWAPPING      Interrupts
         real   virtual    free              in  out   in  out       132 total
Active   9868     14100    6364     ops       1                      100 irq0
All     21140     25372  658588     pages                             14 irq9
                                                                       18 irq10
Proc:r  d  s  w       Csw  Trp  Sys  Int  Sof  Flt        forks
     2  1  5           40   27  193  133   20    8        fkppw
                                                         fksvm
  95.9% Sy   1.4% Us   0.0% Ni   0.0% In   2.7% Id        pwait
|    |    |    |    |    |    |    |    |    |     6 relck
==================================================>     6 rlkok
                                                         noram
Namei         Sys-cache     Proc-cache                   ndcpy
    Calls     hits    %     hits     %                    fltcp
     1043      806   77       34     3                  1 zfod
                                                         cow
Discs  fd0  sd0  md0                                    64 fmin
seeks                                                   85 ftarg
xfers        14                                       1372 itarg
Kbyte       164                                        941 wired
%busy      21.2                                           pdfre
                                                          pdscn
```

systat vmstat, OpenBSD

```
    3 users    Load  1.19  1.52  1.81                  Thu Jan 17 14:31:48 2002

Mem:KB  REAL        VIRTUAL                    PAGING   SWAPPING     Interrupts
      Tot Share     Tot  Share   Free          in  out   in  out      227 total
Act  3348  1068   12940   6704  27016 count     2                      5 lev1
All 35232 11888 358812 148796        pages                           17 lev4
                                                                      5 lev6
Proc:r  p  d  s  w    Csw  Trp  Sys  Int  Sof  Flt      17 cow       100 clock
           2  5        29  206  184  227       374       3 objlk          lev12
                                                         2 objht     100 prof
   9.3% Sys  85.5% User   0.0% Nice   4.4% Idle          62 zfod
|    |    |    |    |    |    |    |    |    |    |    385 nzfod
=====>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  16.14 %zfod
                                                         kern
Namei         Sys-cache     Proc-cache               5408 wire
    Calls     hits    %     hits     %              18312 act
      212      203   96        3     1              11220 inact
                                                    27016 free
Discs  sd0  rd0  rd1                                      daefr
seeks  411                                            372 prcfr
xfers  411                                             46 react
Kbyte   33                                                 scan
  sec  0.1                                                 hdrev
                                                           intrn
```

ktrace

- Traces at system call interface.
- Doesn't require source code.
- Shows a limited amount of information.
- Can be useful to find which files are being opened.
- Collect a dump file with *ktrace*.
- Dump in with *kdump*.

ktrace example

```
71602 sh       NAMI  "/bin/url_handler.sh"
71602 sh       RET   stat -1 errno 2 No such file or directory
71602 sh       CALL  stat(0x80ec108,0xbfbff0b0)
71602 sh       NAMI  "/sbin/url_handler.sh"
71602 sh       RET   stat -1 errno 2 No such file or directory
71602 sh       CALL  stat(0x80ec108,0xbfbff0b0)
71602 sh       NAMI  "/usr/local/bin/url_handler.sh"
71602 sh       RET   stat -1 errno 2 No such file or directory
71602 sh       CALL  stat(0x80ec108,0xbfbff0b0)
71602 sh       NAMI  "/etc/url_handler.sh"
71602 sh       RET   stat -1 errno 2 No such file or directory
71602 sh       CALL  stat(0x80ec108,0xbfbff0b0)
71602 sh       NAMI  "/usr/X11R6/bin/url_handler.sh"
71602 sh       RET   stat -1 errno 2 No such file or directory
71602 sh       CALL  stat(0x80ec108,0xbfbff0b0)
71602 sh       NAMI  "/usr/monkey/url_handler.sh"
71602 sh       RET   stat -1 errno 2 No such file or directory
71602 sh       CALL  stat(0x80ec108,0xbfbff0b0)
71602 sh       NAMI  "/usr/local/sbin/url_handler.sh"
71602 sh       RET   stat -1 errno 2 No such file or directory
71602 sh       CALL  break(0x80f3000)
71602 sh       RET   break 0
71602 sh       CALL  write(0x2,0x80f2000,0x1a)
71602 sh       GIO   fd 2 wrote 26 bytes
      "url_handler.sh: not found
      "
71602 sh       RET   write 26/0x1a
71602 sh       CALL  exit(0x7f)
```

Hardware data structures

Most modern machines are stack oriented. Stacks contain:

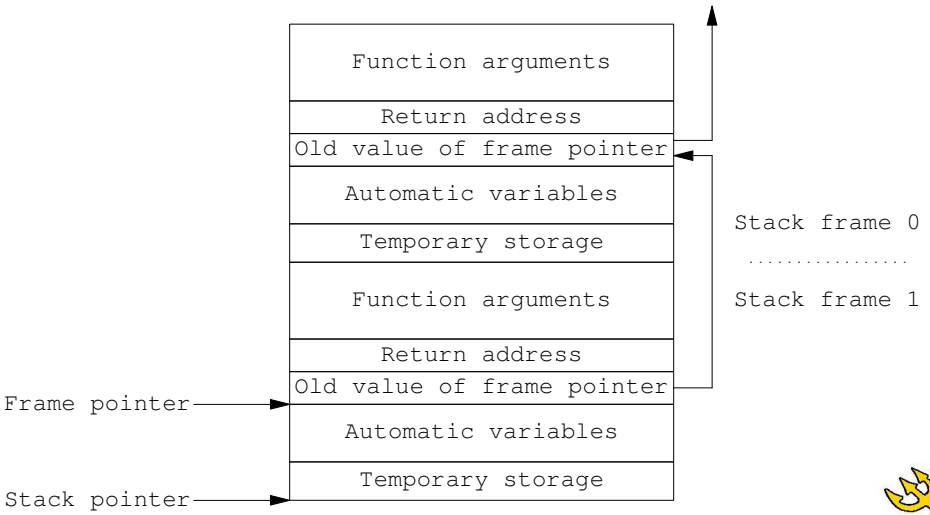
- The parameters with which the function was invoked.
- The address to which to return when the function is complete.
- Saved register contents.
- Variables local to the function.
- The address of the previous stack frame.

Stack pointers

Most architectures have similar stack structures that use two pointers:

- The *stack pointer* points to the last used word of the stack.
- The *frame pointer* points to somewhere in the middle of the stack frame.

Stack structures



Intel registers

The Intel ia32 architecture uses the following registers:

- The *Program Counter* points to the next instruction to be executed. Intel calls it the *Instruction Pointer* or `eip`.
- The *Stack Pointer* is called `esp`.
- The *Frame Pointer* is called `ebp` (*Extended Base Pointer*).
- ia32 still maintains the concept of an *accumulator*, `eax`.

Intel registers (2)

The Intel ia32 architecture uses the following registers:

- The other registers are `ebx`, `ecx` and `edx`. They have some special function, but they can be used in many arithmetic instructions as well.
- The registers `esi` (*Extended Source Index*) and `edi` (*Extended Destination Index*) are purely index registers.
- The `eflags` register contains program status information.
- The *segment registers* contain information about memory segments. Their usage depends on the mode in which the processor is running.

Function calls (1)

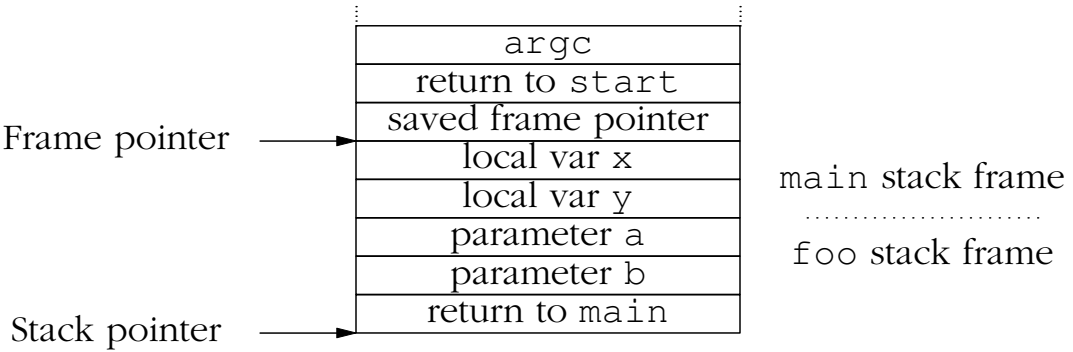
Consider this program:

```
foo (int a, int b)
{
  int c = a * b;
  int d = a / b;
  printf ("%d %d\n", c, d);
  }

main (int argc, char *argv [])
{
  int x = 4;
  int y = 5;
  foo (y, x);
  }
```

Assembler code for calling `foo` from `main` is:

```
pushl -4(%ebp)          value of x
pushl -8(%ebp)          value of y
call _foo               call the function
addl $8,%esp            and remove parameters
```

Stack frame after call



Building stack linkage

`foo` saves the frame pointer `ebp` and loads it with the current value of the stack pointer register `esp`:

```
_foo:   pushl %ebp              save ebp on stack
        movl %esp,%ebp          and load with current value of esp
```

- Now the stack linkage is complete. This is where most debuggers normally set a breakpoint on the entry to a function.
- Next, `foo` creates local storage for `c` and `d`. Each are 4 bytes long, so it subtracts 8 from the `esp` register to make space for them:

```
        subl $8,%esp            create two words on stack
```

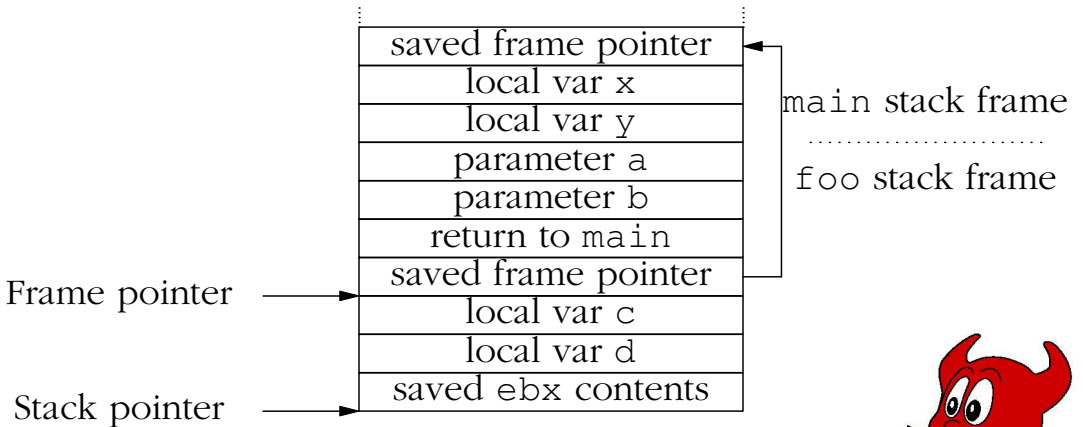
Building stack linkage (2)

Finally, `foo` saves the register `ebx`—the compiler has decided that it will need this register in this function:

