RV and the Deadline Scheduler

How to verify your subsystem at runtime

Gabriele Monaco
Senior Software Engineer

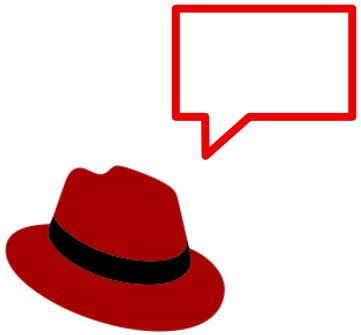
Juri Lelli
Senior Principal Software Engineer

東京 2025 Linux Plumbers Conference
TOKYO, JAPAN / DEC. 11-13, 2025

Red Hat

I wish I can validate the deadline scheduler ...

(check implementation vs. “mental” model)



Subsystem Maintainer

I wish I can validate the Deadline Scheduler

WIKIPEDIA
The Free Encyclopedia

Search Wikipedia | Search

Join Wikipedia Asian Month this November and December!
Contribute in Wikipedia Asian Month and get a postcard!

Runtime verification

1 language

Contents hide

(Top)
History and context
Basic approaches
Examples
HasNext
UnsafeEnum
SafeLock

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

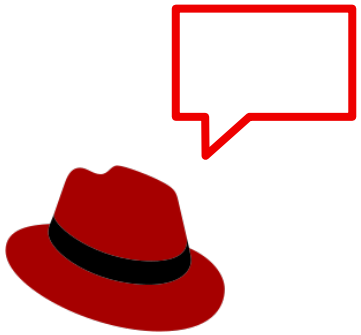
Runtime verification is a computing system analysis and execution approach based on extracting information from a running system and using it to detect and possibly react to observed behaviors satisfying or violating certain properties.[1] Some very particular properties, such as datarace and deadlock freedom, are typically desired to be satisfied by all systems and may be best implemented algorithmically. Other properties can be more conveniently captured as formal specifications. Runtime verification specifications are typically expressed in trac
temporal logics, etc., or e

bootlin
Elixir Cross Referencer

/ Documentation / trace / rv / runtime-verification.rst | All symbols | Search Identifier

```
1  ====================
2  Runtime Verification
3  ====================
4
5  Runtime Verification (RV) is a lightweight (yet rigorous) method that
6  complements classical exhaustive verification techniques (such as *model
7  checking* *and* *theorem proving*) with a more practical approach for complex
8  systems.
9
10 Instead of relying on a fine-grained model of a system (e.g., a
11 re-implementation at instruction level), RV works by analyzing the trace of the
12 system's actual execution, comparing it against a formal specification of
13 the system behavior.
14
15 The main advantage is that RV can give precise information on the runtime
16 behavior of the monitored system, without the pitfalls of developing models
17 that require a re-implementation of the entire system in a modeling language.
18 Moreover, given an efficient monitoring method, it is possible to execute an
19 *online* verification of a system, enabling the *reaction* for unexpected
20 events, avoiding, for example, the propagation of a failure on safety-critical
21 systems.
```

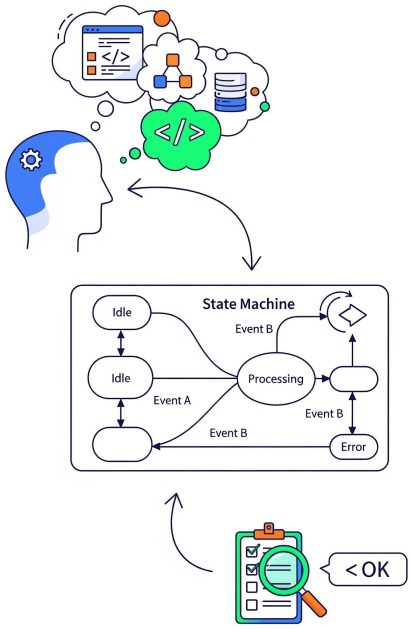

Can we validate the deadline scheduler and server with RV?



Subsystem Maintainer

Expectations

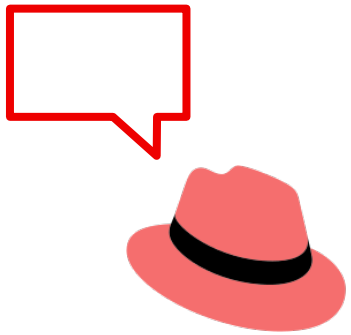
- (I think!) I have a mental model of how single task DEADLINE and dl-server entities behave
- I would like to describe them more formally (state machines?)
- Then check, by collecting some information at runtime, that the system is working according to my description



