

## RV: where are we?

Daniel Bristot de Oliveira Senior Principal Software Engineer



The RV subsystem

#### **RV** Introduction

### **Runtime Verification**

- What is RV?
- Where are we from?
- RV and safety critical systems
- Where are we?
  - Current RV structure in kernel
- What is next

The initial patch set was under submission when I submitted this topic, but it was merged on 6<u>.0</u>!

So we will have a look at the current structure and talk about what is next!



## **Runtime Verification**

- Runtime Verification (RV) is a lightweight (yet rigorous) formal method that with a more practical approach for complex systems.
- Instead of relying on a fine-grained model of a system (e.g., a re-implementation a instruction level), RV works by analyzing the trace of the system's actual execution, comparing it against a formal specification of the system behavior.

This topic was explored during Daniel's PhD, where he used it as the basis for the creation of an preemption model for the PREEMPT\_RT, then used to proof the scheduling latency bound for the kernel-rt



The RV subsystem

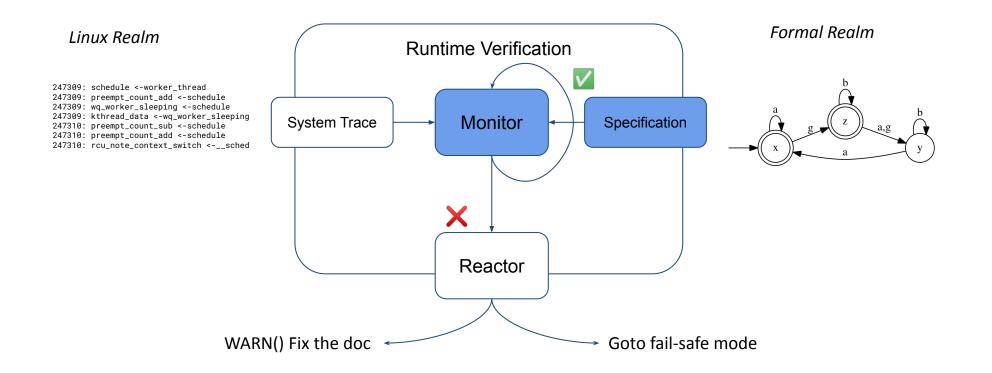
## **Runtime Verification: IOW**

- As the system runs, it generates events to be analyzed
- These events are analyzed against a well-defined description of the system
  - Online Synchronous if it blocks the system
  - Online Asynchronous if it does not block
  - Offline if it verifies the system based on a trace file
- The analysis produces a verdict
- The system can react to an unexpected event
  - $\circ \quad \text{Only makes sense for online RV}$

The current supported method is online synchronous.



## **Runtime Verification on Linux**





The RV subsystem

#### **RV** introduction

## Why formal?

- Because it is precise and unambiguous
  - It is about reasoning, not about code
  - Math is math
  - See this "The Man Who Revolutionized Computer Science With Math\*": https://www.youtube.com/watch?v=rkZzg7Vowao
- It is possible to analyze formal properties of your reasoning
  - Does the logic have contradictions?
  - Is the reasoning deadlock free?
- It is closer to other formal types of demonstration
  - Like the demonstration of the scheduling latency (my talk at LPC 2020).
- It helps to document the code
  - And adds value to the documentation for safety critical systems

We currently support only one formalism, but more methods are are under development.



The RV subsystem

# Where are we from?

#### Where are we from?

#### How it started

- How can I demonstrate the bound for the scheduling latency?
  - In a way that I could convince theoretical researchers
  - But that could also be meaning full for Linux people
- I solved this problem using an automaton model explaining the PREEMPT\_RT synchronization.
  - LPC 2018 talk Mind the gap between real-time Linux and real-time theory
  - Used the model specifications to derive a bound for the scheduling latency
  - LPC 2020 talk: "A theorem for the RT scheduling latency (and a measuring tool too!)

## Automata for complex models

- Automata is a formal method with a large set of operations for composition and validation
- A large automata can be built form a set of small automaton.
- The PREEMPT\_RT thread model has:
  - +9k states
  - +21k transitions
  - Build on a set of automata
    - The vast majority of the automaton modules have 2/3 states
    - The largest has 10 states
- (Scientific) journal paper on the subject ->

	Contents lists available at ScienceDirect	
\$~\$?. }	Journal of Systems Architecture	JSA
ELSEVIER	journal homepage: www.elsevier.com/locate/sysarc	

#### A thread synchronization model for the PREEMPT\_RT Linux kernel

ABSTRACT



Daniel B. de Oliveira<sup>a,b,c,\*</sup>, Rômulo S. de Oliveira<sup>b</sup>, Tommaso Cucinotta<sup>c</sup>

