Deciphering Oopsies

Borislav Petkov <bp@suse.de>

Kernel monster

```
$ cloc linux-2.6/
   48422 text files.
   46327 unique files.
   14332 files ignored.

http://cloc.sourceforge.net v 1.60  T=1611.04 s (23.3 files/s, 10225.6 lines/s)
```

Language	files	blank	comment	code
C	19322	1811546	1772226	9339536
C/C++ Header	15491	361715	627639	1990938
Assembly	1398	45063	58843	330431
XML	155	3209	283	46350
make	893	5620	5063	24396
Perl	38	3622	2858	16793
Bourne Shell	86	937	1864	5139
yacc	7	559	342	3577
ASP.Net	2	129	0	3061
Python	22	658	398	2956
lex	7	258	253	1575
C++	1	208	57	1525
awk	8	89	88	720
Bourne Again Shell	33	122	80	701
NAnt scripts	1	96	0	383
HTML	2	58	0	378
D	3	0	0	296
Pascal	3	49	0	231
Lisp	1	63	0	218
Objective C++	1	55	0	189
ASP	1	33	0	137
m4	1	15	1	96
XSLT	6	13	27	70
sed	1	0	3	30
vim script	1	3	12	27
Teamcenter def	1	0	2	6
SUM:	37485	2234120	2470039	11769759

The Linux Kernel

- Huge monster
- Monsters are also likely to have bugs!
- Monsters can be easily disturbed and are generally cranky
- Make a fuss and barf an oops