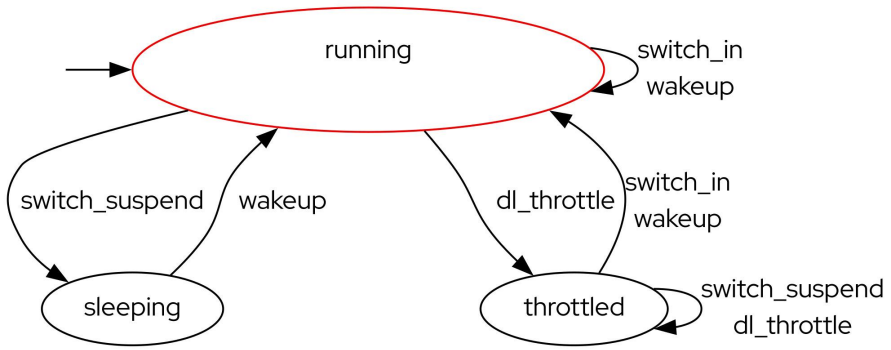
Specifications Ideas

- Deadline tasks don't use more runtime than assigned
- Deadline tasks don't miss their deadline
- If fair tasks are ready they either run autonomously (NORMAL priority) or boosted by the server (DEADLINE priority)

We can use timed automata for these models.



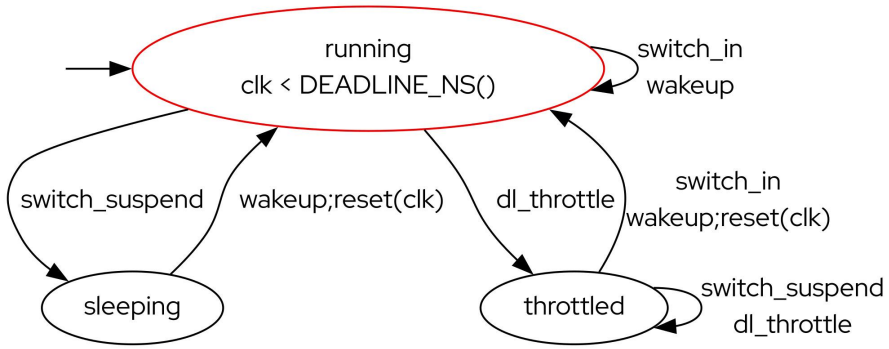
Timed Automata in RV



nomiss model

- Extension of deterministic automata
- State machines where next state depends on current and event
- Events typically tracepoints
- Runs entirely in the kernel

Timed Automata in RV

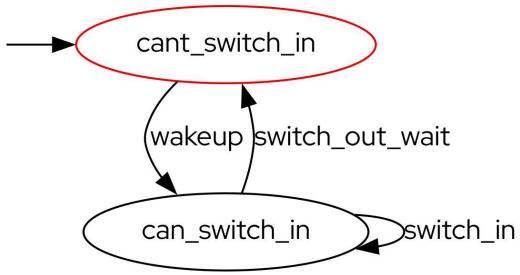


nomiss model

- States and edges can have constraints on variable (e.g. a clock)
- Special action *reset()* restarts a clock
- Constraints determine if transition is valid or when state is valid

Violations

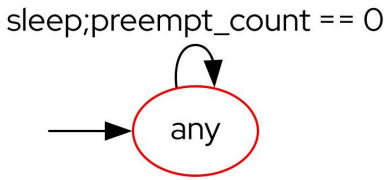
Transitions not allowed by models



Event is not expected in a state

Event *switch_in* in state *cant_switch_in*

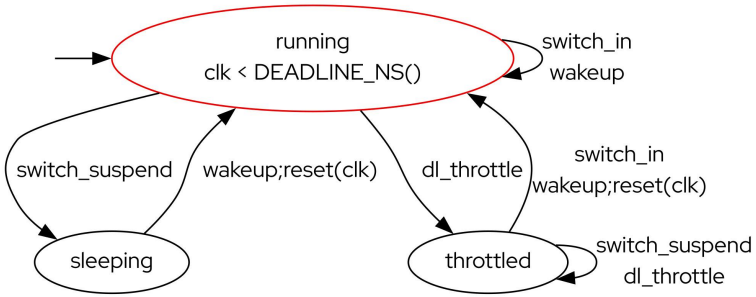
(*switch_in* arrow missing in that state)



Event with false constraint

Event sleep when preempt_count > 0

(constraint checked during event)

