A Survey Of Some Linux Kernel Debugging Techniques

ELC 2013

San Francisco

Kevin Dankwardt, Ph.D.   K Computing

Topics

- Web search
- Documentation
- The Code
- printk
- GDB
- Kernel Configuration
- Virtual Filesystems
- Systemtap

Web Search

- Search engines, (everyone's go to approach, I know)
- Sometimes an article on the principle involved gives you the knowledge you need to deduce the problem
- This slide mostly for a obvious reminder ...

Documentation

- The Documentation directory contains *lots* of awesome information.
- cd Documentation; find . | wc -l ---> 2494 on Linux 3.7.1
- find . -type f -exec wc -l {} \; | awk '{ sum += $1;} END {print "sum = ", sum;}' → 436753
- 436753 / 60 → about 7,279 pages of (often challenging, but informative and entertaining) kernel related information.
- make pdfdocs
- make mandocs
- make installmandocs
- make help

The Code

- make cscop
- make tags
- grep -r

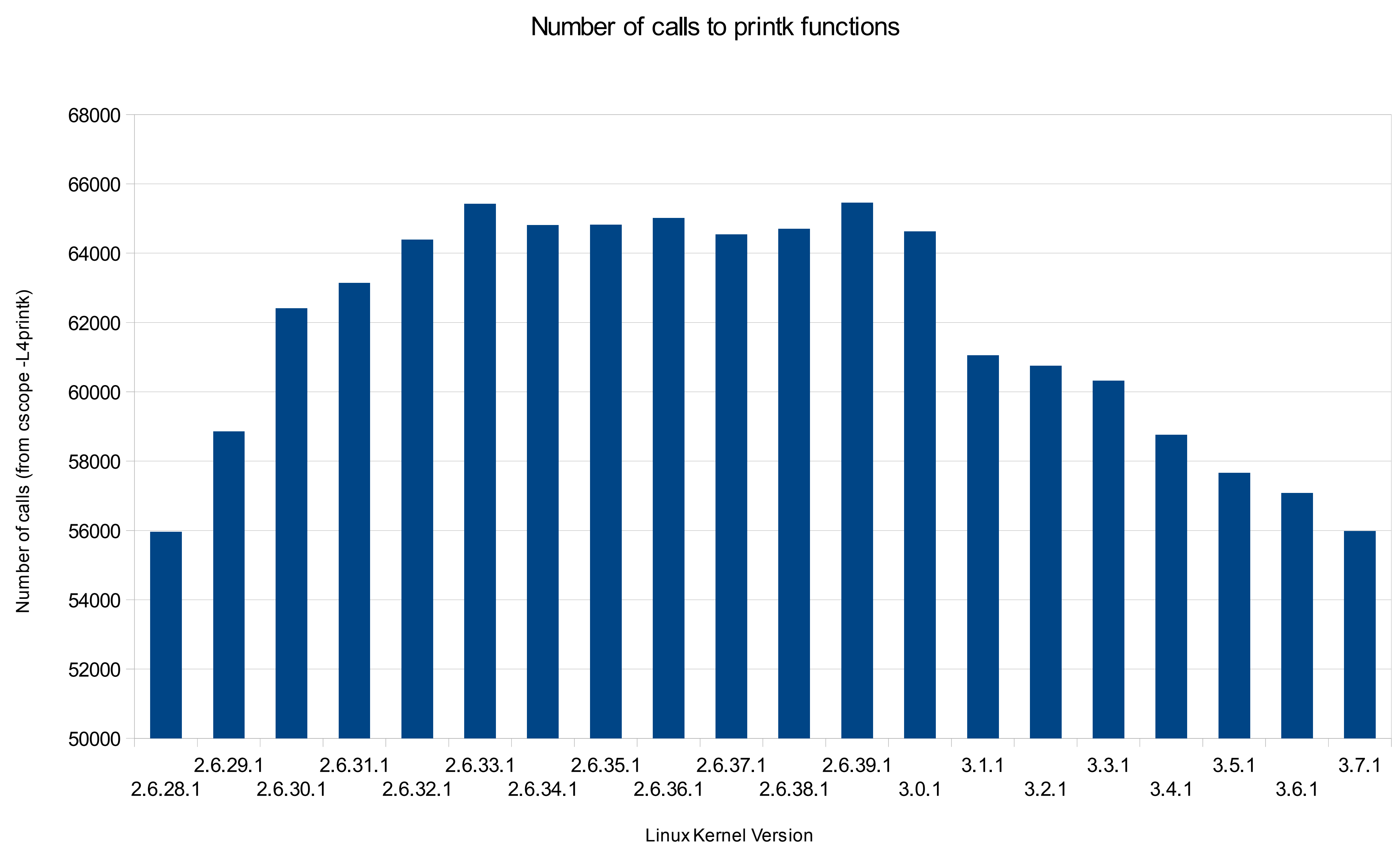
printk()