<sup>a</sup> RHEL Platform/Real-time Team, Red Hat, Inc., Pisa, Italy <sup>b</sup> Department of Systems Automation, UFSC, Florianópolis, Brazil <sup>c</sup> RETIS Lab, Scuola Superiore Sant'Anna, Pisa, Italy

#### ARTICLE INFO

This article proposes an automata-based model for describing and validating sequences of kernel events in Linux PREEMPT_RT and how they influence the timeline of threads' execution, comprising preemption control, inter- rupt handling and control, scheduling and locking. This article also presents an extension of the Linux tracing framework that enables the tracing of kernel events to verify the consistency of the kernel execution compared to the event sequences that are legal according to the formal model. This enables cross-checking of a kernel behavior against the formalized one, and in case of inconsistency, it pinpoints possible areas of improvement of the kernel, useful for regression testing. Indeed, we describe in details three problems in the kernel revealed by using the proposed technique, along with a short summary on how we reported and proposed fixes to the Linux kernel community. As an example of the usage of the model, the analysis of the events involved in the activation of the highest priority thread is presented, describing the delays occurred in this operation in the same granularity used by kernel developers. This illustrates how it is possible to take advantage of the model for analyzing the preemption model of Linux.



#### Where are we from?

### Automata for RV

- Automata is a well define formalism
- Automata is a method to model Discrete Event Systems (DES)
  - Formally, an automaton is defined as:
    - $G = \{X, E, f, x_0, X_m\}$ , where:
      - X = finite set of states;
      - E = finite set of events;
      - f = transition function =  $(X \times E) \rightarrow X$ ;
      - $x_0 =$ Initial state;
      - $X_m$  = set of final states.
  - $\circ$  The language or traces generated/recognized by G is the L(G).

There are multiple different automata definitions, in this case we are talking about Deterministic Automata (DA)

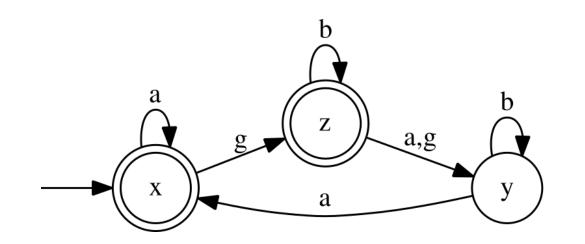
Kernel doc explains it!



The RV subsystem

## Automata for RV

• The good thing about automata is that it also has an graphical representation:



A state-machine is a sort of automata! It is one of the basis of CS.



The RV subsystem

#### Where are we from?

### Using automata to prove things

- It worked! By explaining the logical behavior, we derived a timing bound!
- While developing and testing the model ended up finding kernel bugs
  - So people asked my, why not make it generic?

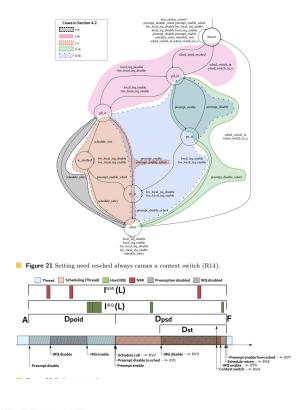
## Demystifying the Real-Time Linux Scheduling Latency

Daniel Bristot de Oliveira () Red Hat, Inc, Italy bristot@redhat.com

Daniel Casini Scuola Superiore Sant'Anna, Italy daniel.casini@santannapisa.it

Rômulo Silva de Oliveira Universidade Federal de Santa Catarina, Brazil romulo.deoliveira@ufsc.br

Tommaso Cucinotta Scuola Superiore Sant'Anna, Italy tommaso.cucinotta@santannapisa.it



#### $L^{IF} \le max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD}, \tag{6}$

**Proof.** The lemma follows by noting that cases (i-a), (i-b), (i-c), (ii-a), (ii-b) are mutuallyexclusive and cover all the possible sequences of events from the occurrence of  $\mathtt{RHP}_i$  and  $\mathtt{set\_need\_resched}$ , to the time instant in which  $\tau_i^{\mathtt{THD}}$  is allowed to execute (as required by Definition 1), and the right-hand side of Equation 6 simultaneously upper bounds the right-hand sides of Equations 2, 3, 4, and 5.

Theorem 8 summarizes the results derived in this section.

Lemma 7.

**Theorem 8.** The scheduling latency experienced by an arbitrary thread  $\tau_i^{\text{THD}}$  is bounded by the least positive value that fulfills the following recursive equation:

 $L = max(D_{ST}, D_{POID}) + D_{PAIE} + D_{PSD} + I^{NMI}(L) + I^{IRQ}(L)$   $\tag{7}$ 

Proof. The theorem follows directly from Lemmas 7 and Equation 1.



