CKI HACKFEST NOTES

NOTE: Please click the blue button at the top right if you want to request comment/edit access. 🥰

Introductions

Participants: *(please add your own name/affiliation here)*

- Red Hat:
  - Major Hayden
  - Veronika Kabatova
  - Nikolai Kondrashov <nkondras@redhat.com>
  - Don Zickus
  - Jakub Racek
  - Inaki Malerba
  - Rachel Sibley
  - Rafael Aquini
  - Bruno Meneguele
  - Eliska Slobodova
  - Stef Walter
- Linaro / LKFT:
  - Dan Rue
  - Anders Roxell
- kernelCI
  - Mark Brown (Arm)
  - Kevin Hilman <khilman@baylibre.com> (BayLibre)
  - Guillaume Tucker <gtucker@collabora.com> (Collabora)
- Fuego
  - Tim Bird <tim.bird@sony.com> (Sony)
- IBM
  - Andrew Donnellan (PowerPC + Patchwork/Snowpatch)
  - Daniel Axtens (PowerPC + Patchwork)
  - Russell Currey (PowerPC + Snowpatch)
  - Michael Ellerman (PowerPC maintainer, KissKB)
- Google
  - Dmitry Vyukov (syzkaller/syzbot)
- OpenXT.org
  - Rich Persaud <persaur@gmail.com>

Resources

KernelCI

Mailing list: kernelci@groups.io: https://groups.io/g/kernelci

Test data standardization

Topic summary: Current status of test result unification across CI systems, what defines test data and metadata.

Session lead: Tim Bird

Notes:

- Problems
  - Test systems are monolithic (tied closely to sub components in their own systems)
    - e.g. Jenkins, ttc, Lava, Beaker, buildbot, etc
  - Need more modular components that can be reused
- Similar APIs needed for components to talk to each other
  - Interfaces are wildly different between CI systems right now
- Tim/Kevin have a complex diagram for CI providers to line up their features to basic functions (the CI loop)
  - Test management system
  - Test scheduler / work farm
  - Device Under Test (DUT) control
  - Artifacts from build/run (run == results data)
  - Backend for reviewing data/results
  - Notifications
  - https://elinux.org/File:CI-Loop-high-level-arch-v3.jpg
- Many systems have implicit APIs (local operations, such as saving a file to a local filesystem)
- What's a test definition?
  - Metadata + instructions for running a test
  - Everything you need before, during, and after a test
- Tim has a quick comparison between Fuego/CKI in his slides
- Main focus on some things we all have in common
  - Metadata
    - Information about the test (license, version, etc)
    - Functional data (type, test format version, packaging manifest)
    - Fuego: YAML
    - CKI: Makefiles
    - **Lots of harmony here already**
  - Prerequisites & dependencies
    - Pre-conditions or state required for a test
      - Something that cannot be changed

            - Kernel versions, permissions, programs/libraries, capacity, etc.
        - Dependencies are things needed to execute the program
    - Results format
        - DB, JSON, xunit, etc
        - Everyone has parsers for figuring out what's passing/failing in a test log

Action items:

- Proposal: Each test emits parser helper data as part of log output
    - Could output a format type when it starts (as KTAP does with TAP header)
    - Apparently there's something else called TAP14?!
    - Q: Why do we want to write multiple parsers? Can't we work with test maintainers to get output written in common format?
        - Problems happen with test suites inside another test suite
        - Some tests use tabs for sub tests to make it easier to parse, but that's not in the TAP spec
    - Goal is to write a universal parser one time and reuse it everywhere
    - Would require work and coordination from test maintainers/authors
        - There needs to be an incentive for test maintainers/authors to do the work
    - Challenge with kernel developers who write a quick bash script but then have no idea how to add it to a test wrapper/framework/harness
    - Kernel developers want a framework to plug into (kselftest harness?)
- Proposal: Use Kernel CI JSON format for feeding common results into the same place
    - URL: https://api.kernelci.org/
    - Could help us find patterns for kernel versions and common test issues
        - Example: one LTP subtest could be flaky across all CI platforms; would be easier to find with results sent to the same place
    - Could be used for more advanced data mining/notifications later on
- Summary:
    - Make guidelines for test writers
    - Must have clear results data from tests (pass, fail, skip)
        - No manual reading/parsing
    - Use common output format with common parsers
    - **Tim:** Need to document what kernel is doing with/to TAP format (KTAP?)
        - Possible next steps: push the standard and start using it across the board. Developers won't need to learn a billion test standards/frameworks and results will be consistent as well
    - Discussion on KernelCI mailing list: https://groups.io/g/kernelci
    -

Common place for upstream results

Topic summary: There are plans to drop all available data into a common DB to use but what next? The place should be easily available and browsable by people wanting to see the results.

