Kernel Software Variability

From a kernel developer's perspective

commonly known as #ifdef challenges

Jesper Dangaard Brouer
Principal Kernel Engineer
Red Hat inc.

Keynote FOSD, May 2016
Feature-Oriented Software Development

Intro

- Software Variability and “Software Product Lines”
  - For me, commonly know as ifdef challenges ;-)
- This is outside my area of expertise
  - I work with the Linux kernel
    - core network stack and memory subsystem
  - Cannot solve your research problems
    - I'll share my interactions with annoying ifdefs
    - In hope to give insight into more problems to solve ;-)
    - And current state of handling ifdef build issues
  - In the future I hope your research will help Linux

Kernel Config #ifdef challenges

- Kernel's config allows great deal of customization
  - Allow to run on big server and small embedded systems
  - Embedded often compile out large parts of kernel
  - Can be viewed as "Software Product Lines"
- Ifdef bugs can be hiding
  - e.g. only visible in certain combinations of kernel configs
- Very subtle bugs can occur due to config ifdef's

Kernel compile/build errors

- Most commonly and easy detectable
  - Config combo's that result in kernel compile errors
  - Some maintainers catch these themselves
    - Before they push their git tree publicly
  - Rest is caught by: kbuild robot
    - Fengguang Wu at Intel have automated system to detect these
    - Recently pickup email patches (before applied to any git tree)
    - (More on kbuild robot later)

Kernel make system

- Kernel make have a
  - `make randconfig`
    - For generating random config options
    - e.g. kbuild robot uses this
- There lots of default config per arch in
  - linux/arch/*/configs/*defconfig
    - Kbuild robot also uses these

Common network issue: CONFIG_IPv6

- IPv6 support can be compiled out
  - See CONFIG_IPV6
  - This is a common thing people get wrong
    - often only result in build bugs
- Git command line hint
  - Find all commits containing string CONFIG_IPV6

```
git log -S CONFIG_IPV6
```

Recent Micro benchmarking work

- micro benchmarking:exclusive access kernel primitives
- Performance differs with different settings of
  - CONFIG_PREEMPT
    - Obviously, slightly more overhead getting exclusive access
  - CONFIG_PREEMPT_COUNT
    - can be enabled even if CONFIG_PREEMPT is disabled
    - is almost as costly as CONFIG_PREEMPT
    - can be selected by DEBUG_ATOMIC_SLEEP and DEBUG_KERNEL
  - CONFIG_DEBUG_PREEMPT
    - also adds a small cost extra

CONFIG_PREEMPT_*

- Functions like: local_bh_{disable,enable} and spinlocks
  - Are affected by these preempt settings

- Performance and Algorithm correctness
  - is affected by these preempt settings
  - Developers need to test different combinations
    - This is time consuming

Core mem affected by PREEMPT

- An optimization of core MM alloc function
  - Commit 9aabf810a6 ("mm/slub: optimize alloc/free fastpath by removing preemption on/off")
  - Resulted in GCC generating bad code on ARM64
    - Caused occasional hangs
    - On CONFIG_PREEMPT + !CONFIG_SMP kernels
    - Specifically GCC 4.9
    - Needed a READ_ONCE() (same as volatile)
    - Fixed in commit:
      - 859b7a0e89 ("mm/slub: fix lockups on PREEMPT && !SMP kernels")
  - Watch out for GCC generated code
    - on ALL architectures

Recent Memory Management development

- In my recent work within
  - performance of Memory Management subsystem
- I need to juggle:
  - CONFIG_SLUB_CPU_PARTIAL, SLUB_STATS, SLUB_DEBUG
  - and the mentioned PREEMPT combinations
- While developing, need enabling
  - debugging options that catch errors and give stats
- When performance measuring
  - need to disable all debug features

Performance: Ifdef in C-struct

- Ifdef's in C-struct is a pain
  - When optimizing for cacheline performance
  - Element alignment depend ifdefs
    - Can changes the cacheline boundaries
    - Can result in false-sharing cacheline bouncing
      - in other-wise performance optimized code
- Tedious process, optimize code for cacheline access
  - I use tool "pahole" to inspect struct layout
  - Adding ifdef, very annoying, requires recompiling
    - nice-to-have: if pahole could account for these ifdefs

Examples of structs with ifdefs

- `struct sk_buff` (include/linux/skbuff.h)
        - CONFIG_XFRM, CONFIG_NF_CONNTRACK, CONFIG_BRIDGE_NETFILTER, CONFIG_NET_SCHED, CONFIG_NET_CLS_ACT, CONFIG_NET_RX_BUSY_POLL, CONFIG_XPS, CONFIG_NETWORK_SECMARK
    - Can result in memset touching 3 vs. 4 cachelines
- `struct net` (include/net/net_namespace.h)
    - huge struct, due to many other structs as members
        - cacheline alignment is a nightmare
        - e.g. CONFIG_IPV6, CONFIG_IEEE802154_6LOWPAN, CONFIG_IP_SCTP, CONFIG_IP_DCCP, CONFIG_NETFILTER, CONFIG_NF_CONNTRACK, CONFIG_NF_TABLES, CONFIG_NF_DEFRAG_IPV6, CONFIG_WEXT_CORE, CONFIG_XFRM, CONFIG_IP_VS, CONFIG_MPLS

Performance: removing code

- Ifdef's removing code sections
  - Can (obviously) also improve performance
    - two reasons:
    - (1) Less instruction to be executed
    - (2) Less use of instruction-cache
- Example: CONFIG_NET_CLS_ACT
  - avoids calling "handle_ing()" in __netif_receive_skb_core()
    - (which gets inlined, thus also reducing i-cache)
      - Recent kernels "handle_ing()" renamed to “sch_handle_ingress”

Your research: good step forward

- In article: “42 Variability Bugs in the Linux Kernel”
  - http://www.itu.dk/people/brabrand/42-bugs.pdf
  - Found and analyzed 42 ifdef kernel bugs
  - Categorized them
  - Provided a online database at http://vbdb.itu.dk/
- No need for me to dig into the details
- Let's look at
  - How do we catch some of these today?

