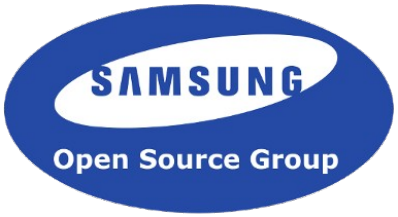


Linux Kernel Debugging Tools Overview

Linux Plumbers Conference, August 2015

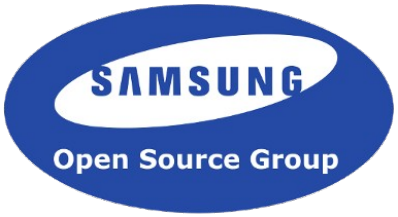
Shuah Khan
Samsung Open Source Group
shuahkh@osg.samsung.com

Abstract

Software and bugs go together, and the Linux Kernel is no exception. Some bugs are easy to find, but some are harder to reproduce and could require several attempts to piece together the right set of conditions to trigger a specific bug. While bugs that result in a system crash or hang are easier spot it is often more challenging to gather the information necessary to debug and fix. In many cases, Kernel logs can provide insight into these bugs, but when a system crashes or freezes Kernel logs might not get the chance to be written out to disk.

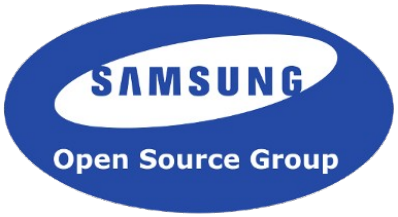
In this talk, Shuah will give an overview of Kernel debug interfaces and tools.

Agenda



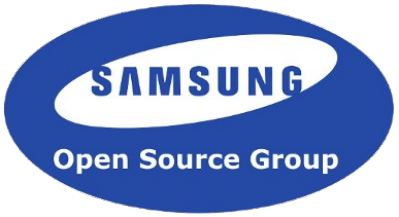
- Linux Kernel has Bugs! Really!!
- Debugging Methods
- Debugging Tools
- Kernel Debug Interfaces
- Tracepoints
- git bisect
- Kernel Debug Support
- Kernel Debug Tools
- Watch out for
- Parting thoughts

Linux Kernel has Bugs! Really!!



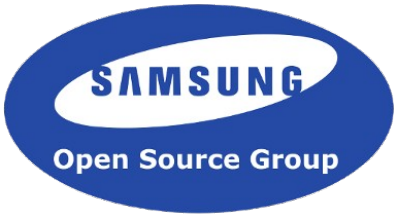
- System Hangs
- System Faults and Panics
- Races and Timing Bugs
- Incorrect or unexpected feature behavior

Debugging Methods ...



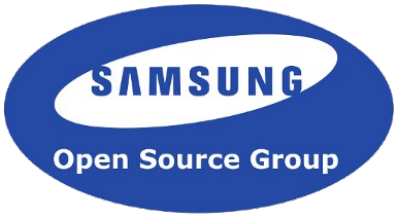
- Kernel Logging
- Redirecting Console Messages
- Check System State (lspci, lsusb etc.)
- Oops messages
- System Hangs - SysRq key
- Races and Timing Bugs
- Incorrect or unexpected feature behavior

Debugging Tools ...



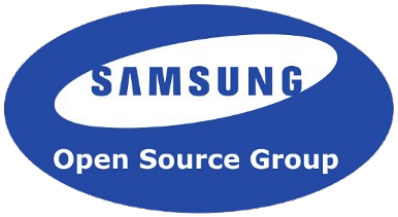
- gdb
- kdb
- Dynamic Probes
- Kmemcheck
- Kernel Address Sanitizer

Kernel Debug Interfaces ...



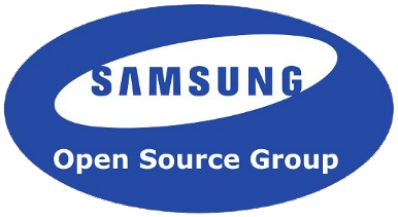
- Debug configuration options
  - Several kernel modules have debug configuration options that can be enabled at compile time. No dynamic control, either on or off. Debug messages go to dmesg.
- Debug APIs – DMA-debug (lib/dma-debug.c)
  - Designed for debugging driver dma api usage errors
  - Keeps track of dma mappings per device
  - Detects unmap attempts on addresses that aren't mapped
  - Detects missing mapping error checks in driver code after dma map attempt.
  - CONFIG_HAVE_DMA_API_DEBUG and CONFIG_DMA_API_DEBUG

Kernel Debug Interfaces ...