- Like `printf()`, but prints to console and log file
  - tail -f /var/log/messages
- May include kernel log level
- The most common debugging technique is to simply print a message.
- There is no comma between log level and string

Example:

```
printk(KERN_INFO “current process id= %d\n”, current->pid);
```

How Many Times Is Printk Called?



Printk Extras

- **Documentation/printk-formats.txt**
  - formats for
    - usual int, long, unsigned long, etc
    - Symbols/Function pointers with offsets
    - Kernel pointers that should be hidden from unprivileged users
    - Buffers as hex strings, MAC/FDDI addresses, IPV4, IPV6 addresses
    - UUID/GUID addresses (little endian or big endian)
- **Preface the printk output with the time**
- **syslog()/klogctl() - control of the kernel printk buffer.**
  - read, clear, disable, enable, set level, number of chars in buffer, size of buffer

```
#include <sys/klog.h>
#include <stdio.h>
int main ()
{
    printf("size of buffer = %d\n",klogctl(10,0,0));
    return 0;
}
```

Log Levels

- Log levels are defined in `<linux/kernel.h>`
- Log Levels include:

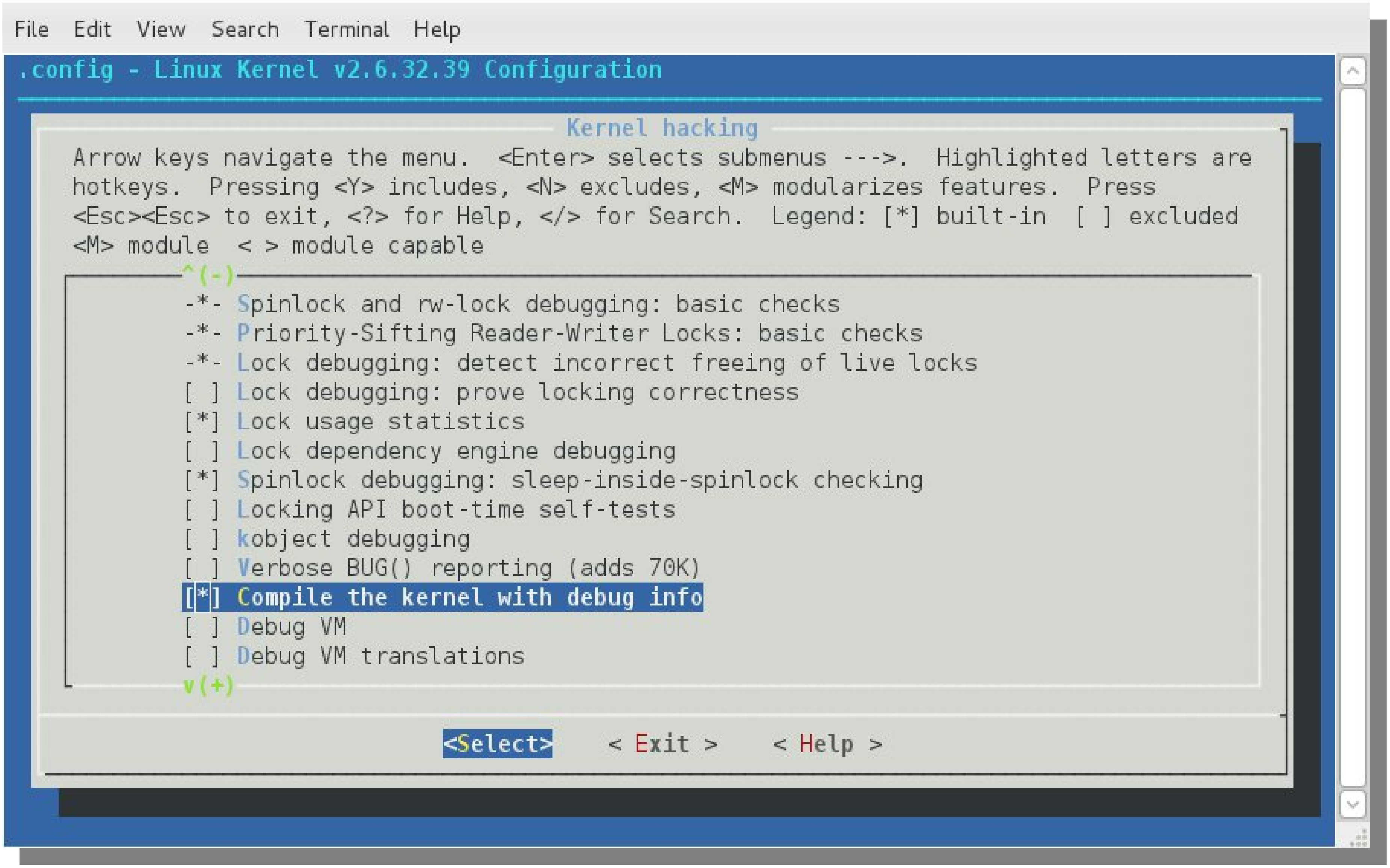
<0>	`KERN_EMERG`
<1>	`KERN_ALERT`
<2>	`KERN_CRIT`
<3>	`KERN_ERR`
<4>	`KERN_WARNING`
<5>	`KERN_NOTICE`
<6>	`KERN_INFO`
<7>	`KERN_DEBUG`