## Efficient automata verification

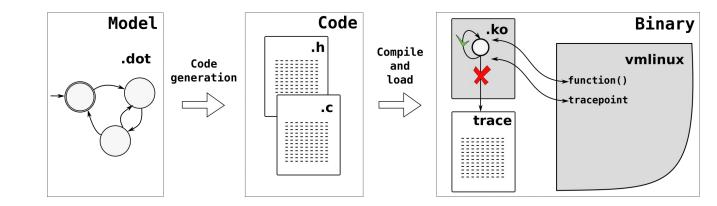
- The PREEMPT\_RT model was a super-high-frequency one
  - Multiple events per microsecond
- Doing it in user-space was not efficient due to trace buffer overhead
- So we developed a way to transform the automata into code and run it in the kernel
  - O(1) operating
  - It was faster than just tracing
    - Because it is a simpler operation
- And we ended up finding more bugs



Daniel Bristot de Oliveira<sup>1,2,3</sup>[0000-0002-4577-7855], Tommaso Cucinotta<sup>2</sup>[0000-0002-0362-0657], and Rômulo Silva de Oliveira<sup>3</sup>[0000-0002-8853-9021]

<sup>1</sup> RHEL Platform/Real-time Team, Red Hat, Inc., Pisa, Italy.
 <sup>2</sup> RETIS Lab, Scuola Superiore Sant'Anna, Pisa, Italy.
 <sup>3</sup> Department of Systems Automation, UFSC, Florianópolis, Brazil.

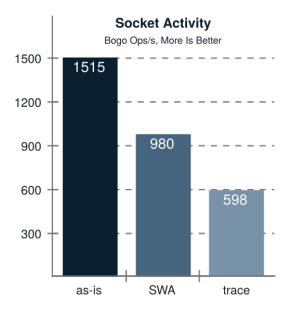
**Abstract.** Formal verification of the Linux kernel has been receiving increasing attention in recent years, with the development of many models, from memory subsystems to the synchronization primitives of the real-time kernel. The effort in developing formal verification methods

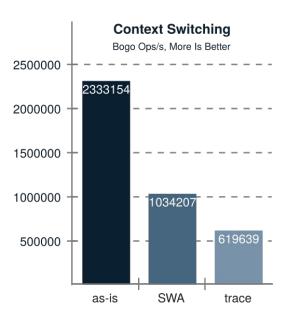


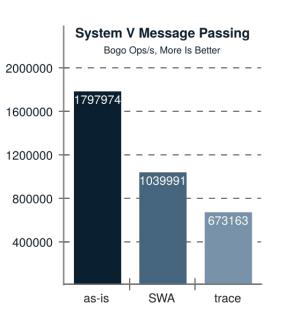


## End efficient

• Running RV code in kernel is faster than tracing it!









## RV meets safety-critical systems

15

The RV subsystem

## RV and safety-critical system

- RV uses formal-method
  - Which is at the top of the list of approaches to use for certification
- Three birds with a single stone!
  - Documents the system using automata
  - It verifies the system at the development/testing phases
  - It monitors the system at runtime
    - Reacting to unexpected events
- It only uses well-established technologies in the safety-critical fields:
  - Pure and straightforward C code
  - Statically allocated memory
  - $\circ \quad \text{No loops and so on} \quad$

Talk at ELCE 2019: Formal Verification Made Easy (and fast!)

https://www.youtube.com/watch?v=BfTuEHafNgg

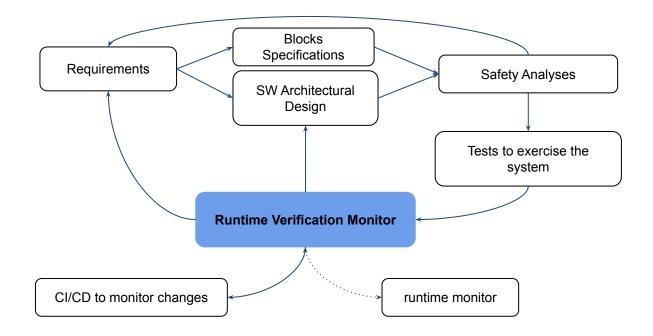




#### RV in safety-critical systems

## RV and safety-critical system

- I started working with people in the Elisa Project
- Gabriele Paoloni and I are working on the approach presented at LPC 2021





The RV subsystem

## RV and safety-critical system

- RV can be used to document the kernel
- But also to document how the system should behave:
  - How the kernel is suppose to behave
  - How the user is supposed to behave
- It can be used to split subsystems into small parts
  - Each small part can be qualified as a black box (ISO 26262 part 12)
  - RV monitors the interface (ISO 26262 part 6)
- Further information:
  - See talk ->
  - Together with Gabriele Paoloni

A Maintainable, Scalable, and Verifiable SW Qualification Approach for Automotive in Linux

https://www.youtube.com/watch?v=6\_gvarChkAg





## Where are we?

📥 Red Hat

19

The RV subsystem

Where are we?