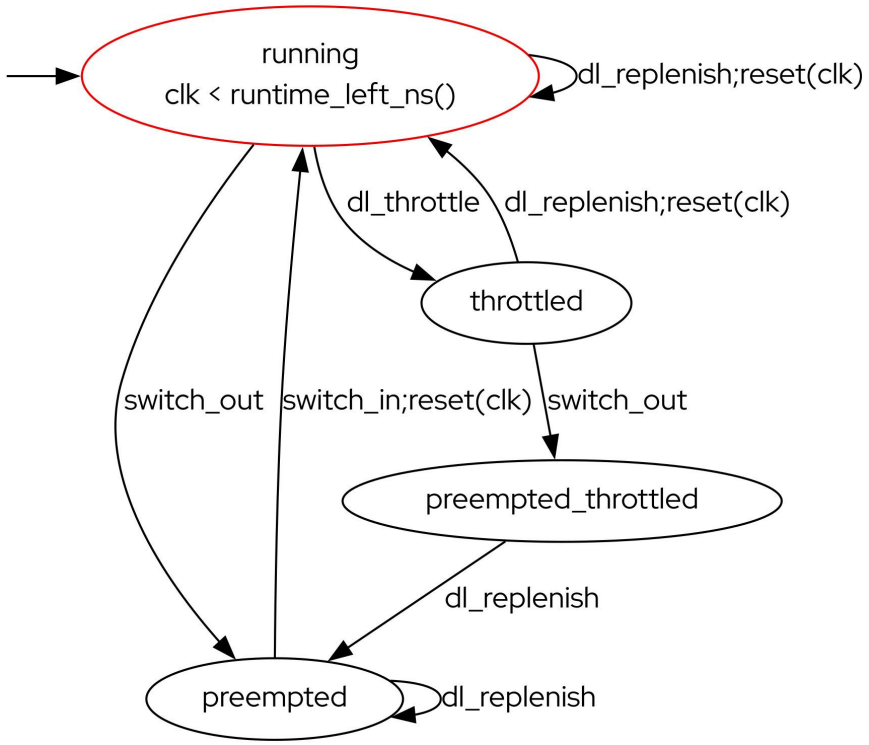


Still in state when constraint is false

Still in running with clk > DEADLINE

(constraint checked after timer or when leaving state via other events)

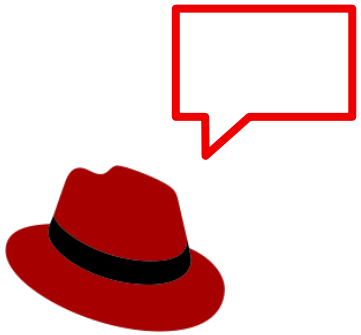
Monitor Instances



throttle model

- States apply to each task
- Could be a per-task monitor
- What about deadline servers?
- Monitor for each deadline entity

We are going to need new tracepoints.

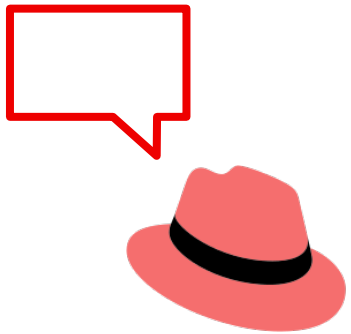


Subsystem
Maintainer

Deadline-specific Events

- Throttle - deadline entity depletes its runtime
- Replenish - runtime replenishment timer fires (or CBS wakeup rule)
- Server start/stop server - first FAIR task enqueued/server idle
- Probably enough for now?

Here's a patch with timed automata and deadline models.



