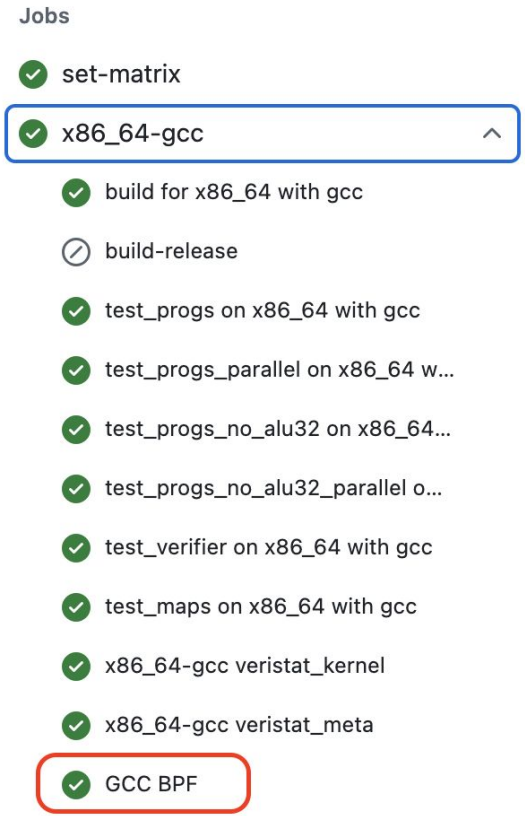
BPF CI

LSF/MM/BPF 2025

Ihor Solodrai

Overview

- Testing GCC BPF
- Testing sched_ext
- Autoscaling kernel builds
- s390x maintenance
- Github Actions maintenance
- How to trigger BPF CI for your change
- /discuss



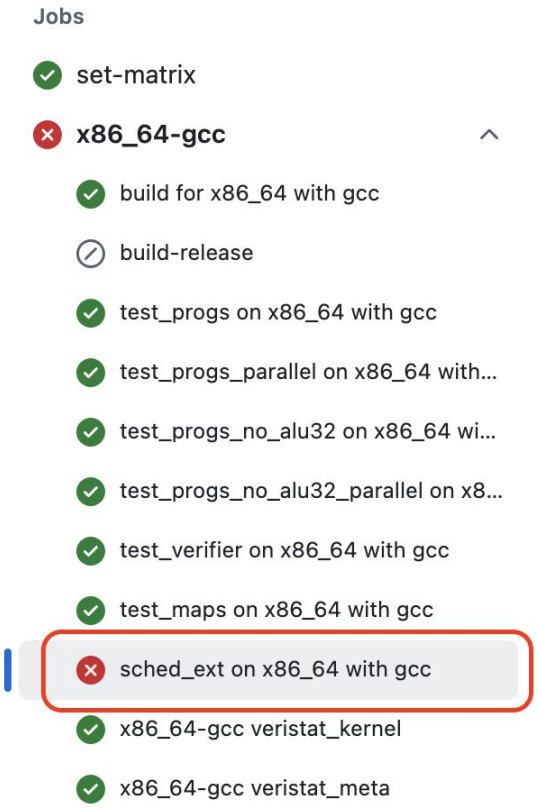
GCC BPF

1. Build kernel
2. Download latest build of GCC snapshot for BPF target (built once a week)
3. Build tools/testing/selftests/test_progs-bpf_gcc
4. OK, if build successful

This means that GCC successfully produced .bpf.o from selftests BPF programs.

About half of the tests fail, so we don’t run them yet.

lore.kernel.org/bpf: “Announcement: GCC BPF is now being tested on BPF CI”



sched_ext

1. Build kernel
2. Build tools/testing/selftests/sched_ext/runner
3. Execute the runner

Not enabled by default yet.

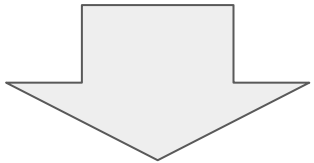
But caught some bugs already!

lore.kernel.org/bpf: “selftests/sched_ext: testing on BPF CI”

Autoscaling kernel builds

Before:

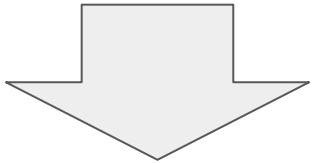
“build x86_64 kernel”
“build x86_64 kernel with LLVM”
“build s390x kernel”
“build aarch64 kernel”
“run x86_64 tests”



x86_64 runners
4 x c5.metal on AWS

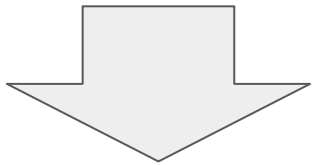
After:

“build x86_64 kernel”
“build x86_64 kernel with LLVM”
“build s390x kernel”
“build aarch64 kernel”



AWS CodeBuild
(autoscaled)