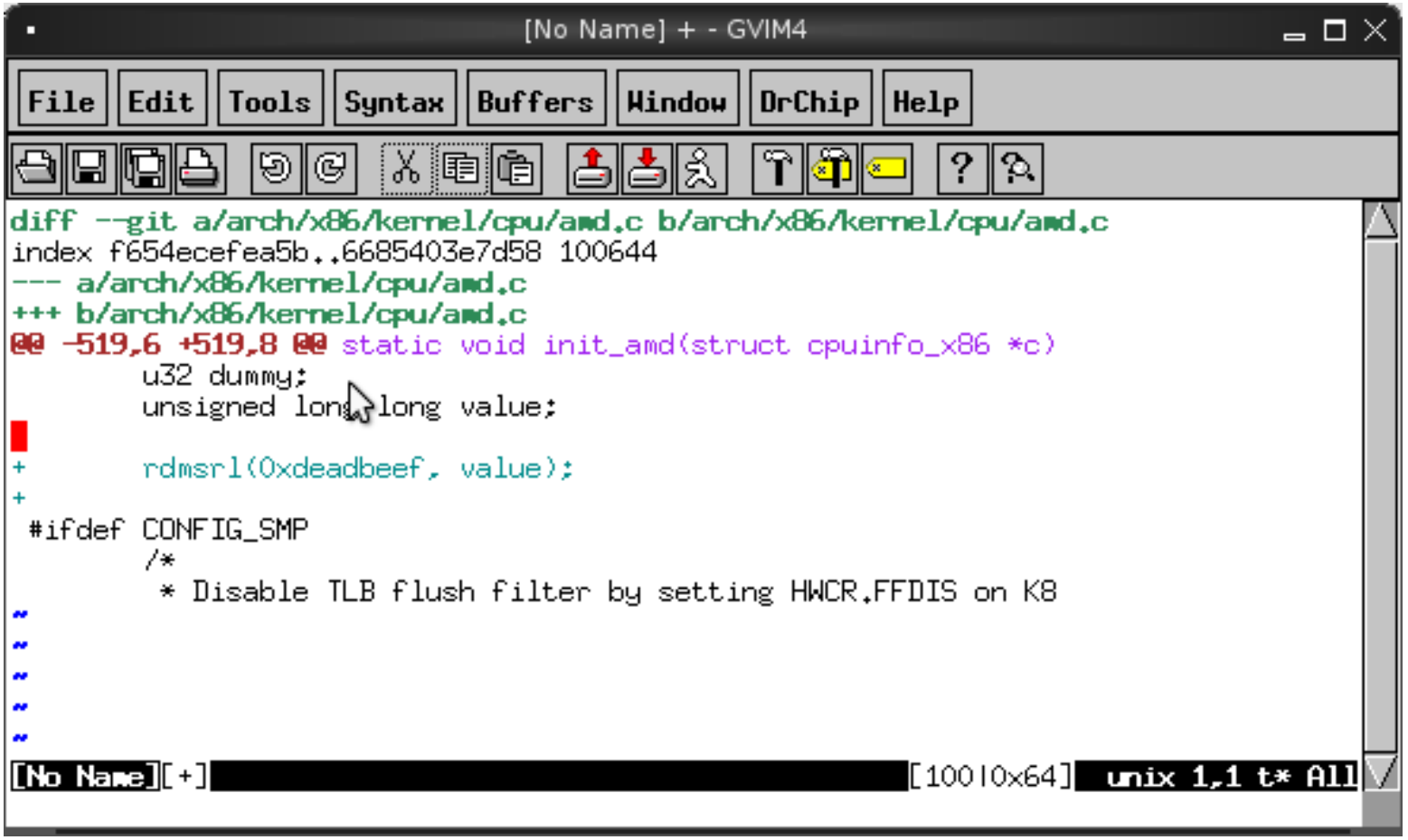
What do you need to know

- Linus: “*The main trick is having 5 years of experience with those pesky oops messages ;-)*”
- Documentation/oops-tracing.txt
- That's a good start +
  - patience
  - coffee
  - :-)

oops.kernel.org

- Arjan's kerneloops tool
- Apparently we're resuscitating the site
- Some preprocessing already there

Let's do a nice one



```
general protection fault: 0000 [#1] PREEMPT SMP
Modules linked in:
CPU: 0 PID: 0 Comm: swapper/0 Not tainted 3.11.0-rc3+ #4
Hardware name: Bochs Bochs, BIOS Bochs 01/01/2011
task: ffffffff81a10440 ti: ffffffff81a00000 task.ti: ffffffff81a00000
RIP: 0010:[<ffffffff81015aaa>]  [<ffffffff81015aaa>] init_amd+0x1a/0x640
RSP: 0000:ffffffff81a01ed8  EFLAGS: 00010296
RAX: ffffffff81015a90 RBX: 0000000000726f73 RCX: 00000000deadbeef
RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffffffff81aadf00
RBP: ffffffff81a01f18 R08: 0000000000000000 R09: 0000000000000001
R10: 0000000000000001 R11: 0000000000000000 R12: ffffffff81aadf00
R13: ffffffff81b572e0 R14: ffff88007ffd8400 R15: 0000000000000000
FS:  0000000000000000(0000) GS:ffff88007fc00000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: ffff88000267c000 CR3: 0000000001a0b000 CR4: 00000000000006b0
Stack:
 ffffffff817cda76 0000000000000001 0000001000000001 0000000000000000
 ffffffff81a01f18 0000000000726f73 ffffffff81aadf00 ffffffff81b572e0
 ffffffff81a01f38 ffffffff81014260 ffffffffffffffff ffffffff81b50020
Call Trace:
 [<ffffffff81014260>] identify_cpu+0x2d0/0x4d0
 [<ffffffff81ad53b9>] identify_boot_cpu+0x10/0x3c
 [<ffffffff81ad5409>] check_bugs+0x9/0x2d
 [<ffffffff81acfe31>] start_kernel+0x39d/0x3b9
 [<ffffffff81acf894>] ? repair_env_string+0x5a/0x5a
 [<ffffffff81acf5a6>] x86_64_start_reservations+0x2a/0x2c
 [<ffffffff81acf699>] x86_64_start_kernel+0xf1/0xf8
Code: 00 0f b6 33 eb 8f 66 66 2e 0f 1f 84 00 00 00 00 00 e8 2b 2b 4e 00 55 b9 ef be ad de 48 89
e5 41 55 41 54 49 89 fc 53 48 83 ec 28 <0f> 32 80 3f 0f 0f 84 13 02 00 00 4c 89 e7 e8 03 fd ff
ff f0 41
RIP  [<ffffffff81015aaa>] init_amd+0x1a/0x640
 RSP <ffffffff81a01ed8>
---[ end trace 3c9ee0eeb6dd208c ]---
Kernel panic - not syncing: Fatal exception
```

So why did we explode?