GDB And The Kernel

- Kernels usually have support for full debugging.
- Sometimes debugging via a JTAG device is preferred.
- When kernel is built “`-g`” you can “examine” a running kernel with:

```
gdb  vmlinux  /proc/kcore
```

Configure Kernel



```
.config - Linux Kernel v2.6.32.39 Configuration

                                Kernel hacking
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are
  hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press
  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable
          ^(-)
          -*- Spinlock and rw-lock debugging: basic checks
          -*- Priority-Sifting Reader-Writer Locks: basic checks
          -*- Lock debugging: detect incorrect freeing of live locks
          [ ] Lock debugging: prove locking correctness
          [*] Lock usage statistics
          [ ] Lock dependency engine debugging
          [*] Spinlock debugging: sleep-inside-spinlock checking
          [ ] Locking API boot-time self-tests
          [ ] kobject debugging
          [ ] Verbose BUG() reporting (adds 70K)
          [*] Compile the kernel with debug info
          [ ] Debug VM
          [ ] Debug VM translations
          v(+)

                  <Select>    < Exit >    < Help >
```

CONFIG Macros For Debugging

- *Get config file from /proc/config.gz if present*
- **KALLSYMS -> /sys addr info, better oops, kallsysms_lookup_name()**
- **KPROBES -> kprobes**
- **DEBUG_INFO -> debug syms in kernel (kdump)**
- **CRASH_DUMP -> for crash dump kernels (kexec)**
- **PROC_VMCORE -> exports dump image**
- **PROFILING, OPROFILE, APIC -> oprofile info**
- **PRINTK_TIME -> adds time stamp to printk output**
- **DEBUG_SLAB -> catch use of freed memory**
- **DEBUG_SPINLOCK_SLEEP -> sleeping while holding lock is reported**
- **DEBUG_SPINLOCK -> uninitialized and other errors**
- **MAGIC_SYSRQ -> magic sys request key**
- **DEBUG_KERNEL, DEBUG_INFO-> general kernel debugging**

KALLSYMS

- All statically and dynamically linked, non-stack, kernel symbols – more than system.map
- Sorted by address
- Shows up as /proc/kallsyms
- Addresses show up as zero for non-privileged users – security feature.
- Eventually it may be made unreadable to non-privileged users. One can change the permissions – say in a init script to make it unreadable if desired.

```
[root@centos6 ~]# head /proc/kallsyms
0000000000000000 D per_cpu__irq_stack_union
0000000000000000 D __per_cpu_start
0000000000004000 D per_cpu__gdt_page
0000000000005000 d per_cpu__exception_stacks
000000000000b000 d per_cpu__idt_desc
```

PROC_VMCORE

- In filesystems → Pseudo filesystems

```
config PROC_VMCORE
        bool "/proc/vmcore support "
        depends on PROC_FS && CRASH_DUMP
        default y
        help
        Exports the dump image of crashed kernel in ELF format.
```