The kbuild robot "0-DAY kernel build"

- The kbuild robot
  - Currently best approach for catching ifdef build bugs
  - Run by Fengguang Wu <fengguang.wu@intel.com>
    - at Intel's Open Source Technology Center
  - Comprehensive, but brute-force approach
  - Sends email directly to developers based on git email
- Mailing lists:
  - https://lists.01.org/mailman/listinfo/kbuild-all
  - https://lists.01.org/mailman/listinfo/kbuild

Kbuild-robot: Catch build bugs

- Brute-force approach of
  - Finding build bugs and compiler warnings
    - test all 489 defconfigs defined in linux/arch/*/configs/
    - generate 900+ randconfigs each day
      - test kernel build + boot
- I their experience
  - randconfigs is quite effective in catching build bugs
  - They find static checks useful and efficient
    - Out-number the number of runtime regressions they caught

Kbuild-robot: More than build bugs

- Performance+power regression testing since 2013
- Functional tests are also supported
- Regressions are tracked for every test run
  - perf/power/boot/functional/latency/memory

- Git repo for reproducing test results
  - https://git.kernel.org/cgit/linux/kernel/git/wfg/lkp-tests.git/
  - For developers to reproduce and fix

Stats(1) about kbuild robot 0-day tests

- Stats directly from Fengguang Wu
- Monitoring 600+ kernel git trees around the world
  - can handle much more
    - so welcome to send the git URL to test
- In a typical day, performs
  - 2016 numbers:
    - 36000+ kernel builds
  - 2015 numbers:
    - 20000+ kernel boots (mostly in QEMU)
    - 12000+ runtime test jobs (mostly in physical machines)

Stats(2) about kbuild robot 0-day tests

- In a typical month, reports (no duplicates and low confident ones)
  - Stats 2015 → 2016 (first 4 month of 2016)
  - 250 → 477 build errors
  - 110 → 228 build warnings
    - Likely attributed to picking up email patches
  - 60 → 16 sparse warnings
  - 20 → 22 coccinelle warnings
  - (2016) Impressive total 743 reports average per month
  - (2015) 20 boot error/warnings
  - (2015) 10 perf/power/functional changes

Kbuild robot: “interface”

- High confident bugs/warnings
  - Send directly to devel-emails based on git info
  - And to mailing list (kbuild-all@01.org)
    - https://lists.01.org/pipermail/kbuild-all/
- Low confident (may be false positives)
  - Send to list (kbuild@01.org) for manual inspection
    - https://lists.01.org/pipermail/kbuild/
    - Manual forward email, if err/warn seems valid
- **Needed: Tool** for analyzing low confident ones
  - Research area?

Tool idea

- For a given patch: What config/ifdef is it affected by?
  - Tool output:
    - Either, List of CONFIG_xxx that influence code
    - Or, generate N .config files to cover combinations
  - Use it yourself, detect interesting variability commits
  - Tedious manual validation of this ifdef construct:
      - 81084651d7 ("slub: support for bulk free with SLUB freelists")
  - Kbuild-robot could also it use
    - but currently solves this brute-force, single devel cannot
  - Especially useful for maintainers
    - Before accepting patches

Example: ARRAY_SIZE() of spinlock array

- Subtle ifdef bug I introduced
  - Didn't realize code was affected by this config
- Array of spinlocks:

```
spinlock_t nf_conntrack_locks[CONNTRACK_LOCKS]
```

- Use ARRAY_SIZE(nf_conntrack_locks) in init-for-loop

```
#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))
```

- How can this result is a div by zero warning?
  - Because on uniprocessor (!CONFIG_SMP)
    - spinlock_t ended-up being an empty definition
- (Note: This was caught by kbuild-robot)

Kbuild robot lessons

- Experience from kbuild-robot also shows
  - You don't need to fix the bugs yourself
  - Detecting and delegating to original devel works well
  - Important to separate low vs. high confidence ones
    - to keep false positives low, to keep devel confidence high ;-)
  - Also learn from: do good report format
    - with git commit and reproducer notes
- Want high impact on the kernel
  - Write a small tool for Fengguang Wu ;-)

Efforts and assumptions

- Ongoing coding style effort to
  - Put #ifdefs into header files by defining stub functions
    - function available independently of config options
    - no #ifdefs in the .c files.
    - Not sure, if this is better or worse
- Upstream maintainers often do “make allyesconfig”
  - Assumes provides the best coverage
    - But likely not for feature-interaction bugs

The End

- Thanks to
  - Associate Professor, Claus Brabrand for inviting me
  - Fengguang Wu, for feedback and stats
    - And for building the kbuild-robot!

Extra

- Extra slides

Other tools

- Travis CI (Continous Integration): https://travis-ci.org/
  - free for Open Source projects (on github)
- Coverity Scan static analysis https://scan.coverity.com/
  - Avail for open source projects for free
- TypeChef
  - https://github.com/ckaestne/TypeChef