Linux Kernel Testing: Where Are We?

Guenter Roeck, Google
linux@roeck-us.net
Agenda

- Test Suites
- Testbeds
- Summary
- Next steps
Test Suites
Test Suites

- Linux Test Project (LTP)
- Module tests in tools/testing
  - kselftest
  - nvdimm
  - ...
- Static code analyzers
- Fuzzing tools
- Subsystem tests
  - e.g. xfstests
Linux Test Project (LTP)

- Collection of tools for testing the Linux kernel and related features
- Started by SGI
- Maintained by IBM, Cisco, Fujitsu, SUSE, Red Hat and others
LTP - Continued

● Coverage
  ○ 1000+ system calls
  ○ 1000+ POSIX conformance tests
  ○ 400+ IO stress tests
  ○ Realtime, networking, cgroups, namespace tests

● Links
  ○ https://linux-test-project.github.io/
  ○ https://github.com/linux-test-project/ltp/wiki
Kernel self-test

- Unit test framework in Linux kernel
- Driven by Shuah Khan
- Part of Linux kernel source
- Links
  - tools/testing/selftests/
  - https://kselftest.wiki.kernel.org/
  - https://lwn.net/Articles/608959/
Fuzzing Tools

- **Trinity**
  - Maintained by Dave Jones
  - A Linux System call fuzz tester
  - [https://github.com/kernelslacker/trinity](https://github.com/kernelslacker/trinity)

- **Syzcaller**
  - Developed and maintained by the Google syzcaller team
  - Unsupervised, coverage-guided Linux syscall fuzzer
  - Meant to be used with KASAN
  - [https://github.com/google/syzkaller](https://github.com/google/syzkaller)
Static Code Analyzers

- Coccinelle
  - Developed and maintained by Julia Lawall
  - A program matching and transformation engine
  - http://coccinelle.lip6.fr/

- Coverity
  - Commercial Static Analyzer
  - Linux kernel tested for free
  - Detailed test results and statistics available
● gcc warnings
● smatch
  ○ “The Source Matcher”
  ○ https://blogs.oracle.com/linuxkernel/entry/smatch_static_analysis_tool_overview
● sparse
  ○ A Semantic Parser for C
  ○ https://sparse.wiki.kernel.org/index.php/Main_Page
Automated Testing
Automated Testing

- **Autobuilders**
  - kisskb
  - 0Day
  - kernelci.org
  - Kerneltests.org
  - Other
    - Olof’s autobuilder, autobooter
    - Tegra builds
    - Buildbot for Mark Brown

- **Static Analysis**
  - Coverity
- Set up and maintained by Michael Ellerman
- The ‘original’ automated kernel build system
- Online (at least) since 2007
- Coverage
  - Most architectures (29)
  - Mainline, next, stable
  - Build only, no boot/runtime tests
Kisskb - Continued

- Used to generate weekly “Build regressions/improvements...” reports
- Build results available per e-mail on request
- Links
0Day: Overview

- Fengguang Wu’s brain child
- Operational since 2013
- Finds and reports
  - Build failures
  - Boot failures
  - Functional bugs
  - Performance regressions and improvements
- By far the most comprehensive test bed
0Day: How does it work?

- Devs new code
  - pull patches
  - SAME DAY turn around
- Create Defect Report
- Build tests/code analysis
  - Bisect to Isolate Defective Code
  - FAIL!
- Boot Tests
  - FAIL!
- Function Tests
  - FAIL!
- Performance Power Tests
  - FAIL!
- PASS!
0Day: Infrastructure

- ~80 servers
  - 18 build servers
  - Other servers used for runtime tests
- ~8 engineers
0Day: Coverage

- 683 Trees
  - Mainline, stable, stable-rc, next
  - Developer trees
- Detects ~1,200 daily branch changes
- Supports almost all kernel architectures
  - Exceptions: metag, arc, hexagon, unicore32
- Up to 2,000 test cases
0Day: Statistics