KPROBES

- **KPROBES is in arch/Kconfig**
- **Provides the fundamental capability of trapping at arbitrary kernel addresses, in function entry and exit.**
- **arch/Kconfig also contains config variables for OPROFILE**
- **Kprobes can be used in loadable kernel modules to register code to be executed at various places in the kernel. Without rebuilding or rebooting.**

```
int my_init(void)
{
        kp.pre_handler=handler_pre;
        kp.post_handler=handler_post;
        kp.fault_handler=handler_fault;
        kp.addr = (void *)func_addr;
        return register_kprobe(&kp);
}
```

```
insmod kprobes_example.ko func_addr=$((0x$(grep do_fork /proc/kallsyms |grep T |cut -f1 -d" ") ))
```

http://www.ibm.com/developerworks/library/l-kprobes/index.html

DEBUG_INFO

- Configured under Kernel hacking
- [*] Compile the kernel with debug info

If you say Y here the resulting kernel image will include debugging info resulting in a larger kernel image. This adds debug symbols to the kernel and modules (gcc -g), and is needed if you intend to use kernel crashdump or binary object tools like crash, kgdb, LKCD, gdb, etc on the kernel

Say Y here only if you plan to debug the kernel.

If unsure, say N.

CRASH_DUMP

- **Found in Processor type and features**
- **[*] kernel crash dumps**

Generate crash dump after being started by kexec. This should be normally only set in special crash dump kernels which are loaded in the main kernel with kexec-tools into a specially reserved region and then later executed after a crash by kdump/kexec. The crash dump kernel must be compiled to a memory address not used by the main kernel or BIOS using PHYSICAL_START, or it must be built as a relocatable image (CONFIG_RELOCATABLE=y).

For more details see Documentation/kdump/kdump.txt

PROFILING

- **General setup**
- **[*] Profiling support (EXPERIMENTAL)**

```
config PROFILING
        bool "Profiling support (EXPERIMENTAL)"
        help
          Say Y here to enable the extended profiling support mechanisms used
by profilers such as OProfile.
```

OPROFILE

- **General setup**
- **< > OProfile system profiling (EXPERIMENTAL)**

```
config OPROFILE
        tristate "OProfile system profiling (EXPERIMENTAL)"
        depends on PROFILING
        depends on HAVE_OPROFILE
        select RING_BUFFER
        select RING_BUFFER_ALLOW_SWAP
        help
          OProfile is a profiling system capable of profiling the
          whole system, include the kernel, kernel modules, libraries,
          and applications.

          If unsure, say N.
```

PRINTK_TIME

- **Kernel hacking**
- **[ ] Show timing information on printks**

```
config PRINTK_TIME
        bool "Show timing information on printks"
        depends on PRINTK
        help
          Selecting this option causes timing information to be
          included in printk output.  This allows you to measure
          the interval between kernel operations, including bootup
          operations.  This is useful for identifying long delays
          in kernel startup.
```

- **Defined in lib/Kconfig.debug**
  - **check that config file for other useful CONFIG macros**

DEBUG_SLAB

- **Kernel hacking**
- **[ ] Debug slab memory allocations**

```
config DEBUG_SLAB
        bool "Debug slab memory allocations"
        depends on DEBUG_KERNEL && SLAB && !KMEMCHECK
        help
          Say Y here to have the kernel do limited verification on memory allocation as well as poisoning memory on free to catch use of freed memory. This can make kmalloc/kfree-intensive workloads much slower.
```

DEBUG_SPINLOCK_SLEEP

- **Kernel hacking**
- **[ ] Spinlock debugging: sleep-inside-spinlock checking**

(using "/" in make menuconfig to search for the symbol)

```
CONFIG_DEBUG_SPINLOCK_SLEEP:

If you say Y here, various routines which may sleep will become very noisy if they are
called with a spinlock held.

Symbol: DEBUG_SPINLOCK_SLEEP [=n]
Prompt: Spinlock debugging: sleep-inside-spinlock checking
    Defined at lib/Kconfig.debug:616
    Depends on: DEBUG_KERNEL [=y]
    Location:
      -> Kernel hacking
```

DEBUG_SPINLOCK

- **Kernel hacking**
- **[ ] Spinlock and rw-lock debugging: basic checks**

```
Symbol: DEBUG_SPINLOCK [=n]
Prompt: Spinlock and rw-lock debugging: basic checks
  Defined at lib/Kconfig.debug:502
  Depends on: DEBUG_KERNEL [=y]
  Location:
    -> Kernel hacking
  Selected by: DEBUG_LOCK_ALLOC [=n] && DEBUG_KERNEL [=y] && TRACE_IRQFLAGS_SUPPORT [=y] && STACKTRACE_SUPPORT [=y]
```