[PATCH] rv: Add Hybrid Automata and Deadline Monitors

Extend RV to support new monitors

Hybrid automata

Timed automata are a special case of those with clocks as environment variables

Per-object monitors

Track generic objects like deadline entities

Deadline tracepoints

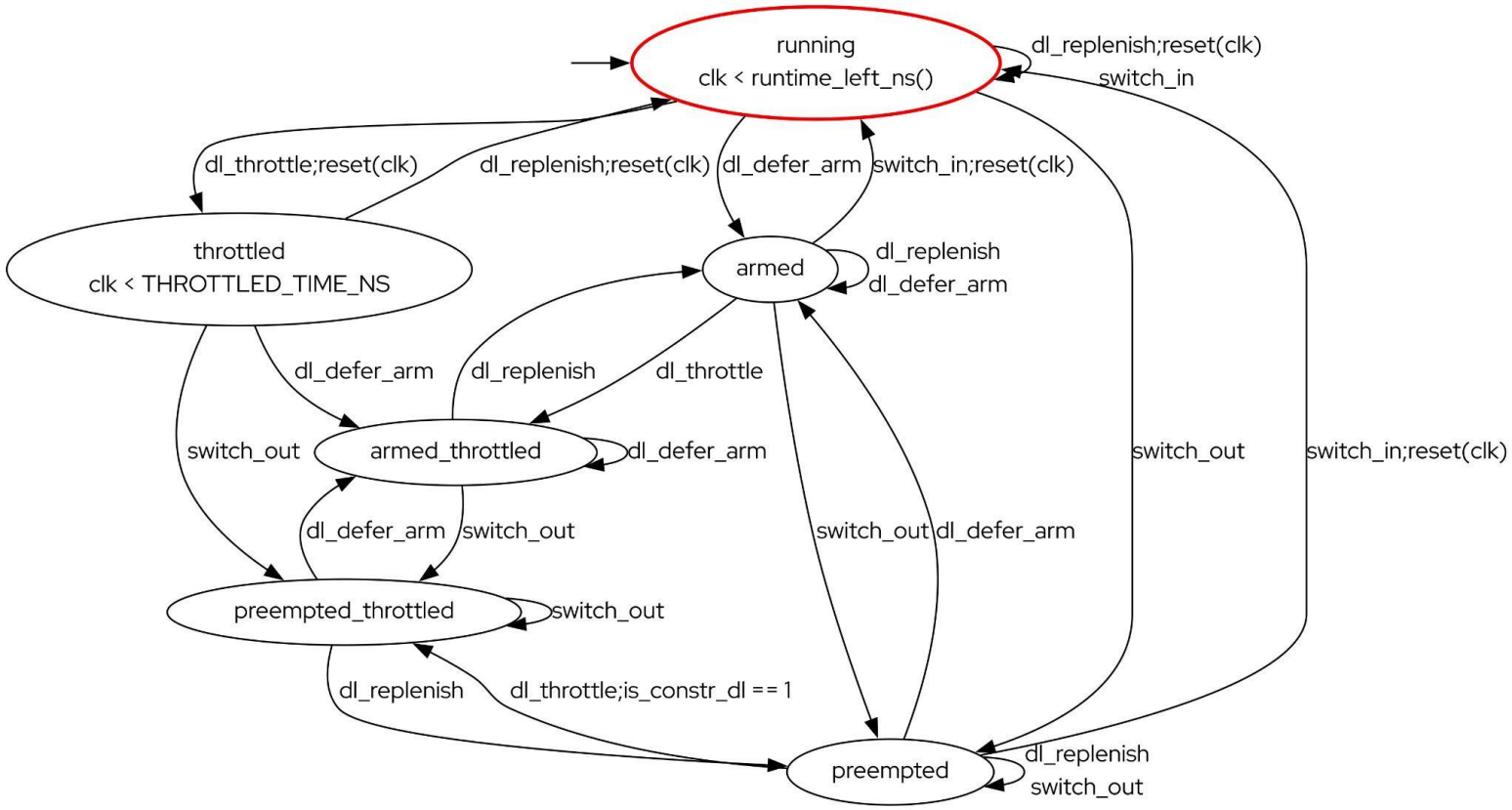
throttle, replenish, server start/stop

Throttle and Nomiss monitors

More complicated than shown before, more states to handle servers

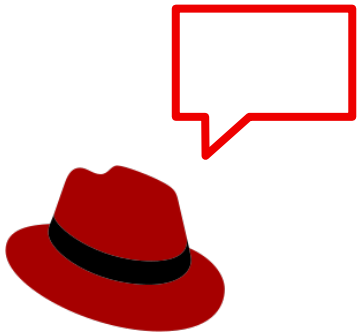
V3 - https://lore.kernel.org/lkml/20251205131621.135513-1-gmonaco@redhat.com/
RFC - https://lore.kernel.org/lkml/20250814150809.140739-1-gmonaco@redhat.com/

Model Adapted to Servers



throttle model

The server's behaviour slightly changed upstream. Is the model still valid?



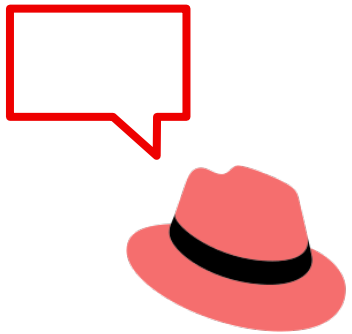
Subsystem
Maintainer

Changes on the Deadline Server

- Models can become obsolete as the code changes, e.g.
  - Actual dl-server stop adds too much timers overhead, make dl-server less aggressive
  - Do not yield even if no runnable tasks, just stop and wait for next wakeup
- Models can then help validate changes

https://lore.kernel.org/all/20250702121158.465086194@infradead.org/T/#u
https://lore.kernel.org/lkml/175878695126.709179.2478529191908747938.tip-bot2@tip-bot2/

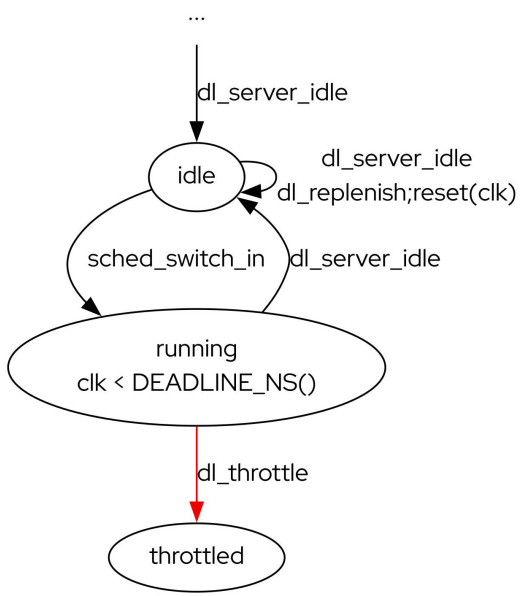
The model spotted the server boosting tasks after its deadline. Is it expected?