- Generally, the first line says why:

```
general protection fault: 0000 [#1] PREEMPT SMP
```

- 0000: error code
- [#1]: die_counter
- flags

Modules

[0.016000] Modules linked in:

- Currently loaded modules
- Taint crap, forced unload, OOT
- My favorite: TAINT_PROPRIETARY_MODULE

Taint line

CPU: 0 PID: 0 Comm: swapper/0 Not tainted 3.11.0-rc3+ #2

```
/**
 *      print_tainted - return a string to represent the kernel taint state.
 *
 *  'P' - Proprietary module has been loaded.
 *  'F' - Module has been forcibly loaded (MODVERSIONS and .vermagic)
 *  'S' - SMP with CPUs not designed for SMP (Some K7 Athlons)
 *  'R' - User forced a module unload.
 *  'M' - System experienced a machine check exception.
 *  'B' - System has hit bad_page.
 *  'U' - Userspace-defined naughtiness requested the user.
 *  'D' - Kernel has oopsed before
 *  'A' - ACPI table overridden.
 *  'W' - Taint on warning.
 *  'C' - modules from drivers/staging are loaded.
 *  'I' - Working around severe firmware bug.
 *  'O' - Out-of-tree module has been loaded.
 *
 *      The string is overwritten by the next call to print_tainted().
 */
```

Hardware name:

`Hardware name: Bochs Bochs, BIOS Bochs 01/01/2011`

- dump_stack_set_arch_desc – arch-specific system identifier
- Usually DMI ID on x86, early in setup_arch()
- DMI sucks, btw:

  `”To be filled by O.E.M. To be filled by O.E.M./M5A97 EVO R2.0, BIOS 1503 01/16/2013”`

thread_info

```
task: ffffffff81a10440 ti: ffffffff81a00000 task.ti: ffffffff81a00000
```

- Task: *current* points to, i.e. struct task_struct
- Thread info: located at bottom of the process stack (on x86 stack grows downwards)

rIP

RIP: 0010:[<ffffffff81015aaa>] [<ffffffff81015aaa>] init_amd+0x1a/0x640

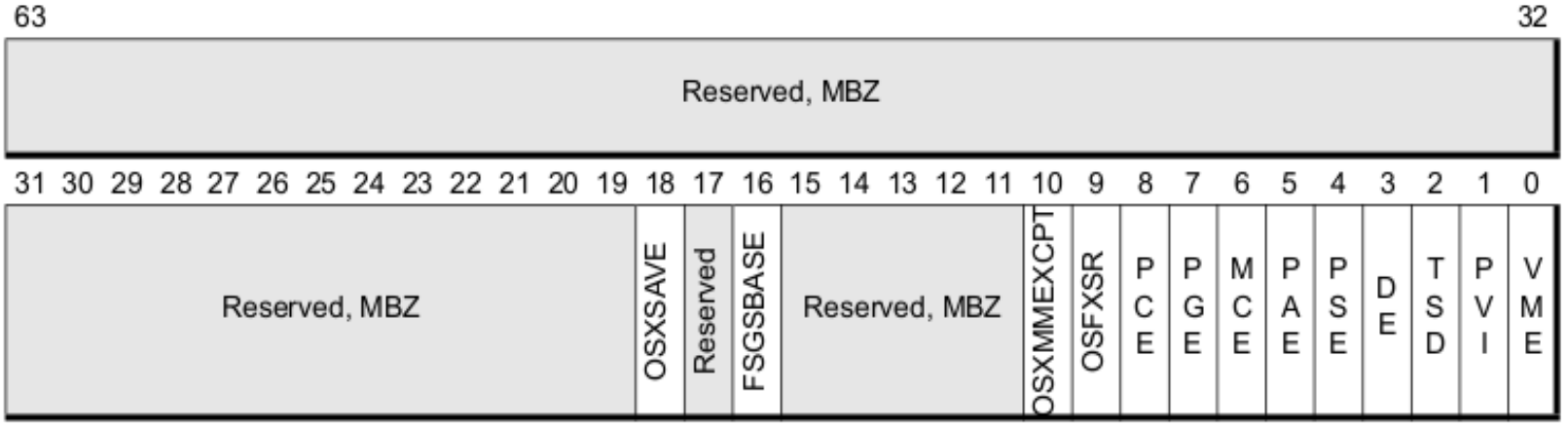
- RIP (Instruction Pointer): the virtual address of the instruction which caused the #GP
- 0010: CS[15:3] → GDT_ENTRY_KERNEL_CS
- Kallsyms: table compression, ratio ~ 50%, see scripts/kallsyms.c
- Call chain is:
  - `printk_address → printk(“...%pB”, ...) → … → sprint_backtrace(buffer, address);`

