Externally verifying Linux deadline scheduling

Theodore Tucker

ROLE

Software Engineer

TENURE

Joined Codethink in 2024

EDUCATION

Still studying:
BSc Combined STEM
Open University

Contents

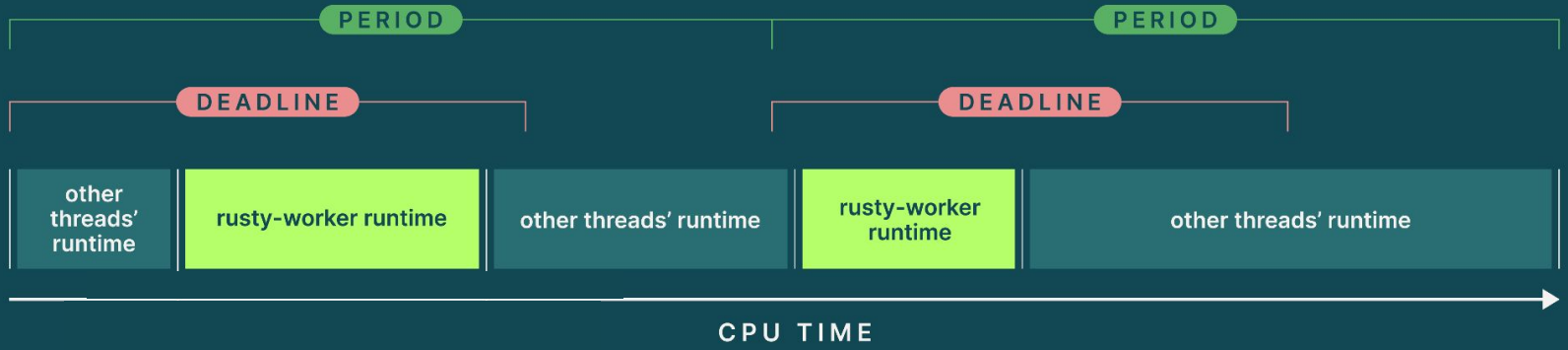
Linux deadline scheduling

- Starting point: **multithreaded Linux-based OS** on **multi-core processor** in **safety-related context**
- Interactions of threads, cores, and external interrupts make behaviour **difficult to predict**...
- ... but threads must meet hard **real-time** performance constraints
- Working towards certification of scheduling as a safety function under **IEC 61508**
- We need to rigorously measure that performance on real hardware for every iteration of the OS image

Linux deadline scheduling

- Linux kernel provides **`SCHED_DEADLINE`** policy (since 3.14) to model and implement real-time scheduling requirements for threads
- Employs earliest deadline first (EDF) algorithm to share CPU time between competing threads
- When a real-time thread starts, it provides its real-time requirements to the kernel with `sched_setattr` syscall
- `sched_setattr` arguments:
  - **Period** (ns) – *how often should I execute?*
  - **Deadline** (ns) – *when in the period should I have finished executing by?*
  - **Runtime** (ns) – *what is my worst case execution time each period?*
- On each `sched_setattr` call, the kernel performs the **admission test**
- *If I accept this thread, can I still share CPU time to meet its and every other thread's demands?*