RV Expert

Server Boosting after Deadline

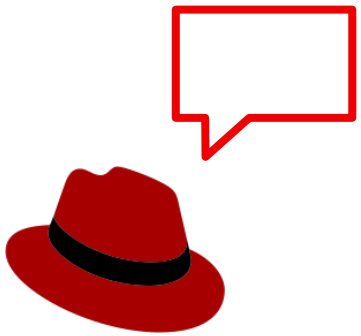
```
     <idle>-0    10.586189: (+949706) event_nomiss:        -15: idle x dl_replenish -> idle
     <idle>-0    10.586194: (+5)     sched_dl_replenish:  comm=server pid=-15 runtime=50000000 deadline=11515716426 yielded=0
     <idle>-0    16.367597: (+5781403) sched_wakeup:        rcuc/15:140 [98] CPU:015
     <idle>-0    16.367674: (+77)    sched_wakeup:        ksoftirqd/15:142 [120] CPU:015
     <idle>-0    16.367730: (+56)    event_nomiss:        -15: idle x sched_switch_in -> running
     <idle>-0    16.367732: (+2)     sched_switch:        swapper/15:0 [120] R ==> ksoftirqd/15:142 [120]
ksoftirqd/15-142 16.367776: (+44)    error_env_nomiss:    -15: event dl_throttle not expected in the state running with clk=4780585437
ksoftirqd/15-142 16.367806: (+30)    sched_switch:        ksoftirqd/15:142 [120] S ==> rcuc/15:140 [98]
```



- Server's deadline = 1s (default)
- *running* means boosting a fair task
- Fair task still running 4.7s after the deadline (when throttle occurs)

[RFC] sched/deadline: Avoid dl_server boosting with expired deadline

The model caught something slightly wrong in the current implementation.



Subsystem
Maintainer

Server Boosting after Deadline

- Report [1]: dl_server stays running when system is idle; after idle period > dl_server period, fair task wakeup gets unexpected boosting with expired (past) deadline despite no actual starvation
- sched/deadline: Fix dl_server time ac

  - Root cause: Time accounting didn't follow s tracking in dl_server_timer() and dl_server_
  - Solution: Reworked idle class time accounti

- sched/deadline: Fix dl_server stop co

  - Root cause: Idle time and fair runtime treat as expected ping
  - Solution: Introduced dl_defer_idle flag to d accounting idle time during current activati till

IT'S WORKING!

Well, the model and RV are working (finding issues).. my code not so much apparently :-P

1 - [RFC] sched/deadline: Avoid dl_server boosting with expired deadline
2 - https://lore.kernel.org/all/20251020141130.GJ3245006@noisy.programming.kicks-ass.net/
3 - https://lore.kernel.org/all/20251101000057.GA2184199@noisy.programming.kicks-ass.net/

Server Boosting after Deadline

- Report [1]: dl_server stays running when system is idle; after idle period > dl_server period, fair task wakeup gets unexpected boosting with expired (past) deadline despite no actual starvation
- sched/deadline: Fix dl_server time accounting [2]

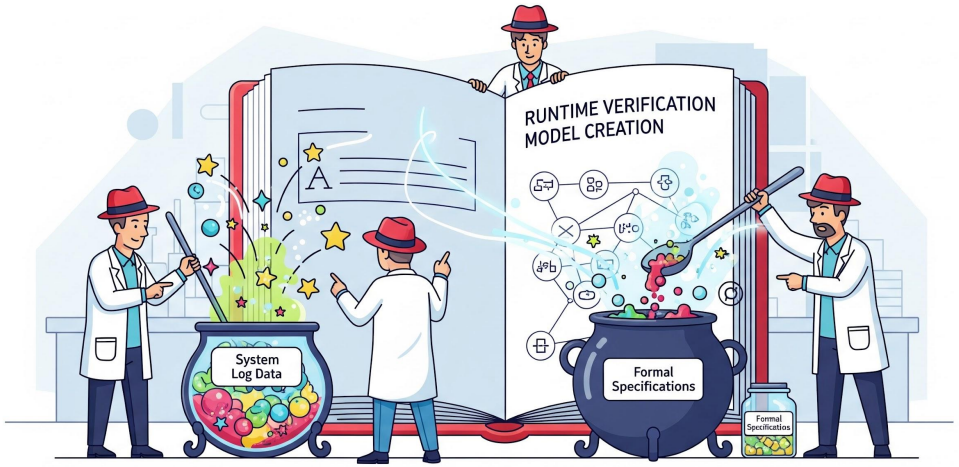
  - Root cause: Time accounting didn't follow scheduler pattern of updating curr before state changes, affecting dl_server runtime tracking in dl_server_timer() and dl_server_start()
  - Solution: Reworked idle class time accounting to match other sched classes and ensure proper propagation

- sched/deadline: Fix dl_server stop condition [3]