MAGIC_SYSRQ

- **Kernel hacking**

```
Symbol: MAGIC_SYSRQ [=y]
Prompt: Magic SysRq key
Defined at lib/Kconfig.debug:39
    Depends on: !UML [=UML]
    Location:
    -> Kernel hacking
  Selected by: KGDB_SERIAL_CONSOLE [=y] && KGDB [=y]
```

- **Details in Documentation/sysrq.txt**

DEBUG_KERNEL

- **About 40 kernel config macros begin CONFIG_DEBUG***
- **Kernel hacking**

```
config DEBUG_KERNEL
        bool "Kernel debugging"
        help
          Say Y here if you are developing drivers or trying to debug and
          identify kernel problems.
```

- **DEBUG_KERNEL is hardly used in kernel source, but lots of other CONFIGs depend on it.**
- **The kernel Makefile checks “CONFIG_DEBUG_INFO” to decide whether to use “-g” on the compile.**

gdb : Examine Data Structure

```
(gdb) print *(struct file_system_type *)file_systems
$1 = {name = 0xc01f3c20 "ext2", fs_flags = 1,
  read_super = 0xc0147d54 <ext2_read_super>, next = 0xc0221eac}

(gdb) print *((struct file_system_type *)file_systems)->next
$2 = {name = 0xc01f2269 "minix", fs_flags = 1,
  read_super = 0xc013ec74 <minix_read_super>, next = 0xc022740c}

(gdb) print (struct file_operations)(*0xc01f3c20)
$3 = {llseek = 0x32747865, read = 0, write = 0, readdir = 0, poll =
 0,
  ioctl = 0, mmap = 0, open = 0, flush = 0x64616552,
  release = 0x69616620, fsync = 0x6572756c, fasync = 0x6e69202c,
  check_media_change = 0x3d65646f, revalidate = 0x2c646c25,
  lock = 0x6f6c6220}

(gdb)
```

gdb : Disassemble

```
(gdb) disassemble init_module
Dump of assembler code for function init_module:
0xc023dd58 <init_module>:    fstpt (%esp,1)
0xc023dd5b <init_module+3>:  mov   0x8(%ebp),%ebx
0xc023dd5e <init_module+6>:  push  %ebx
0xc023dd5f <init_module+7>:  call  0xc0239a10
 <init_task_union+6672>
0xc023dd64 <init_module+12>: jmp   0xc023e18d
 <free_area_init+157>
0xc023dd69 <init_module+17>: lea   0x0(%esi,1),%esi
End of assembler dump.
(gdb)
```

Remote debugging with `kgdb`

- `kgdb` is/was a kernel patch
- Allows you to use one Linux machine (the host system) to perform source-level kernel debugging on another (the target system) using `gdb` over a serial line.
- Only the target needs to run the `kgdb` Linux kernel

View Source Code

- **As a result of** `add-symbol-file`**, Machine A will display the symbols of the module.**
- **You must pass** `add-symbol-file` **the load address of the module from Machine B:**

```
# insmod simple.ko
# cat /sys/module/simple/sections/.text
```

- **On Machine A:**

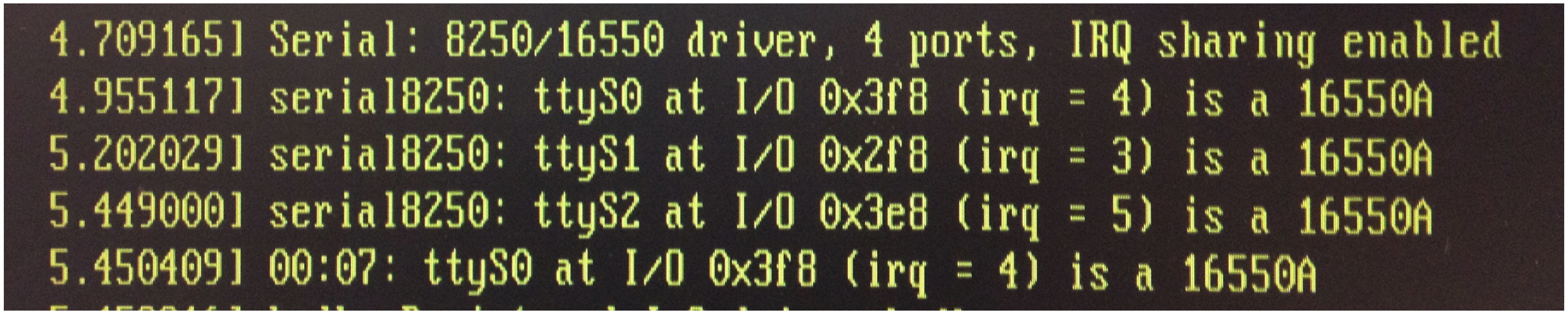
```
(gdb) add-symbol-file simplemod.ko 0xc404304c # address of text
(gdb) list
8
9       int init_module(void) {
10
11              printk (KERN_DEBUG "hello from your module\n");
…
```