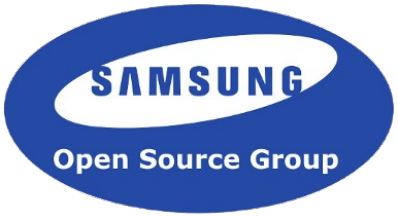
- Dynamic debug
  - Feature allows dynamically enabling/disabling pr_debug(), dev_dbg(), print_hex_dump_debug(), print_hex_dump_bytes() per-callsite.
  - CONFIG_DYNAMIC_DEBUG controls feature enable/disable
  - /sys/kernel/debug/dynamic_debug/control (virtual file)
  - Specify dynamic_debug.verbose=1 kernel boot option to increase the verbosity
  - Documentation/dynamic-debug-howto.txt

Kernel Debug Interfaces ...



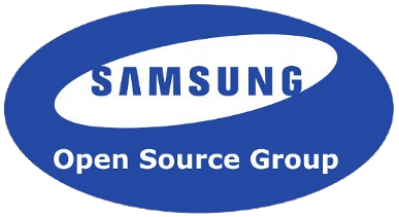
- Dynamic debug
  - Per-callsite: a specific pr_debug() can be enabled using the line number in the source file. e.g: enabling pr_debug() in kernel/power/suspend.c at line 340:
    - $ echo 'file suspend.c line 340 +p' > /sys/kernel/debug/dynamic_debug/control
  - Per-module: passing in dyndbg=”plmft” modprobe or changing or creating modname.conf file in /etc/modprobe.d/ - The later persists across reboots. However for drivers that get loaded from initramfs, change grub to pass in module.dyndbg="+plmft"

Kernel Debug Interfaces ...



- Tracepoints
  - Can be used for tracing for debug, event reporting, and performance accounting.
  - Can be enabled to trigger at run-time
  - Debug hooks such as DMA API debug are either enabled or disabled at compile time.
  - In the case of dynamic debug, debug levels can be changed at run-time, but the code is compiled in.
  - Tracepoint code is different in that it is inactive unless it is enabled. When it is enabled, code is modified to include the tracepoint code. It doesn't add any overhead.

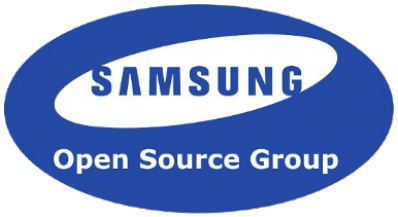
Tracepoint Mechanism ...



The tracepoints use jump labels which are a code modification of a branch. When it is disabled, the code looks like:

```
   [ code ]
    nop
back:
    [ code ]
    return;
tracepoint:
   [ tracepoint code ]
   jmp back;
```

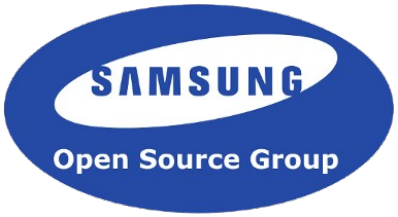
Tracepoint Mechanism ...



The tracepoints use jump labels which are a code modification of a branch. When it is enabled, the code looks like:

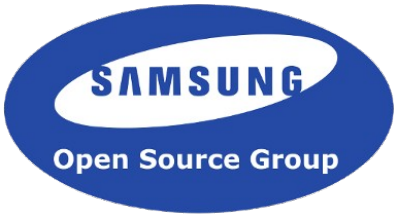
```
   [ code ]
   jmp tracepoint
back:
    [ code ]
    return;
tracepoint:
   [ tracepoint code ]
   jmp back;
```

git bisect - Isolate a Bad Commit ...