```
pushl %ebx                    and save ebx register
```

Stack frame after linkage

Now the stack is complete:



Returning from call

On return from the function, the sequence is reversed:

```
movl -12(%ebp),%ebx     and restore register ebx
leave                   reload ebp and esp
ret                     and return
```

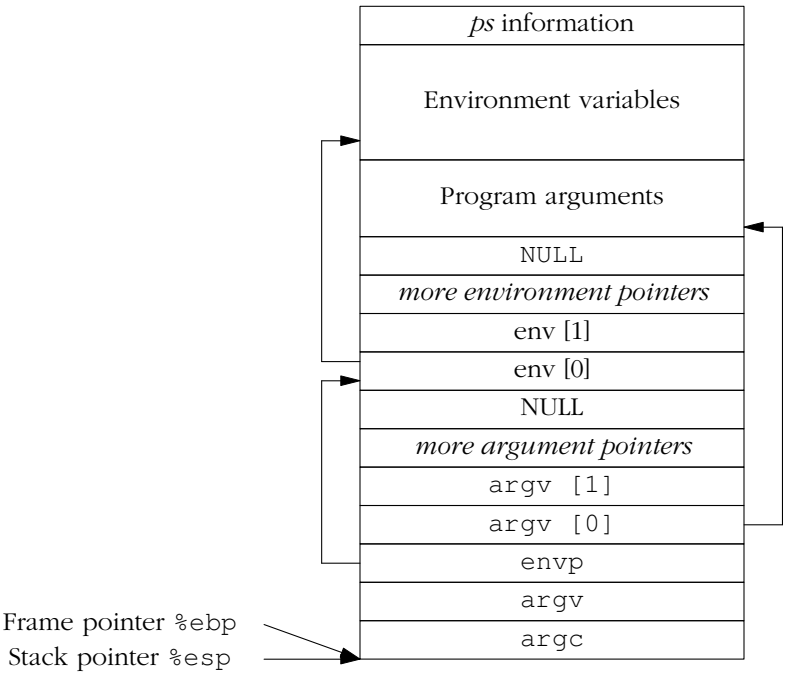
- `movl -12(%ebp),%ebx` reloads the saved register `ebx`, which could be stored anywhere in the stack. This instruction does not modify the stack.
- `leave` loads `esp` from `ebp`, discarding the stack part below the saved `ebp` value.

Returning from call (2)

- `ret` pops the return address into the instruction pointer. The next instruction will be fetched from the address following the `call` instruction.
- Parameters `x` and `y` are still on the stack, so the next instruction in the calling function removes them by adding to the stack pointer:

```
addl $8,%esp                    and remove parameters
```

Stack frame at process start



The kernel source tree

- Traditionally in */usr/src/sys/*.
- Frequently also a symlink */sys/* –> */usr/src/sys/*.
- Divided into machine-dependent and machine-independent sections.
- FreeBSD: individual architecture sources in top-level directory (*alpha*, *i386*, *ia64*, *powerpc*, *sparc64*)
- NetBSD, OpenBSD: individual architecture sources in subdirectory of */sys/arch/*.

The kernel source tree

- FreeBSD up to version 4 kernels built in directory */sys/compile/*.
- FreeBSD version 5 kernels built in directory */sys/i386/compile/*.
- NetBSD, OpenBSD kernels built in directory */sys/arch/i386/compile/*.

The kernel source tree

- Directory */sys/boot/* (FreeBSD) or */sys/stand* (NetBSD, OpenBSD) contains bootstrap code.
- Directory */sys/compat/* contains compatibility code for older versions.
- Directory */sys/conf/* contains MI code for configuring kernels.
- Directory */sys/contrib/* contains code with other licenses.
- Directory */sys/dev/* contains device driver code.

The kernel source tree

- Directory */sys/gnu/* (FreeBSD, OpenBSD) contains GPL code.
- Directory */sys/isa/* (FreeBSD) contains code for ISA bus.
- Directory */sys/kern/* contains core kernel code.
- Directory */sys/net/* contains base networking code.
- Directory */sys/netinet/* contains IPv4 code.
- Directory */sys/netinet6/* contains IPv6 code.

The kernel source tree

- Directory */sys/nfs/* contains NFS code.
- Directory */sys/pci/* (FreeBSD) contains PCI bus code.
- Directory */sys/sys/* contains header files.
- Directory */sys/ufs/* contains UFS related code.
- Directory */sys/vm/* contains VM code.

Building a debug kernel

- Require the `-g` option. Set:

```
makeoptions   DEBUG=-g            #Build kernel with gdb(1) debug symbols
```

- Alternatively, on FreeBSD, run *config* with `-g` option.
- Make kernel with `make kernel`.
- FreeBSD offers `make buildkernel` from */usr/src*. This may cause problems with the pathnames in the debug kernel.
- When using serial debug, build kernel via NFS on debugging machine.

FreeBSD kernel debug options

- `INVARIANTS` and `INVARIANT_SUPPORT` add certain debug checks.
- `DDB` enables the *ddb* debugger.
- In newer versions of FreeBSD, specify `GDB` to enable the *gdb* debugger.
- `DDB_UNATTENDED` stops entering *ddb* on a panic.
- `GDB_REMOTE_CHAT` enables sharing a single line for serial console and *gdb*.
- `KTRACE` enables support for *ktrace*.

FreeBSD kernel debug options (2)

- `DIAGNOSTIC` prints out additional debugging information.
- `REGRESSION` enables additional interfaces used for regression testing.
- `RESTARTABLE_PANICS` allows returning from a panic.
- `KTR` (FreeBSD 5.x only) enables an internal trace buffer.
- `WITNESS` (FreeBSD 5.x only) checks lock consistency.
- Other current 5.x options: `MUTEX_DEBUG`, `WITNESS_DDB`, `WITNESS_SKIPSPIN`.

Preparing for dumps

- *dumpon* tells the system where to put the dump.
- *savecore* saves the dump after reboot.

Preparing for dumps

- Require disk space for dumps, by convention on */var/crash/*.
- Dumps go initially to the swap partition.
- Use *dumpon* to specify where to write the dump.
- In */etc/rc.conf*, set:

```
dumpdev=/dev/ad0b
```

- After reboot, *savecore* saves dump to */var/crash/*
- Make a swap partition at least a little larger than main memory. Dumps must fit completely on one partition.

Serial console

- Offers better control over debugging.
- Can save info to log files even after the system has paniced.
- For FreeBSD, on debugged machine:

```
Type '?' for a list of commands, 'help' for more detailed help.
ok set console=comconsole
```

- Alternatively, set in */boot/loader.conf*:

```
console="comconsole"              # Set the current console
```

- Turn off again with

```
set console=vidconsole
```

Serial console: debugging machine

- Connect with crossover cable.
- Run *cu* or similar program.
- Use standard serial parameters, 9600 bps.
- Use the *ttydn* port.
- May need to adjust ownership or permissions of the serial port:

```
# ls -l /dev/ttyd0
crw-rw-rw-  1 root  wheel   28,   0 Nov  3 15:23 /dev/ttyd0
# ps aux | grep cu
uucp 6828  0.0  0.5  1020  640  p0  I+ 3:21PM   0:00.01 cu -s 9600 -l /dev/ttyd0
uucp 6829  0.0  0.5  1020  640  p0  I+ 3:21PM   0:00.01 cu -s 9600 -l /dev/ttyd0
```

Panic!

A call to panic produces a register summary:

```
Fatal trap 12: page fault while in kernel mode
fault virtual address   = 0x64
fault code              = supervisor read, page not present
instruction pointer     = 0x8:0xc02451d7
stack pointer           = 0x10:0xccd99a20
frame pointer           = 0x10:0xccd99a24
code segment            = base 0x0, limit 0xfffff, type 0x1b
                        = DPL 0, pres 1, def32 1, gran 1
processor eflags        = interrupt enabled, resume, IOPL = 0
current process         = 107 (syslogd)
```

ddb

- Runs entirely on debugged machine.
- Displays to console (including serial console).
- Enter from `panic` if `debugger_on_panic` set.
- `DDB_UNATTENDED` resets `debugger_on_panic`.
- Enter from keyboard with **CTRL-ALT-ESC**.

ddb: entry from keyboard

```
# Debugger("manual escape to debugger")
Stopped at      Debugger+0x44:  pushl   %ebx
db> t
Debugger(c03ca5e9) at Debugger+0x44
scgetc(c16d9800,2,c16d1440,c046ac60,0) at scgetc+0x426
sckbdevent(c046ac60,0,c16d9800,c16d1440,c16d4300) at sckbdevent+0x1c9
atkbd_intr(c046ac60,0,cc04bd18,c024c79a,c046ac60) at atkbd_intr+0x22
atkbd_isa_intr(c046ac60) at atkbd_isa_intr+0x18
ithread_loop(c16d4300,cc04bd48,c16d4300,c024c670,0) at ithread_loop+0x12a
fork_exit(c024c670,c16d4300,cc04bd48) at fork_exit+0x58
fork_trampoline() at fork_trampoline+0x8db>
db>
```

ddb: entry on panic

(continuing from panic message above)

```
kernel: type 12 trap, code=0
Stopped at      devsw+0x7:      cmpl    $0,0x64(%ebx)
db> tr
devsw(0,c045cd80,cc066e04,cc066e04,0) at devsw+0x7
cn_devopen(c045cd80,cc066e04,0) at cn_devopen+0x27
cnopen(c0435ec8,6,2000,cc066e04,0) at cnopen+0x39
spec_open(ccd99b50,ccd99b24,c0320589,ccd99b50,ccd99bc4) at spec_open+0x127
spec_vnoperate(ccd99b50,ccd99bc4,c029984b,ccd99b50,ccd99d20) at spec_vnoperate+0x15
ufs_vnoperatespec(ccd99b50,ccd99d20,0,cc066e04,6) at ufs_vnoperatespec+0x15
vn_open(ccd99c2c,ccd99bf8,0,cc066f0c,cc066d00) at vn_open+0x333
open(cc066e04,ccd99d20,8054000,bfbfef64,bfbfef34) at open+0xde
syscall(2f,2f,2f,bfbfef34,bfbfef64) at syscall+0x24c
syscall_with_err_pushed() at syscall_with_err_pushed+0x1b
--- syscall (5, FreeBSD ELF, open), eip = 0x280aae50, esp = 0xbfbfe960,
ebp = 0xbfbfe9cc ---
```

Kernel gdb

- Uses the same *gdb* program you know and love in userland.
- Provides symbolic capability.
- Provides macro language capability.
- Some modifications to adapt to kernel dump structure.
- Requires `-k` flag when used on kernel dumps.
- Can also run across serial lines.
- Not a very good fit to kernel.
- Can be very slow due to serial line speed.

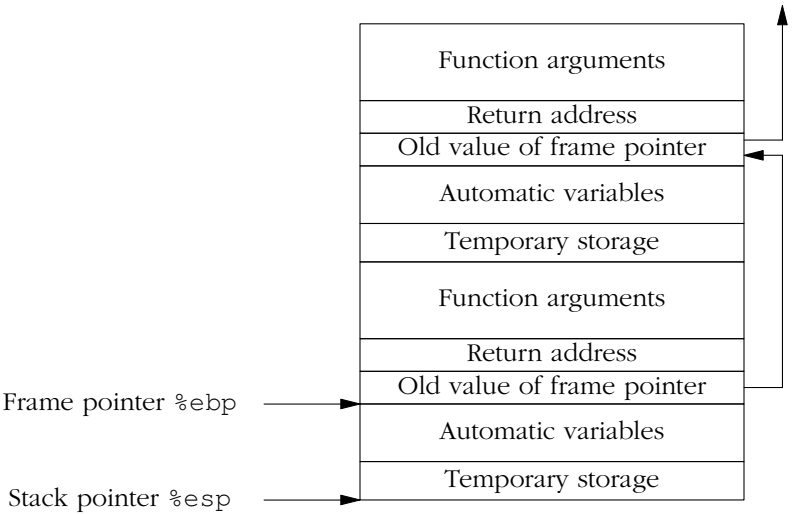
Enter gdb from ddb