  - Root cause: Idle time and fair runtime treated equally—both push activation forward indefinitely, preventing dl_server from stopping as expected
  - Solution: Introduced dl_defer_idle flag to distinguish idle vs. non-idle accounting—only full period of idle stops dl_server while still accounting idle time during current activation

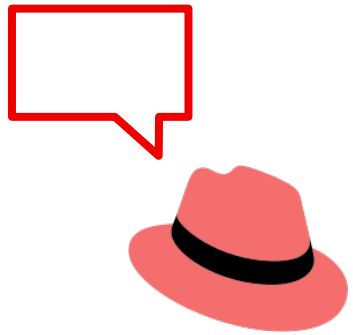
1 - [RFC] sched/deadline: Avoid dl_server boosting with expired deadline
2 - https://lore.kernel.org/all/20251020141130.GJ3245006@noisy.programming.kicks-ass.net/
3 - https://lore.kernel.org/all/20251101000057.GA2184199@noisy.programming.kicks-ass.net/

So, what's the magic recipe for adding new models?



Subsystem Maintainer

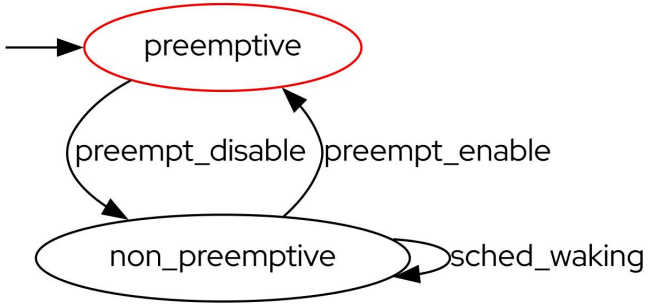
Here are the steps for new models in RV.



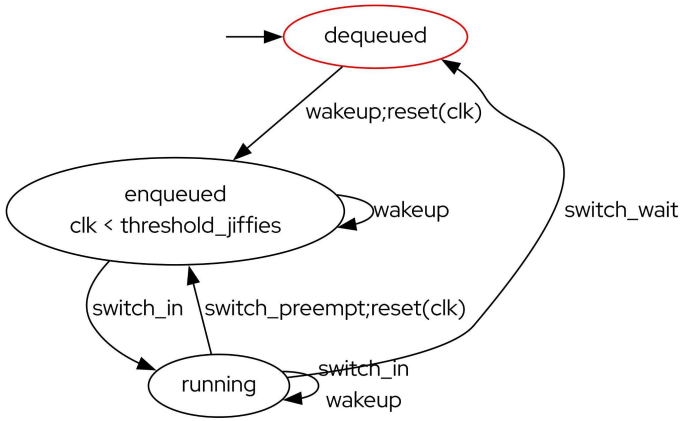
Identify Specifications to Validate

- Rules that must be true for the subsystem
- Should be simple and abstract
- Not implementation dependant
- E.g. deadline tasks cannot run past their deadlines

Choose Model Class



RULE = always (ACQUIRE imply (ALIVE until RELEASE))
ALIVE = not KILLED and not CRASHED



Deterministic Automaton (DA)

Graphical representation (*dot*) of a state machine where next state depends only on current state and event.

Linear Temporal Logic (LTL)

Can be more flexible and composable. Text representation of a rule with defined grammar.

Hybrid/Timed Automaton (HA)

Based on DA. Can represent real time or variables harder to access as constraints on edges or states.

https://docs.kernel.org/trace/rv/deterministic_automata.html
https://docs.kernel.org/trace/rv/linear_temporal_logic.html

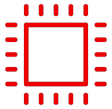
Identify Monitor Type

Number of model instantiations, each with its own state

Global

a single monitor on the system

States = {booting, suspended, running}



Per-CPU

one monitor per CPU

States = {preemption enabled, preemption disabled}

Per-task

one monitor for each thread

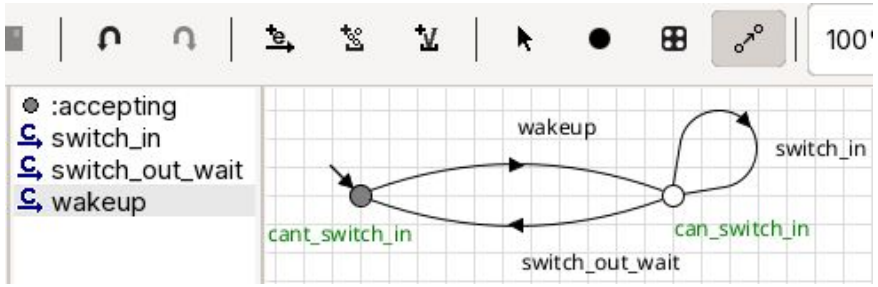
States = {running, ready, suspended}

Need something else?