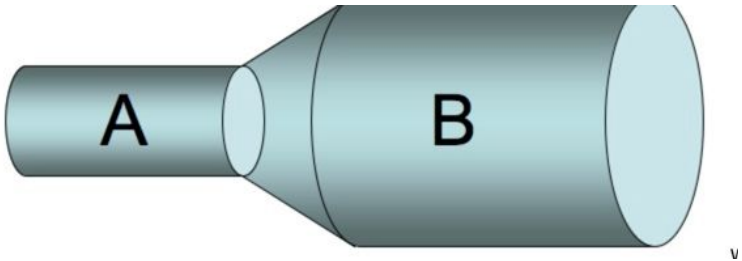
“run x86_64 tests”



x86_64 runners
4 x c5.metal on AWS

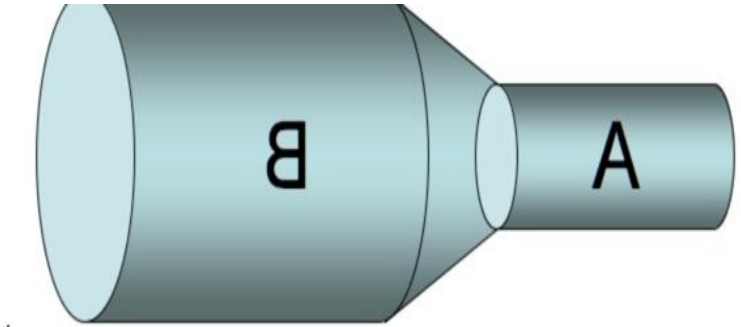
Autoscaling kernel builds

Before:



A: build kernel
B: run tests

After:



□: build kernel
A: run tests

s390x runners

- Custom-built s390x runner binaries are now used instead of binfmt emulation
  - Github does not release s390x build of the official Github Actions runner (.NET app)
- selftests performance degrades on high load (this is community cloud VMs)
- Maintenance is only somewhat automated
- s390x was removed from libbpf CI

Q: Is it worth it? Any alternatives?

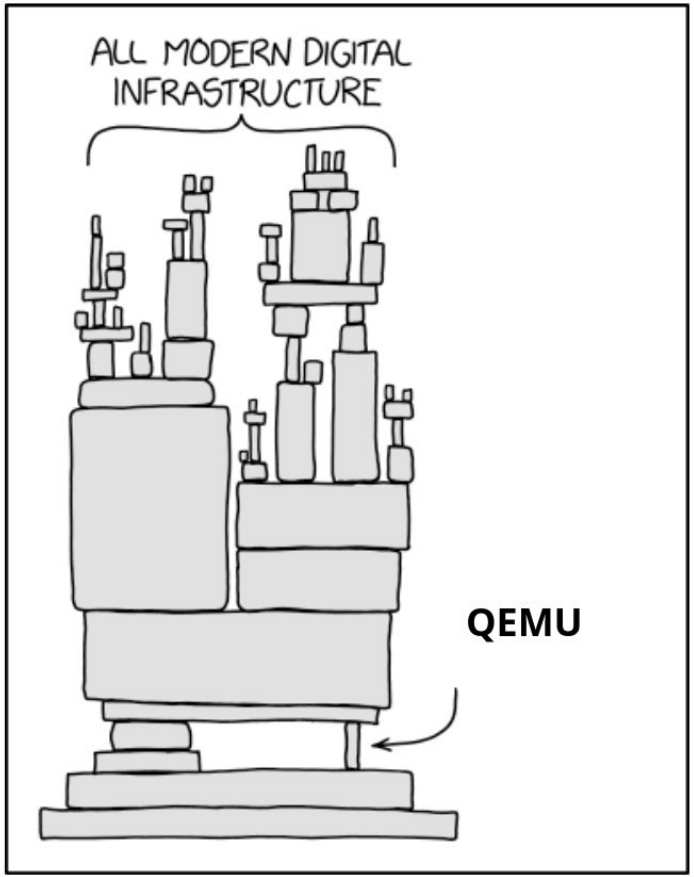
Maintaining Github Actions code

- bash > yaml
  - Don’t like bash? How about python?
    - bpftrace/.github/include/ci.py (kudos to Daniel Xu)
- env variables > action inputs/outputs
- actions/cache isn’t always a good idea
- use sparse checkouts
- “reusable workflows” in github actions are meh (at least so far)
- writing reusable pieces as “actions” works pretty good
  - beware of sneaky dependencies though

Maintaining Github Actions code

When using public actions and/or docker for testing on different architectures, remember that it’s not magic: it’s QEMU.

Link: kernel-patches/runner#67



How to trigger BPF CI for your change

1. `gh repo fork https://github.com/kernel-patches/bpf`
2. `git clone git@github.com:${GH_USERNAME}/bpf.git`
3. `git checkout -b your-feature`
4. `Edit code`
5. `git commit your-changes`
6. `git push`
7. `gh pr create`

/discuss

- What things are not tested but should be?
  - Newer/older LLVM/gcc releases?
  - LST Kernels?
  - 32-bit architectures?
- Reproducing the failures
  - How often do you do it?
  - How difficult is it usually?
- Upstream merges or other dependencies tend to break CI. What can we do?
- How much do we actually care about CI job speed?
  - Waiting 20 mins vs 25: does it really make a difference?