## Merged!

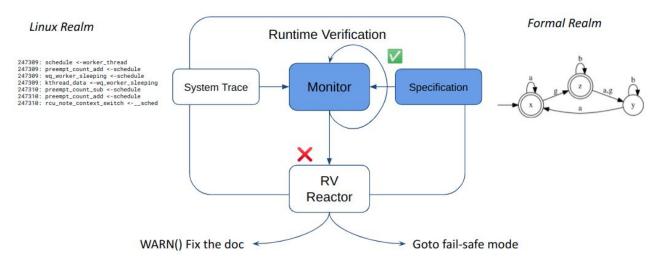
The RV subsystem



#### **RV Interface**

#### Interface and concepts

- RV subsystem is an interface and a set of tools to make RV accessible
- A RV **monitor** composed of:
  - A model/specification
  - The instrumentation
- A **reactor** is an action available to the monitor
  - Can be invoked by the monitor if an unexpected even happens
- Monitors can be enabled at runtime
  - Each monitor can have one enabled **reactor**





**RV Interface** 

## Interface demo

201	1: ( 00) (	4 1
prep-1272	10021-112-1	- 223, 727525; event, etc: con_presentive x_present_enable ++ presentive (final)
10740-1411	6621	-227.7276261 event_sip; presentive a present_disable -1 neo_presentive
ETHE-1215	10021 1112.	227.727556: event wip: non preceptive a precept stuble to preceptive (final)
BURG-ALTE	HER LAR	221.7276611 event_wip: presentive a procept_disable res.preemptive
10101	11000 1	117 121502: event wig: cor, presenting a present anable is presenting (final)
10/01-1278	(1642)1 1.2	- 201.7215827 comm_elg: presentive a present_disable -> nam_presention
grap-1373	10023 1.2	327.7278711 event wint new presentive a present mable >> praneptive offenil
grap-111a	8001	117.137572; event_wip; preesptive a prevept_disable out_prvenptive
STATE LITTLE	10001-1222	107.7279751: event, wip: non-promptive s. prompt, anable -> promptive (final)
grop-11Ti	19621	131.131872: event_sip: preemptive a preempt_disable +> con_presentive
CONTRACTOR -	sterri usa	227-7279914 court, sign mer, presentive a present, enable -> presentive (final).
grep-127k	10011	127.727502: event_wip: presentive a present_diable of var_presentive
0140-3278	1000	221.7273921 event_wip: non_preseptive a presspi_enable -> preseption (final)
BORD ALTS	10000	223,727611) event star presentive a present disable -) non presentive
group-late	themp	122. fittill eventually new presentive a pressur mobile -> presentive (final)
grap-1275	10001	317.727614; event elp; presentive a present staable con preceptive
gree-Jata	10001	207.7216281 event_elpt wer_preseptive a presept_enable -> preseptive (final)
group-1213	1001	227,727635; event wipt presentive a present disable -> non preemptive
BURD-JUTT	1001	221.1210351 event_wip: compreenptive a pricipit_ecoble -> preceptive. (final)
1000-000	10023 41	227,7270381 event, wip: presentine a present disable -> per, presention-
10700-1218	10021 11.3	237.7376.00 event_wip: con_presentive a present_mubile -> presentive (final)
61991-1272	10401	227.727038; event_wigt presentive a present_disable to con_primeptive
grep-1217	10001 12.2	227.7976381 event_wipt run_presentive x present_enable -> presentive (fimil)
BOOM TATE	10021	227-7278381 event sipt presentive a present disable -> respresentive
graph (272)	10001 12.10	1223.727628: event_eip: not_presentive a present_mable presentive (final)
0140-1278	INCOME.	127-7276721 doerd with presentive a present studie of one percention
grap-1177.	101011-003-0	227.727628) event_wip: con_presentive x present_anable presentive (final)
0100-1118	estimat-	227.7276281 event_wip: preesptive a presspt_disable -> non pressptive
1111-0111	10401 12.2.1	127.727632: event wip: non presentive a present wrable or presentive (freed)
group-same	THEO HARD	121.727634; event_wip; presentive a present_disable ran, presentive
menter 12/11	10000)	117.717034: event wigh non properties a present cruble 11 presentity (final)
TENED-10TH	((eta)	jar. 7276361 contr_stp: presentive a presept_disable in nor_preseption
geogr-1277	10021 112	327.727635; event wip: ner presentive a present enable -> presentive (final)
grep-1118	0001	133.7318351 event_wip: preceptive a precept_disable -> sut_preceptive
PAP-DITE	100001-12001	337,72703bi event wight new presenting a present stable to presentive (frield
grep-Little	(Beck) Millia	137.131638: event_stp: premptive a precept_disable +> one_preceptive
#P## 1273	10001	227,7270381 event, sign ran presention of present, studie -+ presention (first)
prep-1273	THEFT HELD	127.727638: event_wip: presentive = present_Winable or sar_presentive
trapingly of the		