Notes:

- Previous discussion thread on the mailing list: https://lists.yoctoproject.org/pipermail/automated-testing/2019-April/000389.html
    - BigQuery solution proposed, CKI promised to provide data. No actual progress on implementation yet
- Possible starting point: schema (from kernelCI).
    - https://api.kernelci.org/schema-test.html
- syzbot schema (centered around crashes rather than tests, but some bits may be useful):
    - distilled version, or if you want to poke around actual code ([1], [2], [3])
- Greg receives multiple reports from several CI systems, can we unify the results into a single report.
- Greg would like a high level report summarizing pass/fail status for testing to easily find regressions
- Patchwork supports some of this result data via the checks API
    - This is already working well on the ppc list today (ozlabs)
    - But what about git trees/commits or the stable queue?
    - There's a kernel patchwork and one on ozlabs -- could be combined
- **Agreement: Reporting and data collection are separate things**
- BigQuery could be a place to upload data as well
    - Complex terms of service
    - Some companies won't want to publish anything, much less to a cloud provider's product
    - There are some options if we make the data public (free storage, perhaps?)
- Companies need a way to run some of these deployments internally for proprietary hardware
- Data store can be switched later; real value comes from sharing/comparing data
- Bigquery presentation on public data sets:
    - https://s3.amazonaws.com/connect.linaro.org/bkk19/presentations/bkk19-300k1.pdf
    - https://s3.amazonaws.com/connect.linaro.org/bkk19/videos/bkk19-300k1.mp4

Action items:

- Review the KernelCI schema to see if each group could push their data into it
    - Perhaps a command line tool to submit a result?
    - **Each CI system should do this review independently**
    - There is a metadata field for miscellaneous data
    - Discuss fields on mailing list that are required, not needed, etc
- **Each testing group should submit their data via KernelCI schema**
    - Tokens are required to push data into the API

    - Ask for a token on the KernelCI ML
- Leave the door open for moving to BigQuery possibly later

- **POC**: https://github.com/spbnick/kcidb

How to avoid effort duplication

Topic summary: How to get results from enough diverse environments (HW, compilers, configurations etc.) without duplicating others' work and still providing useful results.

Notes:

- From mailing list discussion: developers are not happy with receiving multiple "same" reports of questionable quality as having to go through all slows them down
- Nobody wants to write another reporter!
- What do kernel dev/maintainer want?
    - Wants to know when a patchset will bring a regression into a tree
    - Contributor should get a report before the maintainer sees it
    - Patchset dependencies are really hard but really important
        - Painful in downstream and upstream
        - Not a standardized way to define patch dependency relationships
    - Wants to know if the current tree is okay before applying patches
    - Bisection would be helpful
    - Collaboration between developer submitting patch and test maintainer would be helpful
- Compiler issues are fairly limited, mainly around warnings
    - Not a lot to worry with here
    - Report should include it for sure
    - Gets complicated fast with embedded
- Kernel configs should be a click away from the report
    - Everyone has their config fragments
    - Some configs are very opinionated
- Would kernel developers like to have the built kernel presented in a way that is VERY easy to boot and run (and debug?)
    - Yes, in all the formats (thanks, Bruno) ;)
    - Would be handy for testing patches
- Allow kernel developers to request additional tracing during testing, or crash dumps
- How do we avoid running the same tests in each CI system?
    - Is that a bad thing?
    - Greg doesn't like getting PASS/FAIL from different groups on the same test
- LTP takes a long time to run all of its test cases
    - Is it possible to run just one?
- Sample LAVA jobs:
  https://lava.collabora.co.uk/scheduler/job/1817668
  YAML definition: https://lava.collabora.co.uk/scheduler/job/1817668/definition
  Results summary: https://lava.collabora.co.uk/results/1817668
  https://lava.collabora.co.uk/scheduler/job/1818951 (more results from IGT)

Action items:

- Make a proof of concept of an easily runnable/bootable OS + kernel that a kernel developer/maintainer could use to boot their kernel and run it/debug it
    - As few commands/clicks maximum - less than 10 definitely, as few better
    - For build failures, make build environment easily accessible
- Need a way to annotate a component or a failed test
    - “mtest06 was badly written, no time for a fix, we should disable it”
    - Can be another field for test result to make things easy for now
- **We really really need to start centralizing our data to highlight where conflicting issues are located**

Open testing philosophy

Topic summary: Modularity and interoperability of different CIs: standardized API points to “share” different parts of CI systems, resource usage optimization, bisections, ...