Linux deadline scheduling

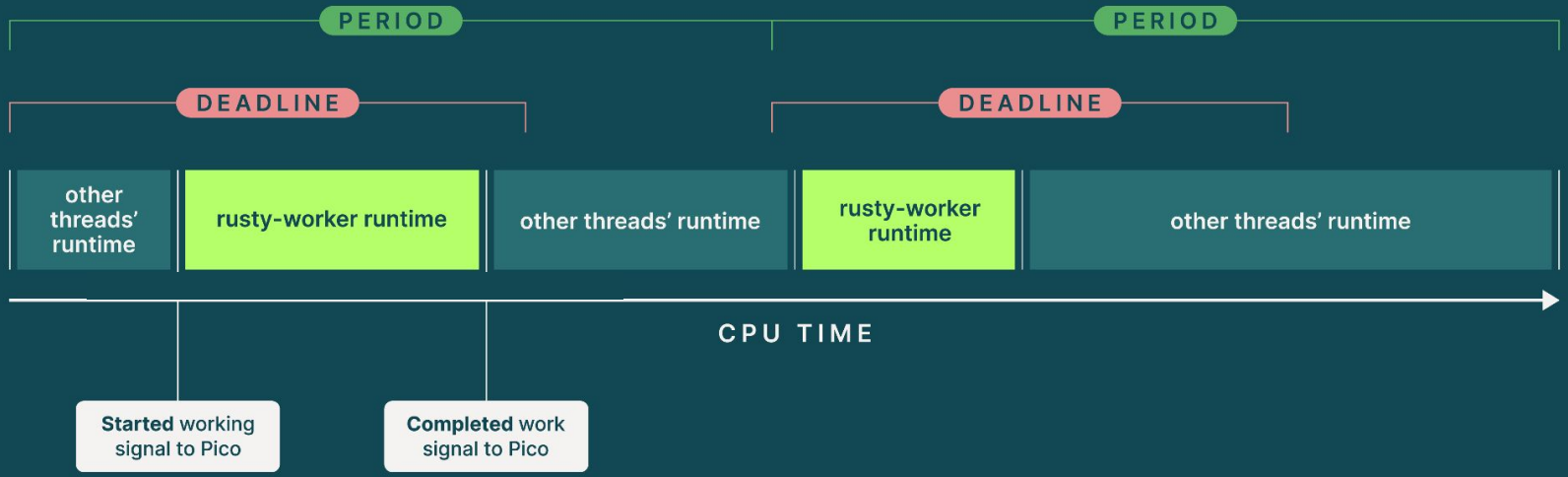


CTRL OS

- Codethink Trustable Reproducible Linux
- Based on latest Linux kernel and **Freedesktop-SDK** userland
- Reference target hardware: **Intel NUC 11TNKv5** (amd64) and **Radxa Rock 5B** (aarch64)
- Testing images integrate **rusty-worker** – a Rust program which exercises the deadline scheduler

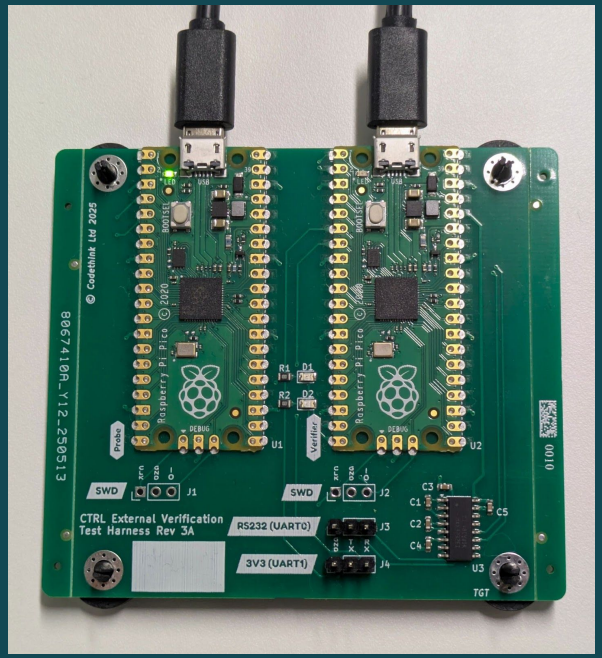
- Starts as a `SCHED_OTHER` “monitor” thread
- Monitor thread starts a `SCHED_DEADLINE` “worker” thread with user-configurable arguments
- Monitor thread uses Linux’s monotonic clock to measure and report the “true” runtime and period of the worker thread
- Measured runtimes and periods reported to an onboard **Safety Monitor** to cause an external mitigation on failure
- Measured runtimes and periods also reported to **systemd journal** for offboard post-test analysis
- But... the deadline scheduler relies on the **same monotonic clock** that rusty-worker uses to measure its performance

Externally measuring deadline scheduling



Test harness

- KiCAD-designed custom PCB
- 2x Raspberry Pi Pico: a **verifier** and a **probe** to flash the verifier with verification firmware at test start
- The **verifier:**
  - receives UART start/finish signals from DUT at TTL or RS232 levels (shifter onboard)
  - computes min/mean/max period/runtime over a 10-second window
  - reports measurements to test runner over USB CDC
- The **probe:**
  - runs RPi Foundation's **debug-probe** firmware
  - receives binaries from the test runner over USB (CMSIS-DAP)
  - flashes binaries to the verifier Pico over ARM SWD



Verification firmware

- Written in Rust with **embedded-hal** and **rp2040-hal** crates
- **Interrupt** triggered when signal received on UART: signal queued with monotonic timestamp *(µs precision)*
- In main loop, signals are dequeued, scheduling parameters calculated, and reported as YAML over USB CDC
-
```
Runtime:
  samples: 999
  min: 322
  mean: 335
  max: 355
Period:
  samples: 999
  min: 9997
  mean: 9999
  max: 10001
window-uid: DFAA4DA88620B17E
```

- Rust requires more **verbose** peripheral initialisation
- Rust provides **compile-time guarantees of type- and memory-safety**, and **deeper understanding of hardware**
- Rust compiler **natively supports cross-compilation** for the RP2040 – no special GCC version needed
- Easier to **reproducibly build the verification firmware binary** with a reproducible compiler
- CTRL OS uses **Apache BuildStream** to define and build OS and CI images...