Demo video

https://youtu.be/V42BFNKbh0g





#### **Monitor synthesis**

#### Monitor synthesis

- The RV enables conversion of an automata.dot -> kernel RV monitor
  - This is done with a dot2k utility
- **dot2k** creates a skeleton of a RV monitor module
- The monitor also uses a set of C macros to generate the monitor
   code for the specific type of monitor
- The only thing left for the humans to do is the **instrumentation** 
  - The connection between a kernel event and the model event

Using code generation automatizes the process, making it less prone to error and easier to qualify.



## Monitor synthesis demo

<pre>// #02-1/inter-interifier()#2-12 http://www.sec.en/ // #02-1/inter-interifier() // #02-1/interifier() // #02-1/interifier() //</pre>		
<pre>introduct = transformation in the introduct = transformation in the i</pre>		
<pre>introduct = transformation in the introduct = transformation in the i</pre>		
<pre>int is the state is the st</pre>	Include - Union (Income In Include - Union (Income Income Inc Include - Union (Income Income Inc Include - Union (Include Income Income Include - Union (Include Income Include - Union (Include Include - Union (Include Include - Union (Include - Union (Incl	
<pre>interest interest interes</pre>	native MODILE_DAME "wrp"	
<pre>trick provider ( c colp) ColeCol_Col_Col_Col_Col_Col_Col_Col_Col_Col_</pre>	Instante - Et acosterent schollosis to	
<pre>xxxxd Dx_Dxxx pres_creating, watgreat dwg;;  xxxxd Dx_Dxxx pres_creating, control dwg;;  xx_xxxd Dx_Dxxx pres_creating, control dwg;;  xx_xxxd Dx_Dxxx pres_creating, control dwg;;  xx_xxx dx_Dx_Dxxx pres_creating, control dwg;;  xx_xx dx_Dx_Dxxx pres_creating;;  xx_xx pres_</pre>	iv-less Data.10	
<pre>su_tandle_scort_etprocesst_clashis_stat; ter(c and formite_present_etplication does, and good long_ (p, and good long_ person); (p) su_tandle_scort_eter(, etplication does, and (process ter()) support_eter(, etplication does, and (process ter()) support_eter(); (p) ter(c does not be etplication support_eter(); ter(c etclication); support_eter(); ter(c etclication); support_eter(); ter(c etclication); support_eter(); ter(c etclication); support_eter(); ter(c etclication); support_eter(); ter(c etclication); support_eter(); ter(c etclication); support_eter(); ter(c etclication); support_eter(); support</pre>	Rhack <u>Sw. Houristor</u> v. wips. CCLASE_LON_HOM_FER_CFNLatp, wowingsmal chary;	
<pre>tects core formite_present_evolutioned_core</pre>	tatis wid herefugirments intoble (wid state) and yet hing tos me tored long participation	
anglanafia provinteration angli anakar angli anakar angli anakar angli angli angli angli angli angli angli ang mangang angli ang angli ang angli ang angli angli angli angli ang ang angli ang	ca_tardie_even_wiproreempt_plaadie_wihit;	
anglanafia provinteranter anglanafia analy anglananat analy mang angla provinteranter (anglanafia anglana) mang angla provinteranter (anglana) mang angla provinteranter (anglana) mang angla provinteranter (anglana) mang angla	terit over heads, present and plants roke, and pool long in, and pool long parent, by	
nerie ann fentle schel en ingforde mine, einer mek senet enze) m fundle som erstereter anning ettil mine minele scheler skenet	an hardle, sourt, eyert, eigige eegh, siddin, eigit.	
en yard's seen with star and a sign at a s		
natic ter mekla siyisətd) im (mixil)		
im muili		
	tatic int models_sip(iot6)	
12.1 Tes 1		
		11.1 Tep

Demo video

https://youtu.be/3yDxz1Sl4k8





## **RV Structure**

25

The RV subsystem

#### **RV** Interface

- kernel/tracing/rv/
  - rv.c: rv interface
    - Startup
    - Register & control monitors
  - rv\_reactor.c:
    - Startup
    - Register & control reactors
  - Reactors