```
db> gdb
Next trap will enter GDB remote protocol mode
db>

Fatal trap 12: page fault while in kernel mode
fault virtual address   = 0x64
fault code              = supervisor read, page not present
instruction pointer     = 0x8:0xc02451d7
stack pointer           = 0x10:0xccd99a20
frame pointer           = 0x10:0xccd99a24
code segment            = base 0x0, limit 0xfffff, type 0x1b
                        = DPL 0, pres 1, def32 1, gran 1
processor eflags        = trace trap, interrupt enabled, resume, IOPL = 0
current process         = 107 (syslogd)
||||$T0b08:d75124c0;05:249ad9cc;04:209ad9cc;#32~.

Disconnected.
#
```

Running serial gdb

```
$ cd /usr/src/sys/compile/CANBERRA
$ gdb
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-unknown-freebsd".
(kgdb) tr
devsw (dev=0x0) at ../../../kern/kern_conf.c:83
83              if (dev->si_devsw)
(kgdb)
```

Stack frames



Running serial gdb: backtrace

```
(kgdb) bt
#0  devsw (dev=0x0) at ../../../kern/kern_conf.c:83
#1  0xc027d0c7 in cn_devopen (cnd=0xc045cd80, td=0xcc066e04, forceopen=0x0)
    at ../../../kern/tty_cons.c:344
#2  0xc027d211 in cnopen (dev=0xc0435ec8, flag=0x6, mode=0x2000, td=0xcc066e04)
    at ../../../kern/tty_cons.c:376
#3  0xc0230f6f in spec_open (ap=0xccd99b50) at ../../../fs/specfs/spec_vnops.c:199
#4  0xc0230e45 in spec_vnoperate (ap=0xccd99b50) at
    ../../../fs/specfs/spec_vnops.c:119
#5  0xc0320589 in ufs_vnoperatespec (ap=0xccd99b50) at
    ../../../ufs/ufs_vnops.c:2676
#6  0xc029984b in vn_open (ndp=0xccd99c2c, flagp=0xccd99bf8, cmode=0x0) at
    vnode_if.h:159
#7  0xc0294c12 in open (td=0xcc066e04, uap=0xccd99d20) at
    ../../../kern/vfs_syscalls.c:1099
#8  0xc035aedc in syscall (frame={tf_fs = 0x2f, tf_es = 0x2f, tf_ds = 0x2f,
    tf_edi = 0xbfbfef34, tf_esi = 0xbfbfef64, tf_ebp = 0xbfbfe9cc,
    tf_isp = 0xccd99d74, tf_ebx = 0x8054000, tf_edx = 0xf7, tf_ecx = 0x805402f,
    tf_eax = 0x5, tf_trapno = 0x0, tf_err = 0x2, tf_eip = 0x280aae50,
    tf_cs = 0x1f, tf_eflags = 0x293, tf_esp = 0xbfbfe960, tf_ss = 0x2f})
    at ../../../i386/i386/trap.c:1129
#9  0xc034c28d in syscall_with_err_pushed ()
#10 0x804b2b5 in ?? ()
#11 0x804abe9 in ?? ()
#12 0x804b6fe in ?? ()
#13 0x804b7af in ?? ()
#14 0x8049fb5 in ?? ()
#15 0x8049709 in ?? ()
(kgdb)
```

Assembler-level view

Consider the following function:

```
void
exec_unmap_first_page(imgp)
        struct image_params *imgp;
{
        GIANT_REQUIRED;

        if (imgp->firstpage) {
                pmap_kremove((vm_offset_t) imgp->image_header);
                vm_page_unwire(imgp->firstpage, 1);
                imgp->firstpage = NULL;
        }
}
```

- One parameter, `imgp`, at `0x8(%ebp)`
- Macro `GIANT_REQUIRED` is not immediately obvious.

Assembler-level view

Build up new stack frame:

```
0xc024935c <exec_unmap_first_page>:     push   %ebp
0xc024935d <exec_unmap_first_page+1>:   mov    %esp,%ebp
```

Assembler-level view

Save registers:

```
0xc024935c <exec_unmap_first_page>:     push   %ebp
0xc024935d <exec_unmap_first_page+1>:   mov    %esp,%ebp
0xc024935f <exec_unmap_first_page+3>:   push   %ebx
```

Assembler-level view

Function body:

```
0xc024935c <exec_unmap_first_page>:     push   %ebp
0xc024935d <exec_unmap_first_page+1>:   mov    %esp,%ebp
0xc024935f <exec_unmap_first_page+3>:   push   %ebx
0xc0249360 <exec_unmap_first_page+4>:   mov    0x8(%ebp),%ebx
0xc0249363 <exec_unmap_first_page+7>:   push   $0x224
0xc0249368 <exec_unmap_first_page+12>:  push   $0xc03aa8fb
0xc024936d <exec_unmap_first_page+17>:  push   $0x1
0xc024936f <exec_unmap_first_page+19>:  push   $0xc0477a20
0xc0249374 <exec_unmap_first_page+24>:  call   0xc0252f74 <_mtx_assert>
0xc0249379 <exec_unmap_first_page+29>:  add    $0x10,%esp
0xc024937c <exec_unmap_first_page+32>:  cmpl   $0x0,0x7c(%ebx)
0xc0249380 <exec_unmap_first_page+36>:  je     0xc024939b <exec_unmap_first_page+63>
0xc0249382 <exec_unmap_first_page+38>:  pushl  0x10(%ebx)
0xc0249385 <exec_unmap_first_page+41>:  call   0xc0358afc <pmap_kremove>
0xc024938a <exec_unmap_first_page+46>:  push   $0x1
0xc024938c <exec_unmap_first_page+48>:  pushl  0x7c(%ebx)
0xc024938f <exec_unmap_first_page+51>:  call   0xc032c14c <vm_page_unwire>
0xc0249394 <exec_unmap_first_page+56>:  movl   $0x0,0x7c(%ebx)
```

Assembler-level view

Function exit:

```
0xc024935c <exec_unmap_first_page>:      push   %ebp
0xc024935d <exec_unmap_first_page+1>:    mov    %esp,%ebp
0xc024935f <exec_unmap_first_page+3>:    push   %ebx
0xc0249360 <exec_unmap_first_page+4>:    mov    0x8(%ebp),%ebx
0xc0249363 <exec_unmap_first_page+7>:    push   $0x224
0xc0249368 <exec_unmap_first_page+12>:   push   $0xc03aa8fb
0xc024936d <exec_unmap_first_page+17>:   push   $0x1
0xc024936f <exec_unmap_first_page+19>:   push   $0xc0477a20
0xc0249374 <exec_unmap_first_page+24>:   call   0xc0252f74 <_mtx_assert>
0xc0249379 <exec_unmap_first_page+29>:   add    $0x10,%esp
0xc024937c <exec_unmap_first_page+32>:   cmpl   $0x0,0x7c(%ebx)
0xc0249380 <exec_unmap_first_page+36>:   je     0xc024939b <exec_unmap_first_page+63>
0xc0249382 <exec_unmap_first_page+38>:   pushl  0x10(%ebx)
0xc0249385 <exec_unmap_first_page+41>:   call   0xc0358afc <pmap_kremove>
0xc024938a <exec_unmap_first_page+46>:   push   $0x1
0xc024938c <exec_unmap_first_page+48>:   pushl  0x7c(%ebx)
0xc024938f <exec_unmap_first_page+51>:   call   0xc032c14c <vm_page_unwire>
0xc0249394 <exec_unmap_first_page+56>:   movl   $0x0,0x7c(%ebx)
0xc024939b <exec_unmap_first_page+63>:   mov    0xfffffffc(%ebp),%ebx
0xc024939e <exec_unmap_first_page+66>:   leave
0xc024939f <exec_unmap_first_page+67>:   ret
```

Debugging running systems

- Use */dev/mem* instead of dump file.
- Can't set breakpoints.
- Things can change while you're looking at them.

Debugging running FreeBSD system

```
# gdb -k /isr/src/sys/i386//MONORCHID/kernel.debug /dev/mem
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-unknown-freebsd"...
IdlePTD at phsyical address 0x004f3000
initial pcb at physical address 0x0e5ccda0
panic messages:
---
---
#0  0xc023a6df in mi_switch () at ../../../kern/kern_synch.c:779
779             cpu_switch();
(kgdb) bt
#0  0xc023a6df in mi_switch () at ../../../kern/kern_synch.c:779
#1  0xffffffff in ?? ()
error reading /proc/95156/mem
```

Debugging running NetBSD system

```
# gdb /netbsd
GNU gdb 4.17
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386--netbsd"...(no debugging symbols found)...
(gdb) target kcore /dev/mem
#0  0xc01a78f3 in mi_switch ()
(gdb) bt
#0  0xc01a78f3 in mi_switch ()
#1  0xc01a72ca in ltsleep ()
#2  0xc02d6c81 in uvm_scheduler ()
#3  0xc019a358 in check_console ()
(gdb)
```

Once again, with symbols

```
# gdb /usr/src/sys/arch/i386/compile/KIMCHI/netbsd.gdb
GNU gdb 4.17 Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386--netbsd"...
(gdb) target kcore /dev/mem
#0  mi_switch (p=0xc0529be0) at ../../../../kern/kern_synch.c:834
834             microtime(&p->p_cpu->ci_schedstate.spc_runtime);
(gdb) bt
#0  mi_switch (p=0xc0529be0) at ../../../../kern/kern_synch.c:834
#1  0xc01a72ca in ltsleep (ident=0xc0529be0, priority=4, wmesg=0xc04131e4
    "scheduler", timo=0, interlock=0x0) at ../../../../kern/kern_synch.c:.482
#2  0xc02d6c81 in uvm_scheduler () at ../../../../uvm/uvm_glue.c:453
#3  0xc019a358 in check_console (p=0x0) at
    ../../../../kern/init_main.c:522
```

Dump analysis with gdb