Verification firmware BuildStream definition *(abridged)*

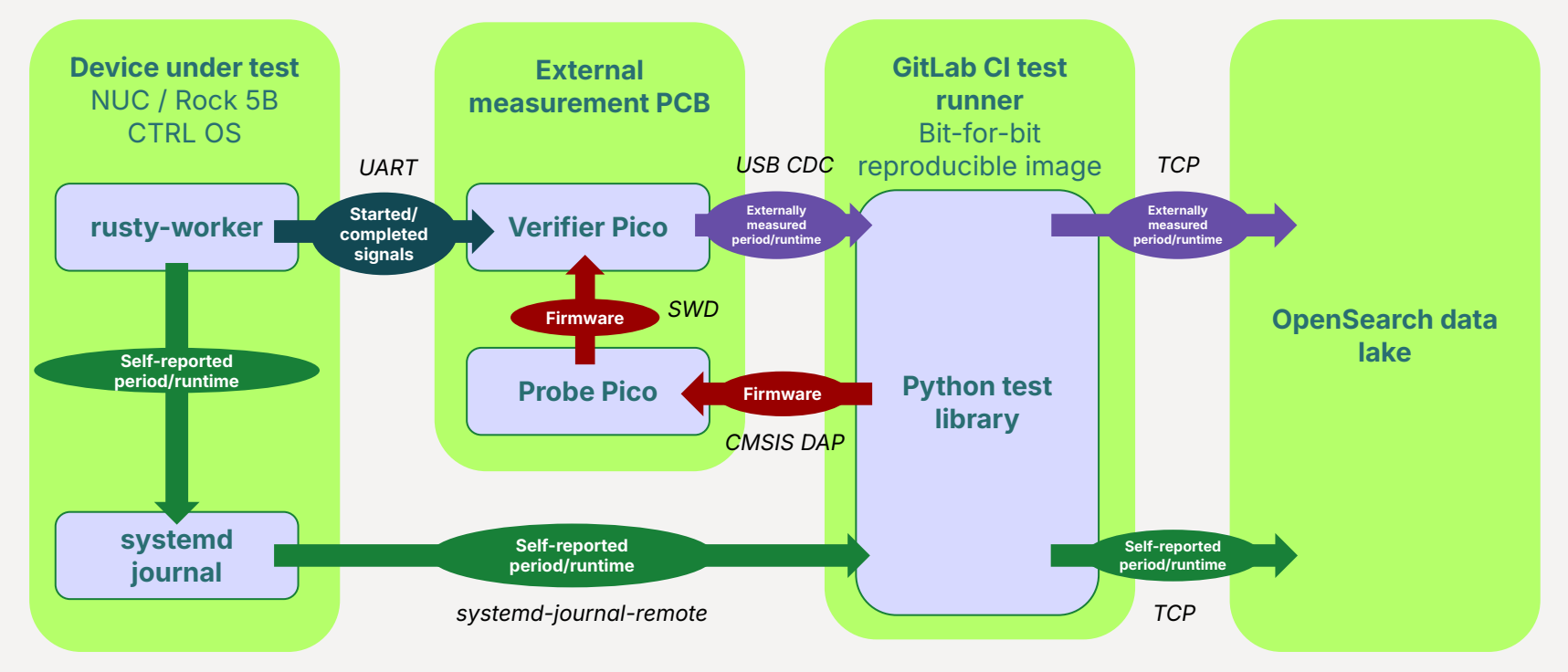
```
kind: manual

build-depends:
- components/flip-link.bst
- components/rust-embedded.bst
- freedesktop-sdk.bst:components/gcc.bst

config:
  install-commands:
  - cargo build --release --features uart0
  - install -D -m 660 target/thumbv6m-none-eabi/release/vfrs_scheduling
%{install-root}/opt/verification-firmware-rs/vfrs_scheduling_uart0

sources:
- kind: git_repo
  url: sif:platform/verification-firmware-rs.git
  track: '*.*.*'
  ref: 2026.01.05_4c619d4c75dbfc81b58d8ea4a63bc4a0c2ac316a-0-g4c619d4c75dbfc81b58d8ea4a63bc4a0c2ac316a
- kind: cargo
  ref:
  - name: arrayvec
    version: 0.7.6
    sha: 7c02d123df017efcdfbd739ef81735b36c5ba83ec3c59c80a9d7ecc718f92e50
  - name: bare-metal
    version: 0.2.5
    sha: 5deb64efa5bd81e31fcd1938615a6d98c82eafcbcd787162b6f63b91d6bac5b3
  # etc.
```