[bristot@x1 linux]\$ ls -l kernel/trace/rv/ total 56 -rw-r--r-- 1 bristot bristot 2233 Aug 24 08:38 Kconfig -rw-r--r-- 1 bristot bristot 299 Aug 24 08:38 Makefile drwxr-xr-x 4 bristot bristot 4096 Aug 24 08:38 monitors -rw-r--r-- 1 bristot bristot 972 Aug 24 08:38 reactor\_panic.c -rw-r--r-- 1 bristot bristot 989 Aug 24 08:38 reactor\_printk.c -rw-r--r-- 1 bristot bristot 19406 Aug 24 08:38 rv.c -rw-r--r-- 1 bristot bristot 1524 Aug 24 08:38 rv.h -rw-r--r-- 1 bristot bristot 11129 Aug 24 08:38 rv\_reactors.c [bristot@x1 linux]\$



## **RV Monitors**

- kernel/tracing/rv/monitors
  - Each monitor has its own directory
  - They are independent from the interface
    - The interface only controls the monitoring session
  - For deterministic automata monitors, there are two files:
    - .h the automata
    - .c instrumentation and control
  - Other types of monitors might have different files
    - But it is important to keep the specification separated from the code

```
[bristot@x1 linux]$ ls -l kernel/trace/rv/monitors/
total 8
drwxr-xr-x 2 bristot bristot 4096 Aug 24 08:38 wip
drwxr-xr-x 2 bristot bristot 4096 Aug 24 08:38 wwnr
[bristot@x1 linux]$ ls -l kernel/trace/rv/monitors/wip/
total 8
-rw-r--r-- 1 bristot bristot 2066 Aug 24 08:38 wip.c
-rw-r--r-- 1 bristot bristot 1023 Aug 24 08:38 wip.h
[bristot@x1 linux]$
```



### **RV Headers**

- include/linux
  - rv.h: monitor interface
- include/rv
  - Instrumentation.h: helper functions for instrumentation
  - automata.h: helpers for automata operation
  - da\_monitor.h: helpers for deterministic automata monitor
  - Other types of monitor (timed automata) will add files here

[bristot@x1 linux]\$ ls -l include/rv total 28 -rw-r--r-- 1 bristot bristot 2559 Aug 24 08:38 automata.h -rw-r--r-- 1 bristot bristot 17416 Aug 24 08:38 da\_monitor.h -rw-r--r-- 1 bristot bristot 885 Aug 24 08:38 instrumentation.h [bristot@x1 linux]\$ ls -l include/linux/rv.h -rw-r--r-- 1 bristot bristot 1595 Aug 24 08:38 include/linux/rv.h [bristot@x1 linux]\$



## **RV** Tools

- tools/verification:
  - dot2/:
    - dot2 tools: tools to create the skeleton of the monitor
    - dot2k\_templates/: templates for the monitor
  - models/
    - A place to store models/specification
    - Sample monitors that can be used as starting point

Consequence de la de la consequencia de la conseque

[bristot@x1 linux]\$ ls -lR tools/verification/ | grep -v total tools/verification/: drwxr-xr-x 3 bristot bristot 4096 Aug 24 08:38 dot2 drwxr-xr-x 2 bristot bristot 4096 Aug 24 08:38 models

#### tools/verification/dot2:

-rw-r--r-- 1 bristot bristot 5879 Aug 24 08:38 automata.py -rw-r--r-- 1 bristot bristot 942 Aug 24 08:38 dot2c -rw-r--r-- 1 bristot bristot 7959 Aug 24 08:38 dot2c.py -rw-r--r-- 1 bristot bristot 1746 Aug 24 08:38 dot2k -rw-r--r-- 1 bristot bristot 5942 Aug 24 08:38 dot2k.py drwxr-xr-x 2 bristot bristot 4096 Aug 24 08:38 dot2k\_templates -rw-r--r-- 1 bristot bristot 667 Aug 24 08:38 Makefile

#### tools/verification/dot2/dot2k\_templates:

-rw-r--r-- 1 bristot bristot 1851 Aug 24 08:38 main\_global.c -rw-r--r-- 1 bristot bristot 1858 Aug 24 08:38 main\_per\_cpu.c -rw-r--r-- 1 bristot bristot 1859 Aug 24 08:38 main\_per\_task.c

tools/verification/models:

-rw-r--r-- 1 bristot bristot 602 Aug 24 08:38 wip.dot -rw-r--r-- 1 bristot bristot 573 Aug 24 08:38 wwnr.dot [bristot@x1 linux]\$ []