Using GDB

- gdb program <core file>
- gdb –pid <pid>
- run <args>
- break <where>
  - function name
  - line number
  - file:line number
- backtrace/where/bt
- step or next
- finish – current function
- continue
- list
- target remote <remote>
  - /dev/ttyS0 – if using serial port
  - <addr>:<port> -- terminal server
- set remotebaud <speed>

Kernel Remote kgdbwait

- kernel ... kgdbwait – in GRUB
- May need serial port information



```
4.709165] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
4.955117] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
5.202029] serial8250: ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
5.449000] serial8250: ttyS2 at I/O 0x3e8 (irq = 5) is a 16550A
5.450409] 00:07: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
```

KGDB After Booting

On Host

- gdb vmlinux
- target remote <serial port or terminalserver:port>
- *do the stuff on the target below and wait for a while ...*
- break do_execve
- c   # for “c”ontinue-

On Target

- stty -F /dev/ttyS1   115200
- echo ttyS1 >/sys/module/kgdboc/parameters/kgdboc
- echo g >/proc/sysrq-trigger

Deadlocks

- configure kernel for deadlock detection
- deadlocks can hang kernel
- mutexes have checking possible, semaphores have little or no checking
- Documentation/mutex-design.txt among other things

Memory leaks

- most kernel memory allocation routines do not provide garbage collection
- thus, allocator must be sure to free
- look at /proc/meminfo
- kernel hacking → Kernel memory leak detector
  - The memory allocation/freeing is traced in a way similar to the Boehm's conservative garbage collector, the difference being that the orphan objects are not freed but only shown in /sys/kernel/debug/kmemleak. Enabling this feature will introduce an overhead to memory allocations. See Documentation/kmemleak.txt for more details.

Magic SYSRQ

- kernel responds to special key sequence unless it is really hung (which unfortunately does happen...) ALT-SysRq or ALT-Print Screen
- CONFIG_MAGIC_SYSRQ
- write integer value into /proc/sys/kernel/sysrq

```
 0 - disable sysrq completely
   1 - enable all functions of sysrq
  >1 - bitmask of allowed sysrq functions (see below for detailed function
       description):
          2 - enable control of console logging level
          4 - enable control of keyboard (SAK, unraw)
           8 - enable debugging dumps of processes etc.
         16 - enable sync command
         32 - enable remount read-only
         64 - enable signalling of processes (term, kill, oom-kill)
        128 - allow reboot/poweroff
        256 - allow nicing of all RT tasks
```

Magic SysRq commands

- 'b' - Will immediately reboot the system without syncing or unmounting your disks.
- 'c' - Will perform a kexec reboot in order to take a crashdump.
- 'd' - Shows all locks that are held.
- 'e' - Send a SIGTERM to all processes, except for init.
- 'f' - Will call oom_kill to kill a memory hog process.
- 'g' - Used by kgdb on ppc and sh platforms.
- 'h' - Will display help (actually any other key than those listed here will display help. but 'h' is easy to remember :-)
- 'i' - Send a SIGKILL to all processes, except for init.
- 'k' - Secure Access Key (SAK) Kills all programs on the current virtual console. NOTE: See important comments below in SAK section.
- 'l' - Shows a stack backtrace for all active CPUs.
- 'm' - Will dump current memory info to your console.
- 'n' - Used to make RT tasks nice-able
- 'o' - Will shut your system off (if configured and supported).
- 'p' - Will dump the current registers and flags to your console.