Wrapper around rust-lld linker to alter memory layout for ARM Cortex-M: ensures stack overflows result in a handleable error rather than overwriting static variables

Rust compiler with `thumbv6m-none-eabi` architecture enabled at build-time in `config.toml`

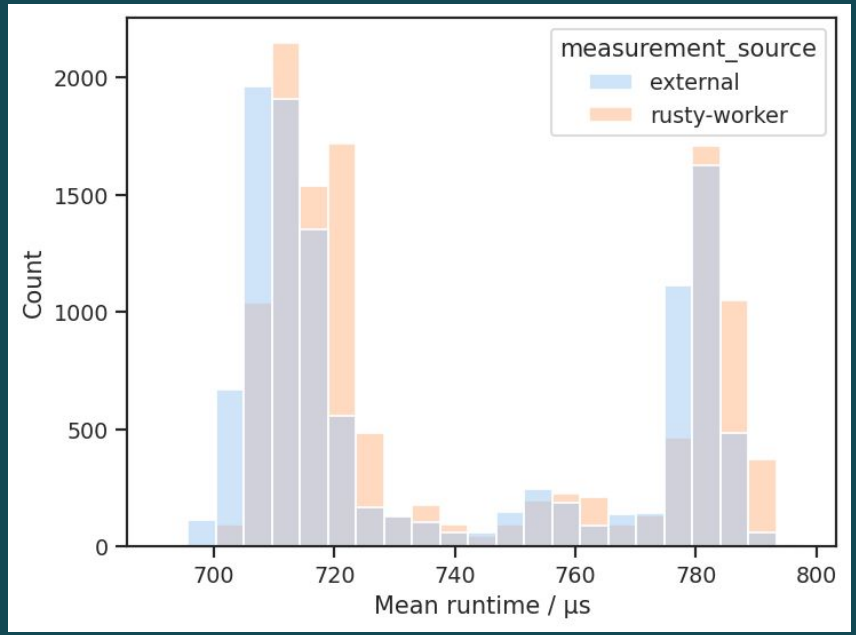
Versions and hashes provided for all input code and libraries: aids SBOM generation

Test architecture



Results

- **According to external measurements, self-reported measurements are accurate**
- **NUC**: paired measurements within **+/- 100 µs**
  Rock 5B: paired measurements within **+/- 10 µs**
- Analysis performed in **Jupyter notebooks** querying data from OpenSearch and applying scientific Python libraries
- We have confidence rusty-worker is accurate, so we can trust its data when arguing CTRL OS's scheduling ability



Results

- External measurements performed for **every iteration of CTRL** build definitions in **GitLab CI** over **15 months**
- Changes in OS reflected in these measurements – profound understanding of kernel behaviour

1. External measurements more accurate on Rock 5B than NUC
   Rock 5B's TTY device is on SoC, NUC's is a PCIe peripheral
2. External measurements less accurate when `PREEMPT_RT` enabled
   kthreads can be interrupted when writing to a TTY device, so time taken to get bits on the wire is non-constant

Continuous compliance: Eclipse Trustable Software Framework (TSF)

Goals:

- Link external measurement results to our expectations of CTRL OS
- Perform tests and analysis automatically for every CTRL OS commit
- Communicate the results and implications to engineers and stakeholders

– Eclipse **Trustable Software Framework** provides both **high-level expectations** for software and **analysis tooling**

– **Expectations** are linked to **evidence** in a **directed acyclic graph (DAG)**, with each node having a **score** [0.0, 1.0]

– **Evidence** may be scored by **Python "validators"** which query OpenSearch and analyse test results

– CI generates a **HTML report** containing scores for each commit's test results, trackable over time

CTRL-EXTERNAL-SCHEDULING-MEASUREMENT-MEAN_PERIOD_HISTORY_NUC | Reviewed: ✔ | Score: 1.0

The absolute error in the critical process's mean period compared with external measurements is less than 100 µs for this SHA of CTRL and its parents in the last 30 days on NUC hardware.