## **RV** Documentation

#### Documentation/trace/rv/:

- runtime-verification.rst
- da\_monitor\_instrumentation.rst
- da\_monitor\_synthesis.rst
- deterministic\_automata.rst
- For each monitor:
  - monitor\_wip.rst
  - monitor\_wwnr.rst

The RV subsystem

#### Runtime Verification Runtime Monitors and Reactors Online RV monitors The user interface Deterministic Automata Deterministic Automata Monitor Synthesis Deterministic Automata Instrumentation Monitor wip Monitor wwn Kernel Maintainer Handbook fault-injection Kernel Livepatching auide Core API Documentation locking Accounting Block cdrom Linux CPUFreg - CPU frequency and voltage scaling code in the Linux(TM) Frame Buffe fpga Human Interface Devices (HID) I2C/SMBus Subsystem Industrial I/O InfiniBand NetLabel Networking **Power Management** TCM Virtual Device Serial Peripheral Interface (SPI) 1-Wire Subsystem Linux Watchdog Support Linux Virtualization Support Linux Hardware Monitoring Linux GPU Driver Developer's Guide Security Documentation Linux Sound Subsystem Documentation Linux Kernel Crypto API Filesystems in the Linux kernel Linux Memory Management Documentation **BPF** Documentation USB support Linux PCI Bus Subsystem Linux SCSI Subsystem Assorted Miscellaneous Devices

#### → C 🔒 kernel.org/doc/html/latest/trace/rv/runtime-verification.html

#### \* » Linux Tracing Technologies » Runtime Verification » Runtime Verification

#### **Runtime Verification**

Runtime Verification (RV) is a lightweight (yet rigorous) method that complements classical exhaustive verification techniques (such as model checking and theorem proving) with a more practical approach for complex systems.

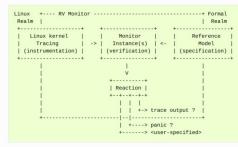
Instead of relying on a fine-grained model of a system (e.g., a re-implementation a instruction level), RV works by analyzing the trace of the system's actual execution, comparing it against a formal specification of the system behavior.

The main advantage is that RV can give precise information on the runtime behavior of the monitored system, without the pitfalls of developing models that require a reimplementation of the entire system in a modeling language. Moreover, given an efficient monitoring method, it is possible execute an *online* verification of a system, enabling the *reaction* for unexpected events, avoiding, for example, the propagation of a failure on safety-critical systems.

#### **Runtime Monitors and Reactors**

A monitor is the central part of the runtime verification of a system. The monitor stands in between the formal specification of the desired (or undesired) behavior, and the trace of the actual system.

In Linux terms, the runtime verification monitors are encapsulated inside the RV monitor abstraction. A RV monitor includes a reference model of the system, a set of instances of the monitor (per-cpu monitor, per-task monitor, and so on), and the helper functions that glue the monitor to the system via trace, as depicted bellow:



In addition to the verification and monitoring of the system, a monitor can react to an unexpected event. The forms of reaction can vary from logging the event occurrence to the enforcement of the correct behavior to the extreme action of taking a system down to avoid the propagation of a failure.

In Linux terms, a reactor is an reaction method available for *RV monitors*. By default, all monitors should provide a trace output of their actions, which is already a reaction. In addition, other reactions will be available so the user can enable them as needed.

For further information about the principles of runtime verification and RV applied to Linux:

Bartocci, Ezio, et al. Introduction to runtime verification. In: Lectures on Runtime Verification. Springer, Cham, 2018. p. 1-33.

Falcone, Ylies, et al. A taxonomy for classifying runtime verification tools. In: International Conference on Runtime Verification. Springer, Cham, 2018. p. 241-262.

De Oliveira, Daniel Bristot. Automata-based formal analysis and verification of the real-time Linux kernel. Ph.D. Thesis, 2020.

#### **Online RV monitors**

Monitors can be classified as offline and online monitors. Offline monitor process the traces generated by a system after the events, generally by reading the trace execution from a permanent storage system. Online monitors process the trace during the execution of the system. Online monitors are said to be synchronous if the processing of an event is attached to the system execution, blocking the system during the event monitoring. On the other hand, an *asynchronous* monitor has a set of advantages. For example, offline monitors can be executed on different machines but require operations to save the log to a file. In contrast, synchronous online method can react at the exact moment a violation occurs.

Another important aspect regarding monitors is the overhead associated with the event analysis. If the system generates events at a frequency higher than the monitor's ability to process them in the same system, only the *offline* methods are viable. On the other hand, if the tracing of the events incurs on higher overhead than the simple handling of an event by a monitor, then a *synchronous online* monitors will incur on lower overhead.

