

# Linux Kernel Debugging

*Your kernel just oopsed - What do you do,  
hotshot?*

Muli Ben-Yehuda

`mulix@mulix.org`

IBM Haifa Research Lab

# Kernel Debugging - Why?

- Why would we want to debug the kernel? after all, it's the one part of the system that we never have to worry about, because it always works.
- Well, no.

# Kernel Debugging - Why?(cont)

- Because a driver is not working as well as it should, or is not working at all.
- Because we have a school or work project.
- Because the kernel is crashing, and we don't know why.
- Because we want to learn how the kernel works.
- Because it's fun! Real men hack kernels ;-)

# Broad Overview of the Kernel

- Over a million lines of code.
  - Documentation/
  - drivers/
  - kernel/
  - arch/
  - fs/
  - lib/
  - mm/
  - net/
  - Others: security/ include/ sound/ init/ usr/  
crypto/ ipc/

# Broad Kernel Overview (cont)

- Supports runtime loading and unloading of additional code (kernel modules).
- Configured using Kconfig, a domain specific configuration language.
- Built using kbuild, a collection of complex Makefiles.
- Heavily dependant on gcc and gccisms. Does *not* use or link with user space libraries, although supplies many of them - sprintf, memcpy, strlen, printk (not printf!).

# Read the Source, Luke

- The source is there - use it to figure out what's going on.
- Linux kernel developers frown upon binary only modules, because they don't have the source and thus cannot debug them.
- Later kernels include facilities to mark when a binary only module has been loaded (“tainted kernels”). Kernel developers will kindly refuse to help debug a problem when a kernel has been tainted.

# Read the Source, Luke (cont)

Use the right tools for the job. Tools to navigate the source include:

- lxr - <http://www.iglu.org.il/lxr/>
- find and grep
- ctags, etags, gtags and their ilk.

Use a good IDE

- emacs
- vi
- One brave soul I heard about used MS Visual Studio!

# Use the source

The two oldest and most useful debugging aids are

- Your brain.
- `printf`.

Use them! the kernel gives you `printk`, which

- Can be called from interrupt context.
- Behaves mostly like `printf`, except that it doesn't support floating point.

# Use the Source (cont)

Use something like this snippet to turn printk's on and off depending on whether you're building a debug or release build.

```
#ifdef DEBUG_FOO

#define CDBG(msg, args...) do {                                     \
    printk(KERN_DEBUG "[%s] " msg , __func__ , ##args );\
} while (0)

#else /* !defined(DEBUG_FOO) */

#define CDBG(msg, args...) do {} while (0)

#endif /* !defined(DEBUG_FOO) */
```

# Use the Source (cont)

- For really tough bugs, write code to solve bugs. Don't be afraid to insert new kernel modules to monitor or affect your primary development focus.
- **Code defensively.** Whenever you suspect memory overwrites or use after free, use memory poisoning.
- Enable all of the kernel debug options - they will find your bugs for you!
- `#define assert(x) do { if (!(x)) BUG(); } while (0)`
- Linux 2.5 has `BUG_ON()`.

# Kernel Debuggers

Linux has several kernel debuggers, none of which are in the main tree (for the time being). The two most common are

- kdb - <http://oss.sgi.com/projects/kdb>
- kgdb - <http://kgdb.sourceforge.net/>

# KGDB

- Requires two machines, a slave and a master.
- gdb runs on the master, controlling a gdb stub in the slave kernel via the serial port.
- When an OOPS or a panic occurs, you drop into the debugger.
- Very very useful for the situations where you dump core in an interrupt handler and no oops data makes it to disk - you drop into the debugger with the correct backtrace.

# ksymoops

- Read Documentation/oops-tracing.txt
- Install ksymoops, available from <ftp://ftp.il.kernel.org>
- Run it on the oops (get it from the logs, serial console, or copy from the screen).
- ksymoops gives you a human readable back trace.
- Sometimes the oops data can be trusted ("easy" bugs like a NULL pointer dereference) and sometimes it's no more than a general hint to what is going wrong (memory corruption overwrite EIP).

# ksymoops(cont)

- Linux 2.5 includes an "in kernel" oops tracer, called kksymoops. Don't forget to enable it when compiling your new 2.5 kernel!
- It can be found under Kernel Hacking -> Load all symbols for debugging/kksymoops (CONFIG\_KALLSYMS).

# ksymoops(cont)

Unable to handle kernel NULL pointer dereference at virtual address 00000000

printing eip:

c014a9cc

\*pde = 00000000

Oops: 0002

CPU: 0

EIP: 0060:[<c014a9cc>] Not tainted

EFLAGS: 00010202

EIP is at sys\_open+0x2c/0x90

eax: 00000001 ebx: 00000001 ecx: ffffffff edx: 00000000

esi: bffffaec edi: ce07e000 ebp: cdbcffbc esp: cdbcffb0

ds: 007b es: 007b ss: 0068

Process cat (pid: 862, threadinfo=cdbce000 task=cdcf7380)

Stack: bffffaec 40013020 bffff9b4 cdbce000 c010adc7 bffffaec 00008000 00000000

40013020 bffff9b4 bffff868 00000005 0000007b 0000007b 00000005 40013020

00000073 00000246 bffff848 0000007b

Call Trace:

[<c010adc7>] syscall\_call+0x7/0xb

Code: 89 1d 00 00 00 00 e8 59 fc ff ff 89 c6 85 f6 78 2f 8b 4d 10

# LKCD

- LKCD - Linux Kernel Crash Dump
- <http://lkcd.sf.net>
- Saves a dump of the system's state at the time the dump occurs.
- A dump occurs when the kernel panics or oopses, or when requested by the administrator.
- Must be configured before the crash occurs!

# Making sense of kernel data

- `System.map` - kernel function addresses
- `/proc/kcore` - image of system memory
- `vmlinux` - the uncompressed kernel, can be disassembled using `objdump(1)`.

# User Mode Linux

- For some kinds of kernel development (architecture independent, file systems, memory management), using UML is a life saver.
- Allows you to run the Linux kernel in user space, and debug it with gdb.
- Work is underway at making valgrind work on UML, which is expected to find many bugs.

# Magic SysRq

- More info at [Documentation/sysrq.txt](#).
- a 'magical' key combo you can hit which the kernel will respond to regardless of whatever else it is doing, unless it is completely locked up.
- `CONFIG_MAGIC_SYSRQ`, echo "1" > /proc/sys/kernel/sysrq
- On x86, press 'ALT-SysRq-<command key>'. The sysrq key is also known as the 'Print Screen' key.

# Magic SysRq(cont)

- 'b' - Will immediately reboot the system without syncing or unmounting your disks.
- 'o' - Will shut your system off (if configured and supported).
- 's' - Will attempt to sync all mounted filesystems.
- 'p' - Will dump the current registers and flags to your console.
- 't' - Will dump a list of current tasks and their information to your console.
- 'm' - Will dump current memory info to your console.
- 'h' - The most important key - will display help ;-)

# Happy Hacking!

Questions? Comments?  
Happy Oopsing!