



Linux Kernel Debugging

"Oops", Now What?


Ross Mikosh & James Washer

IBM

Linux Technology
Center



Outline

- 
- Types of Problems
 - Tools
 - Error and Debug Messages
 - Handling Failures
 - Kernel Investigation
 - Handling a System Crash
 - Oops Analysis Example
 - LKCD/Lcrash
 - More Information



Tools



- Debuggers

- ▶ gdb
- ▶ kdb
- ▶ others?

- Built-In

- ▶ Oops data upon a panic/crash

- Dump Facility

- ▶ Linux[®] Kernel Crash Dump - lkcd

- Linux Trace Toolkit

- ▶ ltt

- Custom Kernel Instrumentation

- ▶ dprobes

- Special console functions

- ▶ Magic SysReq key





Error/Debug Messages

- System error logs
 - ▶ /var/log/*
 - ▶ dmesg
- Syslog
- Console
- Application or Module debug level





Handling Failures

■ System Crash

- ▶ Collect and analyze oops/panic data
- ▶ Collect and analyze dump with lkcd

■ System Hang

- ▶ Use Magic SysReq key
- ▶ NMI invoking a dump using lkcd
- ▶ S/390 - invoke a stand-alone dump





Kernel Investigation

- Debuggers

- ▶ Gdb and /proc/kcore
- ▶ Remote kernel debugging - gdb & serial connection
- ▶ Kdb

- Lcrash on running system

- Adding printk's in the kernel

- Programming a debug module or a new /proc file

- Appropriate for customer/production environment ?





Handling a System Crash

- Occurs when a critical system failure is detected
- Kernel routines call die()
 - ▶ Attempts to report/record system state
 - ▶ Information is limited
- Better to have an entire system memory dump
 - ▶ LKCD project on SourceForge
 - ▶ Thorough analysis and investigation can be done





Panic/Oops Analysis

■ Steps

- ▶ Collect oops output, System.map, /proc/ksyms, vmlinux, /proc/modules
- ▶ Use ksymoops to interpret oops
 - Instructions is /usr/src/linux/Documentation/oops-tracing.txt
 - Ksymoops(8) man page
 - Be Careful...

■ Brief analysis

- ▶ Ksymoops disassembles the code section
- ▶ The EIP points to the failing instruction
- ▶ The call trace section shows how you got there
 - Caution: Noise on the stack?

■ How to find failing line of code?



Oops Example

Unable to handle kernel NULL pointer dereference at virtual address 00000000

c2483069 <--- EIP (Instruction Pointer or Program Counter)

*pde = 00000000

Oops: 0000

CPU: 0

EIP: 0010:[ipv6:__insmod_ipv6_O/lib/modules/2.4.10-4GB/kernel/net/ipv6
ipv6+-472895383/96]

EFLAGS: 00010283

eax: db591f98 ebx: de2aeb60 ecx: de2aeb80 edx: c2483060

esi: 00000c00 edi: d41d0000 ebp: db591f5c esp: db591f4c

ds: 0018 es: 0018 ss: 0018

Process cat (pid: 1986, stackpage=db591000)

Stack: c012ca65 000001f0 ffffffff 00000000 00001000 c014e878 d41d0000 db591f98
00000000 00000c00 db591f94 00000000 de2aeb60 ffffffff 00000000 00001000
deae6f40 00000000 00000000 00000000 c01324d6 de2aeb60 0804db50 00001000

Call Trace: [__alloc_pages+65/452] [proc_file_read+204/420] [sys_read+146/200]
[system_call+51/64]

Code: a1 00 00 00 00 50 68 10 31 48 c2 e8 67 38 c9 fd 31 c0 89 ec



Ksymoops output

Using defaults from ksymoops -t elf32-i386 -a i386

Code; 00000000 Before first symbol

00000000 <_EIP>:

Code; 00000000 Before first symbol

0: a1 00 00 00 00 mov 0x0,%eax

Code; 00000004 Before first symbol

5: 50 push %eax

Code; 00000006 Before first symbol

6: 68 10 31 48 c2 push \$0xc2483110

Code; 0000000a Before first symbol

b: e8 67 38 c9 fd call fdc93877

<_EIP+0xfdc93877> fdc93876 <END_OF_CODE+1e1fa3d8/????>

Code; 00000010 Before first symbol

10: 31 c0 xor %eax,%eax

Code; 00000012 Before first symbol

12: 89 ec mov %ebp,%esp





/proc/ksyms Output

Memory Addr	Symbol	[Module Name]
c2483060	test_read_proc	[test]
c2483000	__insmod_test_O/home/ross/prog/test.o_M3	[test]
c2483110	__insmod_test_S.rodata_L68	[test]
c2483060	__insmod_test_S.text_L176	[test]
c2483080	foo	[test]
de79c340	ip6_frag_mem	[ipv6]
de783d00	addrconf_del_ifaddr	[ipv6]
de78a5bc	ipv6_packet_init	[ipv6]
de78fd70	ipv6_sock_mc_drop	[ipv6]
de781ee4	ip6_call_ra_chain	[ipv6]

- EIP of c2483069 is within the routine test_read_proc in module [test]
- Next, disassemble the module test.o and find the instruction with the offset 9
 - (EIP) - (Base addr of routine)
 - c2483069 - c2483060 = 9



Failing Line of Code

- Excerpt from "objdump -D test.o "

test.o: file format elf32-i386

Disassembly of section .text:

00000000 <test_read_proc>:

0:	55	push	%ebp
1:	89 e5	mov	%esp,%ebp
3:	83 ec 08	sub	\$0x8,%esp
6:	83 c4 f8	add	\$0xfffffffff8,%esp
9:	a1 00 00 00 00	mov	0x0,%eax
e:	50	push	%eax
f:	68 00 00 00 00	push	\$0x0

- C Source Code

```
int test_read_proc(char *buf, char **start, off_t offset, int count, int *eof, void *data)
```

```
{    int *ptr;    ptr=0;    printk("%d\n",*ptr);    return 0; }
```



LKCD System Dump

■ Prework (don't wait until you've had an event)

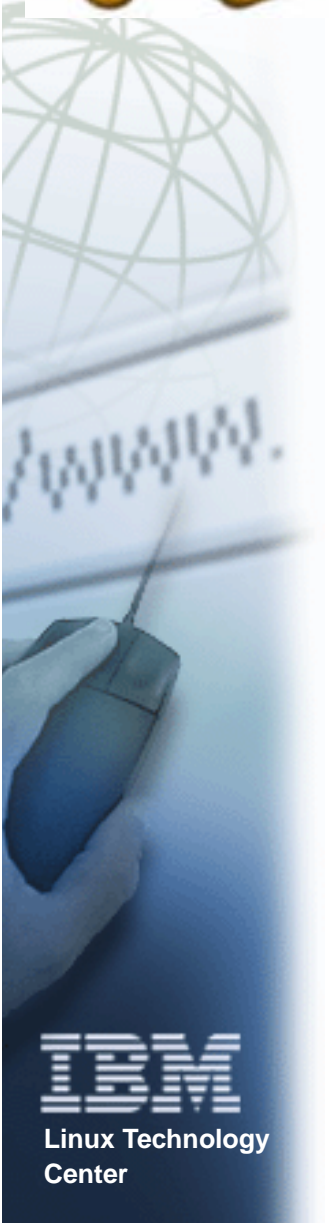
- ▶ Apply kernel patches
- ▶ Configure dump device
 - Dedicated device vs. swap device
- ▶ See tutorial for specific steps

■ Dump invocation

- ▶ Call to panic()
- ▶ Magic SysReq 'c' key
- ▶ Can be nondisruptive
- ▶ NMI (Non maskable interrupt)

■ Analysis preparation

- ▶ Copy dump to filesystem
- ▶ Collect System.map, Kerntypes



IBM

Linux Technology
Center



Lcrash Tool

- Use lcrash to analyze

- ▶ Interactive command oriented tool
- ▶ Can run on a "live" system

- Useful subcommands

- ▶ report
 - Display system summary, dmesg log, task list, and stack trace of failing task
- ▶ bt -f
 - Display back trace of a task
- ▶ task -f
 - Display tasks/processes at the time of the dump
 - print structure statement
 - `print (*((struct task_struct *)0xtask_addr))`



Dump Analysis

- Sample output of "bt" (backtrace)

STACK TRACE FOR TASK: 0xc02fc000 (swapper)

```
0 dump_execute+110 [0xc01bc6ae]
1 dump_execute+93 [0xc01bc69d]
2 panic+144 [0xc0112b80]
3 handle_sysrq+187 [0xc018b1fb] <-- dump invoked
4 handle_scancode+376 [0xc0189ac4]
5 handle_kbd_event+264 [0xc018aaf4]
6 keyboard_interrupt+23 [0xc018ab5f]
7 handle_IRQ_event+75 [0xc01084bf]
8 do_IRQ+161 [0xc01086a1]
9 do_IRQ+161 [0xc01086a1]
```





Dump Analysis (continued)

- Excerpt from "report" lcrash command... shows dmesg log

.....

<6>SysRq: <0>Kernel panic: sysrq

<0>In interrupt handler - not syncing

<4>dump: Dumping to device 0x805 [sd(8,5)] on CPU 0

<4>Dump compression value is 0x0 ...

<4>Writing dump header ...

<4>Writing dump pages ...





For More Information

- IBM Global Services Linux Support Line
 - ▶ Howto/Usage and Defect Support
 - ▶ <http://ibm.com/linux/support>
- Linux Device Drivers, 2nd Edition, Alessandro Rubini and Jonathan Corbet, O'Reilly Publishers
- LKCD - Linux Kernel Crash Dump
 - ▶ <http://lkcd.sourceforge.net/>
- Dprobes
 - ▶ <http://oss.software.ibm.com/developer/opensource/linux/projects/dprobes/>
- Linux Trace Toolkit
 - ▶ <http://www.opersys.com/LTT/index.html>





More Information

- Magic SysReq - /usr/src/linux/Documentation/sysrq.txt
- User Mode Linux
 - ▶ <http://user-mode-linux.sourceforge.net/>
- IKD - Integrated Kernel Debugger
 - ▶ <ftp://ftp.kernel.org/pub/linux/kernel/people/andrea/ikd/>
- KGDB - Remote gdb via serial connection
 - ▶ <http://kgdb.sourceforge.net/>
- KDB - Built-in Kernel Debugger
 - ▶ <http://oss.sgi.com/projects/kdb/>





Acknowledgments

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- The IBM logo is a registered trademark of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, or service names may be trademarks or service marks of others.



Linux Technology
Center