Per-object monitors

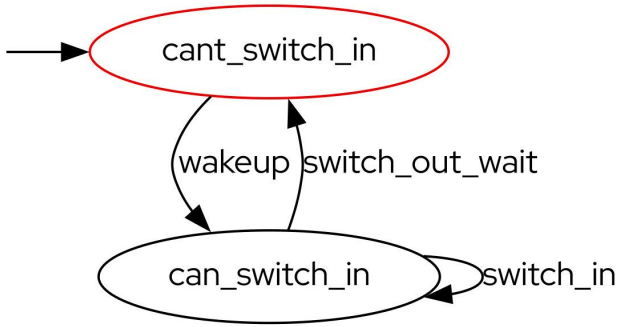
E.g. per-deadline entity

Write Down Model




```
digraph state_automaton {
  {node [style=invis, label=""] "__init_cant_switch_in"};
  {node [shape = ellipse, color = "#e00"] "cant_switch_in"};
  {node [shape = ellipse] "can_switch_in"};
  "__init_cant_switch_in" -> "cant_switch_in";
  "can_switch_in" -> "can_switch_in" [ label = "switch_in" ];
  "can_switch_in" -> "cant_switch_in" [ label = "switch_out_wait" ];
  "cant_switch_in" -> "can_switch_in" [ label = "wakeup" ];
  { rank = min ; "__init_cant_switch_in"; "cant_switch_in"; }
}
```

- Start from existing examples in tools/verification/models [1]
- Use the Supremica IDE [2] for Deterministic Automata (export to *dot*)
- Manually add constraints to Hybrid Automata



1 - https://elixir.bootlin.com/linux/latest/source/tools/verification/models
2 - https://github.com/Chalmers-Control-Automation-Mechatronics/Supremica

Identify Events

Attach to existing or new tracepoints

```
TRACE_EVENT(sched_switch,
    TP_PROTO(bool preempt,
             struct task_struct *prev,
             struct task_struct *next,
             unsigned int prev_state),
```

```
DECLARE_EVENT(sched_dl_replenish,
     TP_PROTO(struct sched_dl_entity *dl_se,
              int cpu),
```

- Reuse tracepoints if possible
  - switch in/out is *sched_switch*
- Filter with arguments when needed
  - Use *prev_state* to catch *switch_out_wait*
- Create new tracepoints otherwise
  - Get necessary arguments

Run Model Through rvgen

```
$ python tools/verification/rvgen monitor -t per_task -c ha -s nomiss.dot

Opening and parsing the specification file /home/gmonaco/nomiss.dot
Writing the monitor into the directory nomiss
Almost done, checklist
  - Edit the nomiss/nomiss.c to add the instrumentation
  - Edit kernel/trace/rv/rv_trace.h:
Add this line where other tracepoints are included and HA_MON_EVENTS_ID is defined:
#include <monitors/nomiss/nomiss_trace.h>

  - Edit kernel/trace/rv/Makefile:
Add this line where other monitors are included:
obj-$(CONFIG_RV_MON_NOMISS) += monitors/nomiss/nomiss.o

  - Edit kernel/trace/rv/Kconfig:
Add this line where other monitors are included:
source "kernel/trace/rv/monitors/nomiss/Kconfig"

  - Move nomiss/ to the kernel's monitor directory (kernel/trace/rv/monitors)
```

- Or use the *-a* option to automate most of the steps

https://docs.kernel.org/trace/rv/monitor_synthesis.html

Complete Generated Code

```
static void handle_switch_in(void *data, /* XXX: fill header */)
{
    struct task_struct *p = /* XXX: how do I get p? */;
    da_handle_event(p, switch_in);
}
static void handle_switch_out_wait(void *data, /* XXX: fill header */)
{
    struct task_struct *p = /* XXX: how do I get p? */;
    da_handle_start_event(p, switch_out_wait);
}
...
rv_attach_trace_probe("model", /* XXX: tracepoint */, handle_switch_out_wait);
rv_attach_trace_probe("model", /* XXX: tracepoint */, handle_switch_in);
```

```
static void handle_switch(void *data, bool preempt,
                          struct task_struct *prev,
                          struct task_struct *next,
                          unsigned int prev_state)
{
    if (prev_state != TASK_RUNNING && !preempt)
        da_handle_start_event(prev, switch_out_wait);
    da_handle_event(next, switch_in);
}
...
rv_attach_trace_probe("model", sched_switch, handle_switch);
```

```
static u64 ha_get_env(enum envs env)
{
      if (env == preempt_count)
            return /* XXX: how do I read preempt_count? */
      return ENV_INVALID_VALUE;
}
```

```
static u64 ha_get_env(enum envs env)
{
      if (env == preempt_count)
            return preempt_count() - 1;
      return ENV_INVALID_VALUE;
}
```


https://docs.kernel.org/trace/rv/da_monitor_instrumentation.html

Test the Monitor

Build and boot the new kernel

Reactors (use RV)

```
# rv mon nomiss -r printk
```

Enable only (use tracepoints with other tools)

```
# echo 1 > /sys/kernel/tracing/rv/monitors/nomiss/enable
```