Register snapshot

```
RSP: 0000:ffffffff81a01ed8  EFLAGS: 00010296
RAX: ffffffff81015a90 RBX: 0000000000726f73 RCX: 00000000deadbeef
RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffffffff81aadf00
RBP: ffffffff81a01f18 R08: 0000000000000000 R09: 0000000000000001
R10: 0000000000000001 R11: 0000000000000000 R12: ffffffff81aadf00
R13: ffffffff81b572e0 R14: ffff88007ffd8400 R15: 0000000000000000
FS:  0000000000000000(0000) GS:ffff88007fc00000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: ffff88000267c000 CR3: 0000000001a0b000 CR4: 00000000000006b0
```

- Look at RCX
- SS (Stack Segment): 0000 – null segment prot checks
- FS.base (MSR) and (%fs-index) segment override
- GS.base (MSR) and (%gs-index) segment (percpu data)
- knlGS for SWAPGS
- CRn – control registers

Lookup register definitions



Bits	Mnemonic	Description	Access Type
63:19	—	Reserved	Reserved, MBZ
18	OSXSAVE	XSAVE and Processor Extended States Enable Bit	R/W
17	—	Reserved	Reserved, MBZ
16	FSGSBASE	Enable RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE instructions	R/W
15:11	—	Reserved	Reserved, MBZ
10	OSXMMEXCPT	Operating System Unmasked Exception Support	R/W
9	OSFXSR	Operating System FXSAVE/FXRSTOR Support	R/W
8	PCE	Performance-Monitoring Counter Enable	R/W
7	PGE	Page-Global Enable	R/W
6	MCE	Machine Check Enable	R/W
5	PAE	Physical-Address Extension	R/W
4	PSE	Page Size Extensions	R/W
3	DE	Debugging Extensions	R/W
2	TSD	Time Stamp Disable	R/W
1	PVI	Protected-Mode Virtual Interrupts	R/W
0	VME	Virtual-8086 Mode Extensions	R/W

objdump

```
0000000000003e0 <init_amd>:
 3e0:   e8 00 00 00 00          callq  3e5 <init_amd+0x5>
 3e5:   55                      push   %rbp
 3e6:   b9 ef be ad de          mov    $0xdeadbeef,%ecx
 3eb:   48 89 e5                mov    %rsp,%rbp
 3ee:   41 55                   push   %r13
 3f0:   41 54                   push   %r12
 3f2:   49 89 fc                mov    %rdi,%r12
 3f5:   53                      push   %rbx
 3f6:   48 83 ec 28             sub    $0x28,%rsp
 3fa:   0f 32                   rdmsr
```

- x86-64 ABI: %rbx,%rbp,%r12-r15 callee-saved
- Reorder insns for better parallelism

Stack:

```
ffffffff817cda76 0000000000000001 0000001000000001 0000000000000000
ffffffff81a01f18 0000000000726f73 ffffffff81aadf00 ffffffff81b572e0
ffffffff81a01f38 ffffffff81014260 ffffffffffffffff ffffffff81b50020
```

- Iterate until !(THREAD_SIZE-1)
- Walk it only in kernel mode
- “<EOI>” to denote end of IRQ stack

Call Trace:

```
[<ffffffff81014260>] identify_cpu+0x2d0/0x4d0
[<ffffffff81ad53b9>] identify_boot_cpu+0x10/0x3c
[<ffffffff81ad5409>] check_bugs+0x9/0x2d
[<ffffffff81acfe31>] start_kernel+0x39d/0x3b9
[<ffffffff81acf894>] ? repair_env_string+0x5a/0x5a
[<ffffffff81acf5a6>] x86_64_start_reservations+0x2a/0x2c
[<ffffffff81acf699>] x86_64_start_kernel+0xf1/0xf8
...
RIP  [<ffffffff81015aaa>] init_amd+0x1a/0x640
 RSP <ffffffff81a01ed8>
```

- How we ended up at this rIP
- “?” - not on current stack frame
- Traverse all stacks in this order: exception <EOE>, IRQ <EOI>, process stack

Code:

```
00 0f b6 33 eb 8f 66 66 2e 0f 1f 84 00 00 00 00 00 e8 2b 2b 4e 00 55 b9 ef be ad
de 48 89 e5 41 55 41 54 49 89 fc 53 48 83 ec 28 <0f> 32 80 3f 0f 0f 84 13 02 00
00 4c 89 e7 e8 03 fd ff ff f0 41
```