- 36,000 builds per day
- 150,000 runtime tests per day
  - ~8000 functional / performance / power tests
  - Remaining tests are boot/trinity tests in qemu
- ~800 build errors reported per month
- ~60 qemu boot failures reported per month
- 60% of failures reported within 2 hours
- 90% of failures reported within 24 hours
- Boot tests may require up to 1 week to complete
- Performance tests may require up to 1 month to complete
0Day: Challenges (from Fengguang)

● UI needs to be improved
● Runtime tests are noisy
  ○ Reporting delays (long runtime, system load)
  ○ Difficult to reproduce
  ○ Difficult to interpret
● High maintenance burden
  ○ Bugs, noisiness
  ○ Keeps the entire team busy
0Day: Links

- [https://01.org/lkp](https://01.org/lkp)
- [https://git.kernel.org/cgit/linux/kernel/git/wfg/lkp-tests.git](https://git.kernel.org/cgit/linux/kernel/git/wfg/lkp-tests.git)
- [https://lists.01.org/pipermail/kbuild-all](https://lists.01.org/pipermail/kbuild-all)
Kernelci: Overview

- Maintained by Kevin Hillman
- Operational since May 2014
- Goals
  - Wide range of Hardware
  - Quickly find regressions
  - Distributed
    - 9 board farms, with more coming
  - Framework independent
    - Most farms use Linaro LAVA
Kernelci: Coverage

- Mainline, next, arm-soc
- Stable, stable release candidates
- Various maintainer trees
- Arm, arm64, x86, mips
- All upstream default configurations, plus variants
  - 260+ configurations
- Build and boot; no runtime tests, no bisect (yet)
- Summary reports for stable release candidates
Kernelci: Statistics

- 9 Build farms
- 31 unique SoCs (arm, arm64, x86, MIPS)
- 260+ Configurations
- 200+ Unique boards
- 1000+ Builds per day
- 2300+ boots per day
Kernelci: Links

- [https://kernelci.org/](https://kernelci.org/)
- [https://lists.linaro.org/pipermail/kernel-build-reports/](https://lists.linaro.org/pipermail/kernel-build-reports/)
- #kernelci on IRC, Freenode
Kerneltests: Overview

- Created to test stable release candidates
- Operational since 2013
- Goals
  - Build all architectures
    - Reasonable snapshot of default configurations
  - Boot all available qemu emulations
  - Basic runtime tests (to be added ...)
- Runs on five PCs with i7 class CPUs
Kerneltests: Coverage

- **Branches**
  - stable-rc, mainline, next, hwmon, watchdog

- **Builds**
  - All architectures and variants
  - Up to 149 defconfigs

- **Boot tests (qemu)**
  - 14 architectures (+variants)
  - Up to 113 platforms

- **Summary reports for stable release candidates**

- **No runtime tests, no bisect, no individual reports**
Kerneltests: Statistics

- **Builds**
  - 15 branches
  - Up to 149 builds per branch
  - 39 architectures and architecture variants

- **Qemu tests**
  - 14 architectures, 8 variants (little/big endian, 32/64 bit)
  - Up to 113 platform boots per branch

- **Average 300-400 builds, 200-300 boots per day**
Kerneltests: Challenges

- UI
- Buildbot stability
- No long-term storage of test results
- Automated reports
- Automated bisect
- Maintenance
  - Toolchains
  - Qemu
- Operational cost
Kerneltests: Links

- [http://kerneltests.org/builders](http://kerneltests.org/builders)
- [https://github.com/groeck/linux-build-test](https://github.com/groeck/linux-build-test)
- [https://github.com/groeck/qemu](https://github.com/groeck/qemu)
Other Build and Test Systems

- **Mark Brown’s Buildbot**
  - x86_64, arm, arm64 (8 builds)

- **Olof’s Autobuilder**
  - mainline and next for arm, arm64, powerpc
  - ~120 configurations

- **Olof’s Autobooter**
  - mainline, next, arm-soc
  - ~75 boards (arm)

- **Tegra Builds**
  - Various Tegra builds and boots on mainline

Results reported at [https://lists.linaro.org/pipermail/kernel-build-reports/](https://lists.linaro.org/pipermail/kernel-build-reports/)
Coverity

- Detailed static analysis on Linux kernel
- Detailed defect reports and statistics
- Kernel contributors get free account to see results
- [https://scan.coverity.com/projects/linux](https://scan.coverity.com/projects/linux)
*** CID 1374326: Incorrect expression (NO_EFFECT)
/tools/objtool/arch/x86/decode.c: 102 in arch_decode_instruction()
96       insn.modrm.nbytes && insn.modrm.bytes[0] == 0xe5)
97     /* mov rsp, rbp */
98     *type = INSN_FP_SETUP;
99     break;
100    case 0x8d:

CID 1374326: Incorrect expression (NO_EFFECT)
Comparing an array to null is not useful: "insn.rex_prefix.bytes", since the test will always evaluate as true.