Notes:

- KernelCI modular pipeline proposal: https://docs.google.com/document/d/15F42HdHTO6NbSL53_iLl77lfe1XQKdWaHAf7XCNkKD8/edit
- Modular pipeline diagram: https://docs.google.com/presentation/d/1-CrMZR3YtP7W7TETsouElgW1w4DNSgiherk0KIlyuhA/edit?usp=sharing
- Getting patches from a mailing list? Check out snowpatch: https://github.com/ruscur/snowpatch

Action items:

- The first step would be to send results to a common place, making it possible to have a common point for the last functional blocks (analyze, report)
- Then the next step may be to have a common place where builds are stored, and have various labs download these builds rather than do their own for the same upstream kernel revisions. Having a modular approach to how the builds are made would also help, rather than being all managed by KernelCI’s Jenkins.
- As KernelCI is making an abstraction above LAVA interaction, we could also be scheduling jobs in other lab types such as Red Hat’s Beaker to get CKI results directly integrated into the main pipeline and run tests using KernelCI builds.

Common hardware pools

Topic summary: Sharing hard-to-get HW with other CIs, companies being able to sign up to have stuff run on their HW to validate it.

Notes:

- From yesterday's discussion, we talked about this being a bit easier with a more modular pipeline approach
    - That also turned into a discussion around Beaker, Lava, other provisioning tools, etc and the limitations of each 😭
-

Action items:

- Find out if it's possible to translate Beaker to Lava and vice versa
    - Just different XML formats after all ;)
- Setup a public Beaker instance for kernel CI to play/develop a transition layer (Don Z)
    - Supply instructions for Beaker-in-a-Box

Getting results to developers/maintainers

Topic summary: How many emails to send and with what data (links to dashboard, how to reproduce the results, etc.). How much data should be sent in the first step and what is OK to be a few clicks away? Using Patchwork checks for tested patches that point to dashboard.

Notes:

- Strong preference for storing check data in Patchwork for patch series
- Two real audiences here:
    - Developer who submitted the patch
    - Maintainers who maintain the kernel
- What communication is being sent now?
    -
- What do maintainers want?
    - Pass/Fail obviously
    - What is it under test? Commit SHA(s), patches, etc.
    - New failures
        - Regressions first
        - Then show things that have been failing for a long time
    - Why did it fail?
        - Get to a crash dump, logs, built kernel, bisect output quickly
        - What was the test doing when it failed? Need to find out what it was testing and how it failed. A snippet of log or something could be helpful.
    - When did it pass again?
        - Reporting fixes is helpful
    - Maintainers don't want a long email full of passing test details, just failed tests
    - Links to extended report data are okay
        - Long emails are annoying
- Failures require some curation from the CI owners
    - Logs for tests are really long and it's difficult to isolate which log lines correlate to the failure
    - Logs look different for different tests
- Why can't there just be a single framework for tests?
    - Should be a best practices document for what tests should do/log
    - Should turn into a standard later
    - We have to go to where the tests are -- our teams aren't big enough to push test maintainers to change their framework
- kselftests is aimed towards getting more tests written; not looking pretty
- "Please tell me what the heck this test is testing"
    - Test has a name that is very ambiguous and a kernel developer doesn't know exactly what it's doing on the system
    - Tests need to have a purpose -- what are they intended to find or prevent?
- Do I want more test coverage or more organized/optimized tests?

    - Number of tests is only one dimension
    - Fewer tests with more hardware is also helpful
    - Trees, compilers, kernel configs are also valuable dimensions
    - Email reports make anything more than two dimensions difficult
- Need more tests but fewer test suites (would be easier to optimize around improvements)
- If more CI systems use kselftests and send those results to lists, then maintainers would push people to put tests in kselftests
    - CI systems need to use kselftests and promote it ASAP

[side conversation]:

- would it be useful to add meta-data directly into kselftest?
    - How to do it? What format? Where does it live?
- it would be good, but shouldn't require it of individual test authors
- Add it to the source as kerneldoc
    - extract it into json? and use as part of reports
    - have to define a schema
    - At a minimum, want the description of the test, and its dependencies
    - If not outputting in KTAP, describe output format
    - (things rejected: put it outside the tree, put it in yaml or json (kernel devs would reject it)

Action items:

- CI systems should start using kselftests and report results to maintainers ASAP
    - Dan Rue would like to work with anyone who is interested in this
    - Run latest kselftests on later versions of kernels (kselftest documentation will be updated soon to say that)
- Email content
    - Include in Kernel CI document eventually
        - Should be living document where maintainers could propose changes
    - Must include:
        - Pass or Fail
        - Commit SHA(s) used
        - Where any patches came from and how they were applied
        - Failing test names
        - Number of passing tests
        - Test environments / architectures
    - Should include:
        - *Will determine this as we iterate*
    - Should link to:
        - Kernel config files
        - Test sources
        - Built kernel artifacts
        - Reproducer
        - QEMU environment for testing
        - Versions of software (compiler, libc, etc)
            - Doesn't have to be super detailed: "GCC 8" or "GCC 9"
    - Ask maintainers if they want to get a pre-test notification

Onboarding new trees and tests

Topic summary: What trees need the most attention/are most bug-prone? Which tests are you missing in CI but find important? Upstream test location unification; making it easy to wrap and onboard tests to CI.

Notes:

- Onboarding new **tests** as it is today:
  - CKI
    - Review how the test is triggered, prerequisites, architecture requirements
    - Onboard it into the pipeline in a “waived” state
      - Waived means the test is not trusted
      - Once it stabilizes, the waived tag is removed
    - Tests are added into jobs for different hardware types
    - Newly onboarded tests aren’t always the most stable or they have narrow requirements
    - There is pre-CI for test changes
  - LKFT
    - Ask developer to add their test to kselftest or LTP :)
  - KernelCI
    - Worked with maintainers to determine how to onboard media tests
    - Worked through dealing with output
- LTP has a higher bar of quality than kselftest for now
- Red Hat has two LTP maintainers who would be happy to talk about making changes to LTP
- Sometimes it’s better to let a project be wild for a while (kselftest) so we don’t stifle innovation/adoption
- We all need curators and librarians for new tests
- New test suites need to get “wrapped”
  - Each system has something like this right now
  - LKFT doesn’t have the waived concept, but would like to have it
- Can we (CI systems) influence these test developers to change how they report?
- If there’s a reason why people are not pushing tests into LTP, we should get those deficiencies fixed
- Onboarding new **trees** (as it is today):
  - To onboard a new tree to test with CKI, submit a PR to https://gitlab.com/cki-project/pipeline-data/ we can help with format if needed
- How do we work with a kernel maintainer to add their tree?
  - Ask them which dimensions they care about: architectures, test suites, compiler version, compile options, kernel configs, kernel tree, branch of tree, etc
  - What tests do you care about, how do we run them, and where are they?
    - Consider trees like media, graphics, audio -- might have highly specialized test
  - Kernel maintainers should soon define what tests they want in the maintainers file (yay!)
  - Kernel maintainers should be able to see what they’re going to get

        - Consolidated reporting could make the value offering better
    - In the end we want kernel developers/maintainers **to care** about what the CI systems are doing
- What is success?
    - **Everyone needs to watch Dmitry's talk from LPC 2019**
    - More bugs are being fixed
    - Subsystems are not broken for long periods
    - Developers/maintainers care about breaking (and unbreaking) the build
    - Maybe Linus will tell maintainers to have CI at some point (we can hope!)
    - It all starts with maintainers using CI and talking about it with other maintainers
- What's the workflows mailing list? (it's new as of 2019-09-12)
    - No outcomes for testing
    - It was assumed that we (Kernel CI) will handle that ;)
    - No strict no or yes on anything
    - Identity and common data formats coming soon (possibly)
    - Structured data formats for patches and suggested changes (communication protocols for CI to say "we tested this" other than sending email)
    - Someone even suggested web UIs :)
    - Intel's 0-day is slowing in responses and maintainers want to know if the testing started

Action items:

- Find a way to interact with test maintainers
    - Recommended to use automated testing list since LTP authors are already there
    - Call out deficiencies on the LTP list directly
- Articulate what we want to make it easier for kernel developers to contribute to LTP
    - Current coding best practices are not user friendly
    - There is confusion about doing pull requests or submitting patches via email
    - Red Hat to work with developers to on-board tests in LTP and work with upstream to solve issues.
- Define what success looks like for all of us
    - Write it down publicly

CI bug tracking

Topic summary: Is bugzilla.kernel.org useful to report bugs from CI? Is anyone actually checking and fixing stuff that's there in a timely manner? Would it make sense to have a bug tracker for possible issues found by CI? Automated bug submission on failed tests.