- Only in kernel mode too
- <> marker points to the insn
- scripts/decodecode is your best friend here
- Let's objdump rIP:

```
ffffffff81015a90:       e8 2b 2b 4e 00          callq  ffffffff814f85c0 <__fentry__>
ffffffff81015a95:       55                      push   %rbp
ffffffff81015a96:       b9 ef be ad de          mov    $0xdeadbeef,%ecx
ffffffff81015a9b:       48 89 e5                mov    %rsp,%rbp
ffffffff81015a9e:       41 55                   push   %r13
ffffffff81015aa0:       41 54                   push   %r12
ffffffff81015aa2:       49 89 fc                mov    %rdi,%r12
ffffffff81015aa5:       53                      push   %rbx
ffffffff81015aa6:       48 83 ec 28             sub    $0x28,%rsp
ffffffff81015aaa:       0f 32                   rdmsr
```

scripts/decodecode

```
All code
========
   0:   00 0f                   add    %cl,(%rdi)
   2:   b6 33                   mov    $0x33,%dh
   4:   eb 8f                   jmp    0xffffffffffffff95
   6:   66 66 2e 0f 1f 84 00    data32 nopw %cs:0x0(%rax,%rax,1)
   d:   00 00 00 00
  11:   e8 2b 2b 4e 00          callq  0x4e2b41
  16:   55                      push   %rbp
  17:   b9 ef be ad de          mov    $0xdeadbeef,%ecx
  1c:   48 89 e5                mov    %rsp,%rbp
  1f:   41 55                   push   %r13
  21:   41 54                   push   %r12
  23:   49 89 fc                mov    %rdi,%r12
  26:   53                      push   %rbx
  27:   48 83 ec 28             sub    $0x28,%rsp
  2b:*  0f 32                   rdmsr            <-- trapping instruction
  2d:   80 3f 0f                cmpb   $0xf,(%rdi)
  30:   0f 84 13 02 00 00       je     0x249
  36:   4c 89 e7                mov    %r12,%rdi
  39:   e8 03 fd ff ff          callq  0xfffffffffffffd41
  3e:   f0                      lock
  3f:   41                      rex.B
```

Caveats

- Compiler optimizations
- Alternatives patching
- RIP points in modules space

Verify with source

- make arch/x86/kernel/cpu/amd.s

```
init_amd:
.LFB1536:
.loc 1 518 0
.cfi_startproc
call    __fentry__
.LVL82:
pushq   %rbp    #
.LCFI24:
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
.LBB418:
.LBB419:
.loc 3 64 0
movl    $-559038737, %ecx  #, tmp281
.LBE419:
.LBE418:
.loc 1 518 0
movq    %rsp, %rbp #,
.LCFI25:
.cfi_def_cfa_register 6
pushq   %r13    #
pushq   %r12    #
movq    %rdi, %r12 # c, c
.cfi_offset 12, -32
.cfi_offset 13, -24
pushq   %rbx    #
subq    $40, %rsp  #,
.LBB421:
.LBB420:
.loc 3 64 0
#APP
# 64 "/home/boris/kernel/linux-2.6/arch/x86/include/asm/msr.h" 1
rdmsr
# 0 "" 2
#NO_APP
```

```
mov $0xdeadbeef, %ecx
```

.lst

- make arch/x86/kernel/cpu/amd.lst

```
static void init_amd(struct cpuinfo_x86 *c)
{
ffffffff81015a90:   e8 00 00 00 00         callq  ffffffff81015a95
<init_amd+0x5>
ffffffff81015a91: R_X86_64_PC32  __fentry__+0xfffffffffffffffc
ffffffff81015a95:   55                     push   %rbp
ffffffff81015a96:   b9 ef be ad de         mov    $0xdeadbeef,%ecx
ffffffff81015a9b:   48 89 e5               mov    %rsp,%rbp
ffffffff81015a9e:   41 55                  push   %r13
ffffffff81015aa0:   41 54                  push   %r12
ffffffff81015aa2:   49 89 fc               mov    %rdi,%r12
ffffffff81015aa5:   53                     push   %rbx
ffffffff81015aa6:   48 83 ec 28            sub    $0x28,%rsp
ffffffff81015aaa:   0f 32                  rdmsr
```

Fixing

- Fully understood, coherent view of the issue
- Patch, build, test (in kvm, if possible), send out, CC: -stable
- Live fix: ksplice

Further info

- Documentation/oops-tracing.txt
- LDD v3, chapter 4: “Debugging Techniques”
- LKML, pestering people :)
- Processor manuals and specs
- …
- …