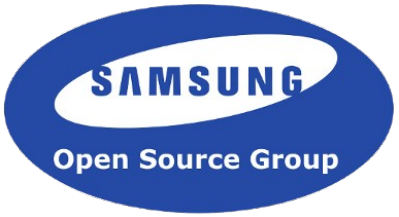
- This is how it works:
  - $ git bisect start
  - $ git bisect bad # Current version is bad
  - $ git bisect good v4.2-rc6 # last good version
- Once one bad and one good version are specified, git bisect will start bisecting pulling in commits done between the good version and the bad.
- At each step, compile, test, and tag the version good or bad. There could be several versions to test.

git bisect - Isolate a Bad Commit ...



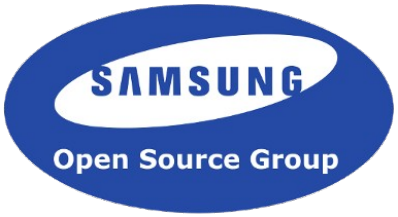
- $ git bisect log # shows the step by step bisect progress
- $ git bisect reset # resets, to be used in case of mistakes in tagging, save git log output and replay prior to reset
- $ git bisect replay git_log_output
- When the last version is tested, git bisect will flag a commit that is bad.
- git bisect can be run on a section of kernel source tree if the problem is clearly in that area. For example, debugging a problem in radeon driver, running git bisect on drivers/drm/radeon will limit the bisect scope the commits to drivers/drm/radeon
  - $ git bisect start -- drivers/drm/radeon

Kernel Debug Configuration Options ...



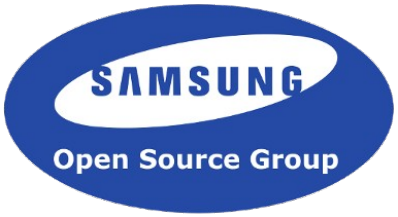
- Lock Debugging (spinlocks, mutexes, etc…)
- Debug Lockups and Hangs
- Read-copy update (RCU) Debugging
- Memory Debugging

Kernel Test and Debug Modules ...



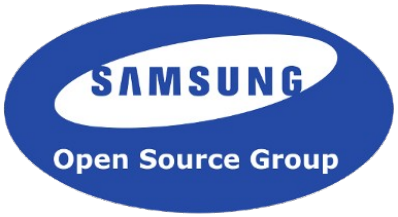
- test_firmware
- test_bpf
- Find more under kernel sources: lib directory

Kernel Debug Tools ...



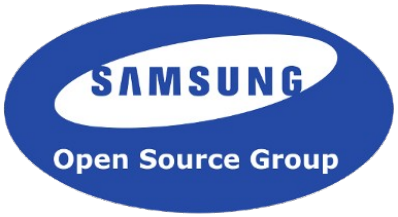
- kmemcheck
  - detects and warns about uninitialized memory
  - CONFIG_KMEMCHECK should be enabled
  - Documentation/kmemcheck.txt
- kmemleak
  - Debug feature to detect possible kernel memory leaks in a way similar to a tracing garbage collector
  - CONFIG_DEBUG_KMEMLEAK should be enabled
  - Documentation/kmemleak.txt

Kernel Debug Tools ...



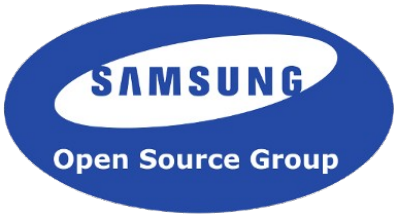
- Kernel Address Sanitizer
  - http://events.linuxfoundation.org/events/linuxcon-north-america/program/schedule
  - Andrey Konovalov

Watch Out for ...

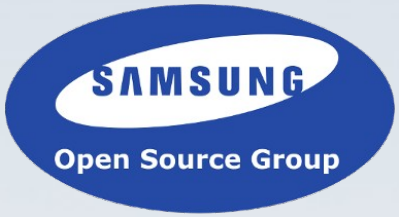


- Debug options:
  - Add performance cost
  - Use non-stack allocations
  - Execute non-optimal code paths
  - Are not great for debugging races/timing bugs
  - Generate too many debug messages

Parting thoughts ...



- Use the Source
- Watch the System
- Debugging Techniques
  - https://lwn.net/images/pdf/LDD3/ch04.pdf
- Elinux Debugging Portal
  - http://elinux.org/Debugging_Portal



Thank You!