```
# cd /var/crash
# ls -l
total 661
-rw-r--r--  1 root  wheel          3 Sep 20 11:12 bounds
-rw-r--r--  1 root  wheel    3464574 Sep 16 06:13 kernel.10
-rw-r--r--  1 root  wheel    3589033 Sep 18 09:08 kernel.11
-rw-r--r--  1 root  wheel    3589033 Sep 19 03:13 kernel.12
-rw-r--r--  1 root  wheel    3589033 Sep 20 10:50 kernel.13
-rw-r--r--  1 root  wheel    3589033 Sep 20 11:03 kernel.14
-rw-r--r--  1 root  wheel    3589033 Sep 20 11:12 kernel.15
lrwxr-xr-x  1 root  wheel         61 Sep 20 16:13 kernel.debug
   -> /src/FreeBSD/4.4-RELEASE/src/sys/compile/ECHUNGA/kernel.debug
-rw-r--r--  1 root  wheel          5 Sep 17  1999 minfree
-rw-------  1 root  wheel  134152192 Sep 18 09:08 vmcore.11
-rw-------  1 root  wheel  134152192 Sep 19 03:13 vmcore.12
-rw-------  1 root  wheel  134152192 Sep 20 10:50 vmcore.13
-rw-------  1 root  wheel  134152192 Sep 20 11:03 vmcore.14
-rw-------  1 root  wheel  134152192 Sep 20 11:12 vmcore.15
# cd /src/FreeBSD/4.4-RELEASE/src/sys/compile/ECHUNGA
# gdb -k /var/crash/kernel.debug /var/crash/vmcore.12
```

Dump analysis with gdb

Let's look at this dump:

```
panicstr: general protection fault
panic messages:
---
Fatal trap 9: general protection fault while in kernel mode
instruction pointer     = 0x8:0xc01c434b
stack pointer           = 0x10:0xc99f8d0c
frame pointer           = 0x10:0xc99f8d28
code segment            = base 0x0, limit 0xfffff, type 0x1b
                        = DPL 0, pres 1, def32 1, gran 1
processor eflags        = interrupt enabled, resume, IOPL = 0
current process         = 2638 (find)
interrupt mask          = net tty bio cam
trap number             = 9
panic: general protection fault

syncing disks... 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
giving up on 6 buffers
Uptime: 17h53m13s

dumping to dev #ad/1, offset 786560
dump ata0: resetting devices .. done
---
#0  dumpsys () at ../../kern/kern_shutdown.c:473
473             if (dumping++) {
(kgdb)
```

Dump analysis with gdb

First, get a backtrace:

```
(kgdb) bt
#0  dumpsys () at ../../kern/kern_shutdown.c:473
#1  0xc01c88bf in boot (howto=256) at ../../kern/kern_shutdown.c:313
#2  0xc01c8ca5 in panic (fmt=0xc03a8cac "%s") at ../../kern/kern_shutdown.c:581
#3  0xc033ab03 in trap_fatal (frame=0xc99f8ccc, eva=0)
    at ../../i386/i386/trap.c:956
#4  0xc033a4ba in trap (frame={tf_fs = 16, tf_es = 16, tf_ds = 16,
      tf_edi = -1069794208, tf_esi = -1069630360, tf_ebp = -912290520,
      tf_isp = -912290568, tf_ebx = -1069794208, tf_edx = 10, tf_ecx = 10,
      tf_eax = -1, tf_trapno = 9, tf_err = 0, tf_eip = -1071889589, tf_cs = 8,
      tf_eflags = 66182, tf_esp = 1024, tf_ss = 6864992})
    at ../../i386/i386/trap.c:618
#5  0xc01c434b in malloc (size=1024, type=0xc03c3c60, flags=0)
    at ../../kern/kern_malloc.c:233
#6  0xc01f015c in allocbuf (bp=0xc3a6f7cc, size=1024)
    at ../../kern/vfs_bio.c:2380
#7  0xc01effa6 in getblk (vp=0xc9642f00, blkno=0, size=1024, slpflag=0,
    slptimeo=0) at ../../kern/vfs_bio.c:2271
#8  0xc01eded2 in bread (vp=0xc9642f00, blkno=0, size=1024, cred=0x0,
    bpp=0xc99f8e3c) at ../../kern/vfs_bio.c:504
#9  0xc02d0634 in ffs_read (ap=0xc99f8ea0) at ../../ufs/ufs/ufs_readwrite.c:273
#10 0xc02d734e in ufs_readdir (ap=0xc99f8ef0) at vnode_if.h:334
```

Dump analysis with gdb

Stack trace, continued:

```
#11 0xc02d7cd1 in ufs_vnoperate (ap=0xc99f8ef0)
    at ../../ufs/ufs/ufs_vnops.c:2382
#12 0xc01fbc3b in getdirentries (p=0xc9a53ac0, uap=0xc99f8f80)
    at vnode_if.h:769
#13 0xc033adb5 in syscall2 (frame={tf_fs = 47, tf_es = 47, tf_ds = 47,
      tf_edi = 134567680, tf_esi = 134554336, tf_ebp = -1077937404,
      tf_isp = -912289836, tf_ebx = 672064612, tf_edx = 134554336,
      tf_ecx = 672137600, tf_eax = 196, tf_trapno = 7, tf_err = 2,
      tf_eip = 671767876, tf_cs = 31, tf_eflags = 582, tf_esp = -1077937448,
      tf_ss = 47}) at ../../i386/i386/trap.c:1155
#14 0xc032b825 in Xint0x80_syscall ()
#15 0x280a1eee in ?? ()
#16 0x280a173a in ?? ()
#17 0x804969e in ?? ()
#18 0x804b550 in ?? ()
#19 0x804935d in ?? ()
(kgdb)
```

Dump analysis with gdb

The most important stack frame is the one below `trap`. Select it with the `frame` command, abbreviated to `f`, and list the code with `list` (or `l`):