Notes:

- imalerba: proposal to use GitLab project issues to track bugs
    - Simple API, a lot of labels to use to mark progress
- Emails are fine to announce a bug was found + details but it's impossible to track all submitted bugs by a CI system, comparing them with possible duplicates by other CIs etc.
- Is bugzilla.kernel.org useful?
    - Resounding no from the group
    - BZ sends emails to the list, maintainers want replies by email
    - Sending an email to the ML is preferred because it doesn't force maintainers to deal with a separate system
    - If CI sends a bug to the maintainer, the maintainer might make a bug in BZ to track the fix if it takes time
- syzkaller sends emails, but then maintainers want all the features of BZ in syzkaller's dashboard
- If CI opens a bug, then there needs to be some sort of closure when the bug no longer exists
- Some CI systems need to demonstrate progress through metrics to maintain funding and grow
- syzkaller asks developers to add a tag in there patch to do correlation
    - Some developers do it, some don't
- Having an ID/UUID is handy for tracking fixes

Action items:

- Put the bug report tracker/ID/URL in the data submitted by each CI system
    - The link goes in the shared data stored centrally
    - Each CI system can track their own independently
    - As we start collecting shared results/failures, we can look for common ways of combining and/or de-duplicating failures and coming up with a common tracking ID

Bugs and result interpretation

Topic summary: Test result interpretation, regression/fix detection, infrastructure issue detection, using neural networks to detect and categorize issues, “known issue” detection...

Notes:

- Tim Bird: skiplists and pass criteria
- Carlos Hernandez: using neural networks for bug detection
- Sasha has some nice machine learning working with the stable tree right now to identify patches which are easily backportable or are critical security/bug backports
- Would be nice to have a "confidence rating" for individual boards in a lab, that can be used to rate test results
    - Based on how flaky the board hardware is
- Kevin suggests using emojis to report relative flakiness
    - 😀🙂🤨😐😣😠😡💩

Action items:

- Stef has a link to some machine learning application inside a container that is worth checking out
    - Code: https://github.com/cockpit-project/cockpituous/tree/master/learn
    - DevConf Talk: https://devconfcz2019.sched.com/event/Jcg0/using-machine-learning-to-find-linux-bugs
        - Slides: https://github.com/stefwalter/slides-machine-learning-bugs
        - Video: https://www.youtube.com/watch?v=vU6KeUEWmsQ

Security when testing untrusted patches

Topic summary: How do we merge, compile, and test kernels that have untrusted code in them and have not yet been reviewed? How do we avoid abuse of systems, information theft, or other damage?

Notes:

- Problems/threats:
- **Treat anyone/everything as hostile**
- Identity helps with knowing that a developer really submitted a certain patch
- Syzkaller currently does:
    - Build kernel with new local user
    - All builds in its own user/network namespace
    - Blacklist certain sender email addresses / regular expressions
    - Partition build machines away from other systems
- How do we handle the hardware tests?
- What if someone submits code that is patented or poorly licensed and you're distributing it?
- Multi-level approach:
    - Trusted person: run on hardware
    - Untrusted person: run in a VM
    - Identity verification is still required
- Incentivize developers to submit w/identity or signatures if they want more testing
    - Still needs to be as easy as possible with least delays
- Developers also may want to compromise test results (to look okay when it's not okay)
    - Think about certification for autonomous cars
- Perhaps do VM testing for untrusted patches and deeper tests on signed tags from maintainer
- Hardware could be exposed into VMs via SR-IOV
- Attackers could modify system component firmware or BIOS
    - Signed BIOS?
- How effective are tests in VMs?
- Could limit what gets tested based on which files are changed
- Maybe you just don't build/test
    - Ignore whitespace changes or changes to comments
    - Ignore changes to architectures which aren't ones you care about
- Do reproducible builds and compare results to know if anything changed

Action items:

- Don to verify DCI/Beaker publicly documents best practices for a lab network.

Additional topics

Compat (32b userspace and 64bit kernel)

Notes:

- distros use this commonly, but it's not often tested
- also a common attack surface

Actions:

Events

Linux.conf.au 2020 Kernel Miniconf - 1 day stream as part of LCA2020, 13-17 January, Gold Coast, Queensland, Australia. Andrew will be putting out a CFP very shortly - see http://lca-kernel.ozlabs.org/. Testing/CI/process talks very welcome.
(Actions: Everyone: submit an overwhelming number of papers)

Platform Security Summit 2019: 3 day single track, focused on hardware-assisted security and open+closed software supply chains, Oct 1-3 near Seattle, WA, USA. Kevin may be able to speak about CKI.
2018 videos (TPM2,firmware,boot): https://platformsecuritysummit.com/2018/videos
2019 speakers: https://platformsecuritysummit.com