102       if (insn.rex_prefix.bytes && /* Should probably be insn.rex_prefix.nbytes */
103          insn.rex_prefix.bytes[0] == 0x48 &&
104          insn.modrm.nbytes && insn.modrm.bytes[0] == 0x2c &&
105          insn.sib.nbytes && insn.sib.bytes[0] == 0x24)
106     /* lea %rsp, %rbp */
107     *type = INSN_FP_SETUP
Coverity: Statistics

![Coverity Dashboard](image)

**Linux**

- **Overview**
- **Project Settings**
- **Members**
- **Invite**

### Analysis Metrics

**Version:** 4.8.0

- **Oct 16, 2016**
  - Last Analyzed
- **11,515,875**
  - Lines of Code Analyzed
- **0.49**
  - Defect Density

### Defect changes since previous build dated Oct 03, 2016

- **334**
  - Newly detected
- **285**
  - Eliminated

### Defects by status for current build

- **17,232**
  - Total defects
- **5,601**
  - Outstanding
- **1,422**
  - Dismissed
- **10,209**
  - Fixed

See how defect density for 'Linux' compares with defect density for other open source projects. [Learn more](#)
Coverity: Top Defects per Component
Coverity: Outstanding Defects

**Outstanding vs Fixed defects over period of time**

- Fixed defects
- Outstanding defects

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3,000</td>
<td>3,000</td>
<td>3,000</td>
<td>3,000</td>
<td>3,000</td>
<td>3,000</td>
<td>3,000</td>
</tr>
<tr>
<td>6,000</td>
<td>6,000</td>
<td>6,000</td>
<td>6,000</td>
<td>6,000</td>
<td>6,000</td>
<td>6,000</td>
</tr>
<tr>
<td>9,000</td>
<td>9,000</td>
<td>9,000</td>
<td>9,000</td>
<td>9,000</td>
<td>9,000</td>
<td>9,000</td>
</tr>
<tr>
<td>12,000</td>
<td>12,000</td>
<td>12,000</td>
<td>12,000</td>
<td>12,000</td>
<td>12,000</td>
<td>12,000</td>
</tr>
</tbody>
</table>
Coverity: Defect Density

The graph compares the defect density of the project with the average defect density of open source projects that are similar in size (i.e. more than 1 million lines of code).
Summary
● Test coverage has improved significantly over the last 2-3 years
● Test coverage still continuously improving
● The number of kernel bugs (per LOC) follows a downward trend
● People start paying attention to kernel bug reports
Not so Good

- Kernel stability still perceived as insufficient
  - Especially for stable releases
  - Need to further improve test coverage and quality of test reports
- Total number of open defects increases over time
  - Follows kernel code size increase
  - Need to analyze and fix outstanding bugs
- Sometimes it takes a long time for known bugs to get fixed
- No clear guideline how to handle false positives (especially from gcc)
- Not enough people actively engaged in ‘generic’ bug analysis and fixing
Next Steps

- Spread the word
  - Available test suites
  - Testbeds
  - Test coverage
- Improve test coverage
  - Especially but not only for stable releases
- Figure out how to better handle known positives
- Actually *fix* known bugs
  - Bug reports from autobuilders / autobooters
  - Bug reports from static analyzers
Next steps - continued

- Identify and track available test suites
- Improve test coverage
  - More functional tests (both in qemu and on real hardware)
  - Automatic bisect
  - Module tests
    - Implement and run
  - More testing on real hardware
- Improve test feedback
  - Automatic reports
  - Unified reporting
  - UI to pull test results
Thank You