Debugging Tools and Methods for Kernel Developers

Johannes Leopold

January 5, 2016

Overview

printk

```
printk(KERN_ALERT "You should already be familiar with this!\n");
```

- A way to print messages from Kernel code
- Grouped by importance
- slow!
- Rate limiting is a good idea (i.e. only printing the same messages again after a fixed time)

How to view the messages

- **dmesg:** Command to print current kernel message buffer
  - shows newest kernel messages (things printed with printk)
  - dmesg –help for more options, though usually not necessary
- **/var/log/kern.log:** File containing kernel messages
- **/var/log/messages:** Also contains kernel messages, amongst other things.
  - only logs kernel messages when syslogd daemon is running!

Alternatives to printk

- **dev_dbg, dev_info, etc.:** Substitute for printk when writing device drivers

```
int dev_dbg(struct device *dev, char *format, ...);
```

  - Message importance is part of the function name
  - Prints where the message is coming from (*dev)
  - does so in a consistent format ($\rightarrow$ machine readable)
- **pr_dbg, pr_emerg, etc.:** Similar to dev_*, except not specific to device drivers

```
int pr_info(const char *format, ...);
```

- Somewhat controversial

Dynamic debugging

- debug-level messages can be toggled at runtime
- virtually no additional cost when off
- write to **/sys/kernel/debug/dynamic_debug/control** to toggle on/off
  - (or wherever your debugfs is mounted instead of /sys/kernel/debug)

Syslogd

- Service that collects messages from other services and daemons
- also logs kernel messages
- writes to **/var/log/messages**
- Config file: **/etc/syslogd.conf**
- logfiles prefixed with - will not be flushed to disk immediately
  - this is useful if you are sending a lot of messages, which can cause slowdown

/proc filesystem

- /proc is a virtual file system, used by the kernel to export information
- every file is tied to a kernel function, contents are created on writing to/reading files
- e.g. **/proc/modules** is a list of all currently loaded modules
- your modules can generate their own /proc files
  - for new code, it is recommended to use sysfs instead
  - just temporarily for debugging, /proc is easier to use however

Working with /proc

- We will use a **seq_file** implementation, because it is safer.
- Use **proc_create** to create proc files
  - old documentation might say create_proc_entry, this is deprecated
- If you do not need to write much data, you can use the simplified seq_file methods

- More info: https://www.linux.com/learn/linux-training/37985-the-kernel-newbie-corner-kernel-debugging-using-proc-qsequenceq-files-part-1

Code Demo

debugfs

- RAM-based file system specifically created for debugging
- Allows you to make kernel layer information available in userspace
- Unlike proc fs and sysfs, there are no rules.
- **CONFIG_DEBUG_FS** needs to be enabled
- Many distributions come with a debugfs mounted already,
  - use "`mount | grep debugfs`" to check

- Good tutorial: http://opensourceforu.efytimes.com/2010/10/debugging-linux-kernel-with-debugfs/
- A word of warning: https://lwn.net/Articles/429321/

ioctl

- An alternative to /proc
- retrieves data in binary form, rather than text documents
  - this eliminates overhead, and makes it faster than reading from /proc
- does not require splitting data in fragments smaller than a page

Watching

- track problems in kernel modules down by watching behaviour of userspace programs
- test kernel code, make sure it does the right thing
- when it does not, find out in which cases
- then, look at the code
- Use a debugger on the program, or strace

strace

- a tool for tracing interactions with the kernel in programs
- shows you your programs system calls, state changes, signal deliveries
- can also show arguments to calls
  - e.g. useful for seeing what files a program is accessing
- can be used on any program, regardless of debugging support.

Demonstration

Static code analysis

- A way of finding common bugs without even compiling the code
- finds things like e.g. null pointer dereferences (but not always!)
- Tools commonly used for the kernel are **Sparse** and **Coccinelle**
  - Sparse is built into the kernel make system! use **make C=1** or **make C=2**
- others such as **Coverity** can be used too
- gcc warnings
- **checkpatch.pl** - tool provided by the kernel to find common mistakes and style errors

Kernel Debuggers

- very time consuming, a "last resort"
- require recompiling kernel with special options
  - **CONFIG_GDB_SCRIPTS** on (for gdb)
  - **CONFIG_FRAME_POINTER**, if supported, on
  - **CONFIG_DEBUG_INFO_REDUCED** off
- anger the linux gods

- Linus' rant about kernel debuggers: http://lwn.net/2000/0914/a/lt-debugger.php3

Kernel Debuggers

- gdb
  - You will need QEMU for this (or JTAG-based hardware)
  - Documentation/gdb-kernel-debugging.txt
- DTrace
  - also specifically designed for kernel debugging (originally for Solaris)
- kgdb
  - debugger specifically for linux kernel (as well as a few BSDs)
  - debug a machine from a second machine using serial or network connection

Kernel Dynamic Probes

- a probe is an automated breakpoint
- implanted dynamically in running modules
- no need to modify module source code
- ability to inject/simulate faults
- insert code (e.g. printks) without recompiling
- kprobes and jprobes

kprobes

- kprobes are written as a module
  - can be loaded / unloaded using insmod / rmmod
  - thus can be written for and used on an already running system
- kprobes can be inserted anywhere at an address or symbol
- consist of pre-handler, post-handler and fault-handler
- called before the probe point, after the probe point, and on fault within kprobe

jprobes

- jprobes are an "extension" of kprobes
- always inserted at start of kernel function
  - jprobes can therefore access function parameters
- No pre- and post-handler, only one.
- struct jprobe contains struct kprobe

Oops

- A deviation from correct behaviour in the kernel
- not necessarily unrecoverable
- when unrecoverable, causes a kernel panic
- produces an error log
- process causing the deviation is then killed
- A common cause is e.g. a null pointer dereference

- See also: Documentation/oops-tracing.txt

Kernel Options

- find current settings in **/boot/config-$(uname -r)** (on CentOS)
- Many debugging features require you to enable Kernel options
- There are a lot of options!
- Find a (hopefully) complete list of relevant ones in the presentation files

- How to build a Custom Kernel on CentOS: https://wiki.centos.org/HowTos/Custom_Kernel

Sources

- https://lwn.net/Articles/434833/
- https://lwn.net/Articles/487437/
- http://www.linfo.org/dmesg.html
- http://kernelnewbies.org/KernelDebug
- http://www.makelinux.net/ldd3/
- https://sourceware.org/systemtap/kprobes/
- http://lwn.net/Articles/115405/
- also, the pages linked on previous slides