```
(kgdb) f 5
#5  0xc01c434b in malloc (size=1024, type=0xc03c3c60, flags=0)
    at ../../kern/kern_malloc.c:233
233             va = kbp->kb_next;
(kgdb) l
228                     }
229                     freep->next = savedlist;
230                     if (kbp->kb_last == NULL)
231                             kbp->kb_last = (caddr_t)freep;
232             }
233             va = kbp->kb_next;
234             kbp->kb_next = ((struct freelist *)va)->next;
235     #ifdef INVARIANTS
236             freep = (struct freelist *)va;
237             savedtype = (const char *) freep->type->ks_shortdesc;
(kgdb)
```

Dump analysis with gdb

You might want to look at the local (automatic) variables. Use `info local`, which you can abbreviate to `i loc`:

```
(kgdb) i loc
type = (struct malloc_type *) 0xc03c3c60
kbp = (struct kmembuckets *) 0xc03ebc68
kup = (struct kmemusage *) 0x0
freep = (struct freelist *) 0x0
indx = 10
npg = -1071714292
allocsize = -1069794208
s = 6864992
va = 0xffffffff <Address 0xffffffff out of bounds>
cp = 0x0
savedlist = 0x0
ksp = (struct malloc_type *) 0xffffffff
(kgdb)
```

Dump analysis with gdb

The line where the problem occurs is 233:

```
233             va = kbp->kb_next;
```

Look at the structure `kbp`:

```
(kgdb) p *kbp
$2 = {
  kb_next = 0xffffffff <Address 0xffffffff out of bounds>,
  kb_last = 0xc1a31000 "",
  kb_calls = 83299,
  kb_total = 1164,
  kb_elmpercl = 4,
  kb_totalfree = 178,
  kb_highwat = 20,
  kb_couldfree = 3812
}
```

Dump analysis with gdb

So far we have found:

- Crash in `malloc`.
- Probably not a bug in `malloc`.
- What do we do now?

gdb macros

- gdb includes a macro language.
- Syntax is reminiscent of C, but different enough to be confusing.
- No good reference.

gdb macros: an example

```
(kgdb) ps
  pid    proc     addr   uid  ppid  pgrp   flag stat comm         wchan
 2638 c9a53ac0 c99f7000     0  2624  2402  004004  2  find
 2626 c9980f20 c99b0000     0  2614  2402  004084  3  sort          piperd c95d2cc0
 2625 c9a53440 c9a94000     0  2614  2402  004084  3  xargs         piperd c95d3080
 2624 c9a53780 c9a7d000     0  2614  2402  000084  3  sh            wait c9a53780
 2616 c9a535e0 c9a72000     0  2615  2402  004184  3  postdrop      piperd c95d2e00
 2615 c997e1a0 c9a4d000     0  2612  2402  004084  3  sendmail      piperd c95d3b20
 2614 c9a53e00 c9a41000     0  2612  2402  004084  3  sh            wait c9a53e00
 2612 c997f860 c99e8000     0  2413  2402  004084  3  sh            wait c997f860
 2437 c9a53c60 c9a54000     0  2432  2432  004184  3  postdrop      piperd c95d34e0
 2432 c997e340 c9a1d000     0  2400  2432  004084  3  sendmail      piperd c95d31c0
 2415 c997eb60 c9a21000     0  2414  2402  004084  3  cat           piperd c95d3760
 2414 c997f1e0 c99f2000     0  2404  2402  000084  3  sh            wait c997f1e0
 2413 c997e9c0 c9a30000     0  2404  2402  000084  3  sh            wait c997e9c0
 2404 c997e4e0 c9a38000     0  2402  2402  004084  3  sh            wait c997e4e0
```

gdb macros: an example

```
define ps
    set $nproc = nprocs
    set $aproc = allproc.lh_first
    set $proc = allproc.lh_first
    printf "  pid    proc    addr   uid  ppid  pgrp   flag stat comm          wchan\n"
    while (--$nproc >= 0)
        set $pptr = $proc.p_pptr
        if ($pptr == 0)
           set $pptr = $proc
        end
        if ($proc.p_stat)
            printf "%5d %08x %08x %4d %5d %5d  %06x  %d  %-10s   ", \
                   $proc.p_pid, $aproc, \
                   $proc.p_addr, $proc.p_cred->p_ruid, $pptr->p_pid, \
                   $proc.p_pgrp->pg_id, $proc.p_flag, $proc.p_stat, \
                   &$proc.p_comm[0]
            if ($proc.p_wchan)
                if ($proc.p_wmesg)
                    printf "%s ", $proc.p_wmesg
                end
                printf "%x", $proc.p_wchan
            end
            printf "\n"
        end
        set $aproc = $proc.p_list.le_next
        set $proc = $aproc
    end
end
```

The End

- For more information, RTFM.
- If there is no FM, WTFM.
- The latest version of the course notes are at *http://www.lemis.com/grog/Papers/Debug-tutorial/tutorial.pdf*.
- The latest version of these slides are at *http://www.lemis.com/grog/Papers/Debug-tutorial/slides.pdf*.
- Notes from this course should go into the FreeBSD handbook.
- Volunteers required!