`strace`

- The `strace` command prints the system calls made by a process/program to `stderr`
- A good tool for bug isolation
- Command Usage:

`strace` ***program <arguments>*** `...`

Example:

```
# strace ls /dev
```

Filesystems For Debugging

- The Linux kernel supports a variety of virtual filesystems
- Proc
  - main model is one page or smaller text file metaphor
  - discouraged from new use
  - 30,000+ entries
  - mount -t proc proc /proc
- Sys
  - representation of kernel object hierarchy
  - 10,000+ entries
  - mount -t sysfs sysfs /sys
- Debug
  - fast and flexible way of exporting debugging data
  - intended to be simpler than proc or sys
  - few entries, maybe 100+
  - mount -t debugfs debugfs /sys/kernel/debug

`/proc`

- Interface to kernel information
- Data appears as text files
- Use `cat` or `more` to display files
- Some users of Linux do not have `/proc` (it must be mounted)

/proc/interrupts

Lists all IRQs and which device is using them.

```
# cat /proc/interrupts
           CPU0
  0:      68751 XT-PIC timer
  1:        595 XT-PIC keyboard
  2:          0 XT-PIC cascade
  8:          1 XT-PIC rtc
 10:        127 XT-PIC eth0
 12:       9634 XT-PIC PS/2 Mouse
 13:          1 XT-PIC math error
 14:     194343 XT-PIC ide0
 15:          0 XT-PIC ide1
```

/proc/iomem

Shows memory mapped control registers and device RAM

```
# cat /proc/iomem

00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-05ff9fff : System RAM
  00100000-002549c6 : Kernel code
  002549c7-002ab89f : Kernel data
```

K Computing

/proc/devices

Lists all device drivers compiled into the kernel or currently loaded.

```
# cat /proc/devices
Character devices:
1 mem
2 pty
3 ttyp


Block devices:
1 ramdisk
2 fd
3 ide0
```

/proc/modules

Lists all dynamically loaded modules

```
# cat /proc/modules
simplemodule 1536 1 - Live 0x1281e000
runrpc 160421 1 - Live 0x12a7c000
```

Usage count

Size of module

Creating files in `/proc`

- Define function that fills a page of memory with the contents of the new proc file
- Register read only new nodes with `create_proc_read_entry(name, mode, directory_base, read_proc, data)`
- Unregister nodes with `remove_proc_entry()`
- `Documentation/DocBook/procfs_example.c`

Sysfs

- represent hierarchy of kernel objects
- directories, regular files and symbolic links
- try the tree command on /sys
- top level directories represent the major kernel subsystems
- originally designed to help debug device driver model
- http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

The Sys Filesystem

- tied to kobjects
- mount -t sysfs sysfs /sys
- a directory per object is created
- directory structure represents object hierarchy
- many parts seemingly undocumented -> what's new ...
- contains info about:
  - block drivers
  - character drivers
  - buses
  - and more

K Computing

Sysfs Information

- **driver information**
  - e.g., module/e1000e/parameters
  - allows easy userspace access to selected driver (module) parameters
  - parameters must have been registered with non-zero permissions
  - lots of module parameters don't show up as entries
- **device information**
  - e.g., devices/system/clocksource/clocksource0/available_clocksource
- **bus and lots of other info**

K Computing

Command line parameters

- **insmod module.ko *parm=value***
- **module_param(name, type, permissions)**
- **module_param_array(name, type, number, perm)**
- **MODULE_PARM_DESC()**
- **Module parameters are listed in /sys/module under the module name.**

Use of debugfs

- place for kernel code to make info they consider debugging info, available
- uses the relay interface for efficient means of transferring large amounts of data.
- filesystem to mount on /sys/kernel/debug
- mount -t debugfs debugfs /sys/kernel/debug
- examples from /sys/kernel/debug
  - ieee80211
  - kprobes
  - usbmon
- see Documentation/relay.txt and fs/debugfs for more info.

Creating files in debugfs

- various debug_* functions
- debugfs_create_dir()
- debugfs_create_file()
- debugfs_remove()
- see for example: drivers/net/wireless/ath5k/debug.c

SystemTap

- A scripting language for running code inside the kernel.
- Not a filesystem, but, may be used for debugging.
- Must install several systemtap packages and debuginfo for kernel.
- Lots and lots of existing scripts.
- May be much simpler than writing straight C code.
- There is essentially a library of functionality.
- Can compile on one machine and run on another, but the other must have the same architecture and distribution and have systemtap-runtime installed.

https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/SystemTap_Beginners_Guide/index.html