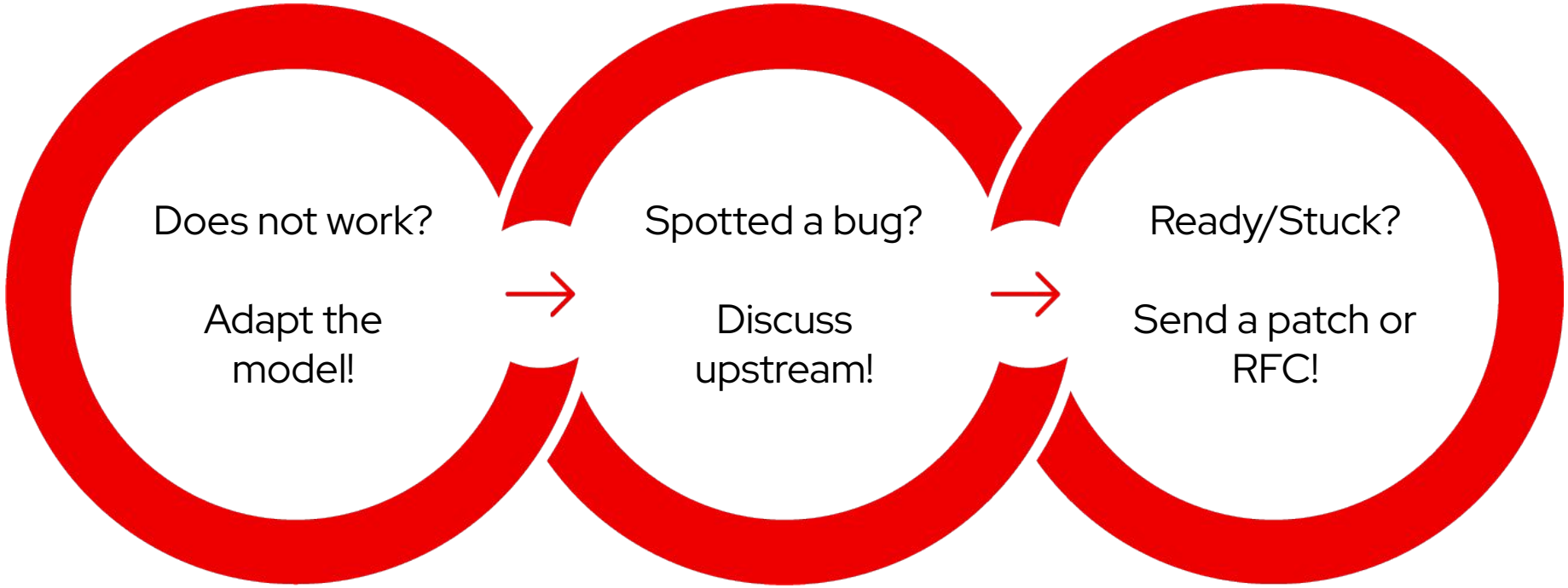
Tracefs (trace-cmd)

```
# trace-cmd record -e sched_switch -e sched_waking -e rv:*_nomiss ./workload
```

Perf (count violations)

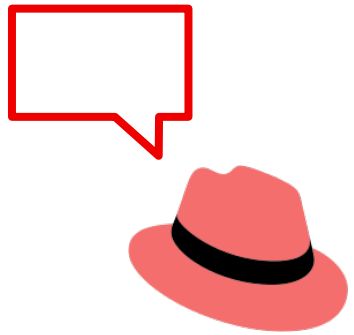
```
# perf stat -e rv:error_nomiss -e rv:error_env_nomiss ./workload
```


Iterate

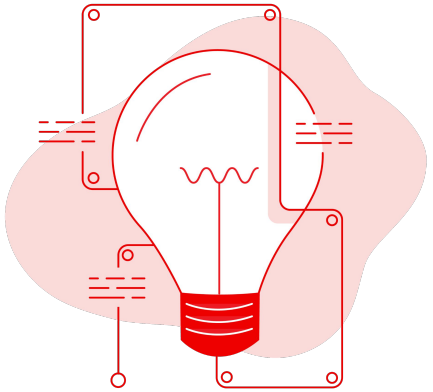

Does not work?
Adapt the model!
Spotted a bug?
Discuss upstream!
Ready/Stuck?
Send a patch or RFC!

What's next?

Discussed this morning at the Sched/RT MC



RV Expert



Support multiple deadline servers and more models

Currently expect one server (fair) per CPU, changing upstream (SCX server).

Validate other schedulers / subsystems

Verify functional / timing rules without changing the main code.

New model classes

Timed automata may not be the right tools (e.g. no concurrency).

RV and real-time properties, Sched/RT MC
https://lpc.events/event/19/contributions/2218/

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

 linkedin.com/company/red-hat

 facebook.com/redhatinc

 youtube.com/user/RedHatVideos

 twitter.com/RedHat