Indeed, the research presented in:

View page source

## RV: what is next?

31

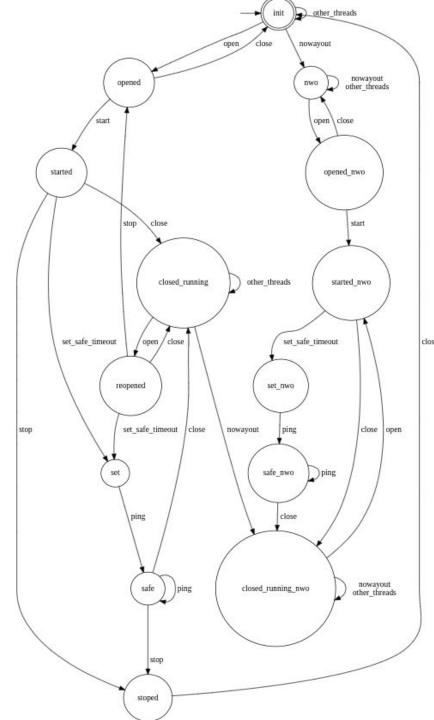
The RV subsystem



#### Watchdog monitor

## Watchdog monitor

- It was part of the first patchset (working with Elisa people)
- The idea is to monitor the watchdog usage until reaching a safe state
  - Open -> start -> set a timeout -> ping at least once
  - Avoid some features to reduce the amount of code to be inspected without changing the code
    - Like the hrtimers usage on watchdog dev
- The monitor automaton is generic
- In the patchset I added some options requested by the safety analysis made in the Elisa group
  - But because I did it all in a single patch, the explanation was not clear
- The patchset also included a user-space tool to exercise the monitor and to serve as starting point for the monitor



#### **Monitor options**

### Monitor options

- We can enrich monitors with additional options/parameters
- For example:
  - int value to be the max safe watchdog timeout
- Each monitor has its folder also to store these specific options
- I am not sure if I will:
  - Add a file per option
    - Easy to use
    - More memory
  - A single option file
    - It will need a syntax like the event's format file
    - Less memory

#### Monitor via module

## Modular monitors

- Now models are built-in only
- But they can be loaded as module I started them as module
- I removed the export symbols in the initial patch set to avoid problems
- I just need to export symbols in include/linux/rv.h
- It has a drawback:
  - Each monitor will have its own DECLARE\_TRACE
  - That is why each monitor has a src dir: to store these monitor specific things

#### #ifdef CONFIG\_RV\_REACTORS struct rv\_reactor { const char \*name; const char \*description; void (\*react)(char \*msg); #endif struct rv\_monitor { const char \*name; \*description; const char enabled; bool (\*enable)(void); int (\*disable)(void); void void (\*reset)(void); #ifdef CONFIG\_RV\_REACTORS (\*react)(char \*msg); void #endif bool rv\_monitoring\_on(void); int rv\_unregister\_monitor(struct rv\_monitor \*monitor); int rv\_register\_monitor(struct rv\_monitor \*monitor); int rv\_get\_task\_monitor\_slot(void); void rv\_put\_task\_monitor\_slot(int slot); #ifdef CONFIG\_RV\_REACTORS bool rv\_reacting\_on(void);

int rv\_unregister\_reactor(struct rv\_reactor \*reactor);
int rv\_register\_reactor(struct rv\_reactor \*reactor);

#endif /\* CONFIG\_RV\_REACTORS \*/

<mark>b</mark> Red Hat

34

The RV subsystem

## Monitors with ebpf

- It is also possible to have monitors in eBPF
- I have a dot2bpf implementation
  - C & libbpf

dot2bpf

- Process the automata in the kernel
- Feedback to user-space
- But it works and will share most of the kernel headers
  - To deduplicate the code of the eBPF and in-kernel option
- There is also a new set of monitors that I am doing with ETH Zurich that will use eBPF for user-space processing



The RV subsystem

## Qualifying the monitors code

- We need to qualify the monitors' code according to the safety regulations
- This is part of the work we do with Elisa
- Red hat is committed to that as well
- The monitor was designed with this in mind
  - Pure C
  - No memory allocation
  - Self-generated
- But it will certainly require some changes that might add overhead
  - So we might have a da\_safe\_monitor.h



#### Other monitors and formalism

### Other monitors

- Integrating the preemption model
  - It will require some extra work in the instrumentation, adjusting existing tracepoints
  - It will happen along with rtsl integration on rtla
    - So we will have the logical and the timing verification of the proven scheduling latency.
- Other monitors for RT
  - Function calls that are not guaranteed to be real-time
  - [Potential] priority inversion scenarios
- I am working with ETH Zurich and the University of Copenhagen on other types of formalism
  - They are more complex than automata, and allow timing in the equations

**RV Where are we?** 

## **Questions?**

The RV subsystem