Supported Requests:

Item	Summary	Score	Status
CTRL-EXTERNAL-SCHEDULING-MEASUREMENT-Performance_HISTORY_NUC	The external measurements of the critical process's scheduling are consistent with the critical process's own reports for this SHA of CTRL and its parents in the last 30 days on NUC hardware.	0.98	✔ Item Reviewed ✔ Link Reviewed

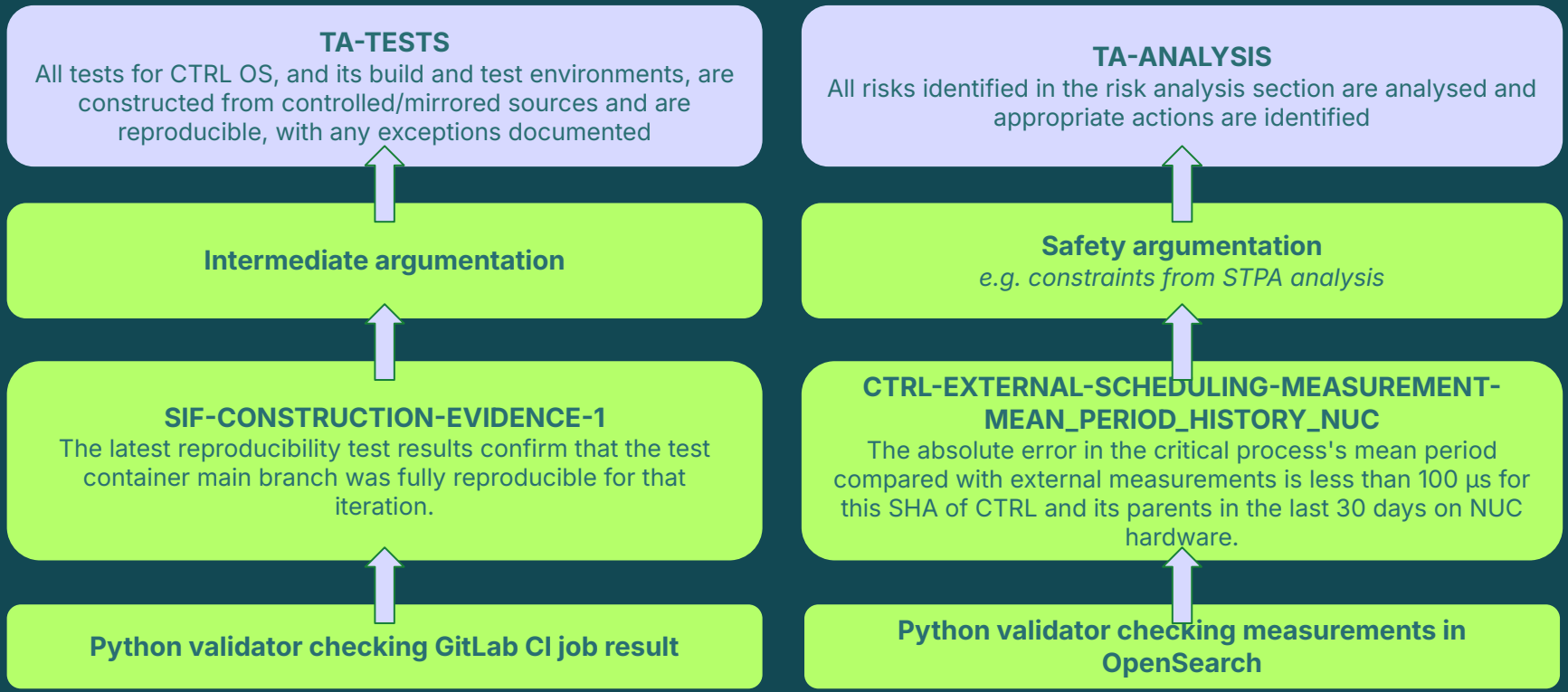
Supporting Items:

None

Validator:

Validator Score: 1.0

Argumentation



Conclusion

- External measurements provide confidence in our analysis of Linux deadline scheduling and deep understanding of the kernel
- With open-source software and hardware components, from compilers to infrastructure, we orchestrated complex hardware-in-the-loop tests as part of regular CI pipelines without sacrificing provenance
- The **Trustable Software Framework** allows those test results to continuously support our defence of Linux as a feasible RTOS to engineers, stakeholders, and safety assessors

https://gitlab.com/theodoretucker_ct/slides

https://gitlab.com/CodethinkLabs/trustable/trustable

Thank You.
Questions?

Codethink Ltd.

3rd Floor Dale House,
35 Dale Street,
MANCHESTER,
M1 2HF,
United Kingdom

theodore.tucker@codethink.co.uk

connect@codethink.co.uk