Untangling the Intricacies of Thread Synchronization in the PREEMPT_RT Linux Kernel

Daniel B. de Oliveira\textsuperscript{1,2,3}, Rômulo S. de Oliveira\textsuperscript{3}, Tommaso Cucinotta\textsuperscript{2}
bristot@redhat.com, tommaso.cucinotta@santannapisa.it, romulo.deoliveira@ufsc.br,
Agenda

- Introduction
- Related Work
- Background on automata theory
- Proposed approach
- Application of the model
- Conclusions and future work
Linux as a RTOS

- Linux has been used as RTOS on many academic and industrial projects.
  - It has become a fundamental block of real-time distributed systems, e.g.:
    - Sensor Networks
    - Robotics
    - Factory automation
    - Military Drones
    - Distributed and service oriented multimedia systems
    - Distributed high frequency trading systems
Determinism on Linux

- The PREEMPT RT changes a set of in-kernel operations that enhance the deterministic operation of Linux.
- Operations, however, are not atomic.
  - Incurring in non-negligible delays;
  - Even for tasks that are not related.
- The understanding of these rules and how they affect the timing behavior of Linux are fundamental for the development of real-time applications and algorithms.
The in-kernel synchronization mechanisms are complex
- They involve various task contexts (Threads, IRQs, NMI)
- Low-level hardware details
- Kernel hacks
- Not a single place in the code to understand

It may take years to understand them all
- That is why many projects ignore them
  - But they end up not landing on Linux

How can we explain Linux synchronization?
- And what are the benefits of it?
Tracing and DES

- Linux developers use tracing features to analyse the system:
  - They see tracing events that cause states change of the system.
- Discrete Event Systems (DES) methods also use these concepts:
  - events, trace and states...
- DES can be used in the formalization of system.
- So, why not try to describe Linux using a DES method?
Paper contributions

- Proposes an automata-based model for describing and verifying the behavior of thread management code in the Linux kernel:
  - Considers the FULLY_PREEMPTIVE mode
  - Includes
    - IRQ/NMI (and its management)
    - Locking: Mutex, rw locks and semaphores
    - Scheduling

- Presents the extension of the Linux trace features that enables the trace of the events used by the automata in a real scenario.
- Presents how the model can be used to understand Linux
- Presents how the model helps catching bugs in Linux
Agenda

● Introduction
● Related Work
● Background on automata theory
● Proposed approach
● Application of the model
● Conclusions and future work
Related work: Automata

Automata and discrete-event systems have been extensively used to verify timing properties of real-time systems:

- Usage of timed automata for schedulability tests
  - Daws and Yovine - 1995
  - Cimatti, Palopoli, Ramadian - 2008
  - Wang, Li, Wonham - 2016

- To reduce the complexity of the system by using compositions of automata;
  - Lampka, Perathoner, and Thiele - 2013

- Schedulability analysis and code generation
  - Amnell, Fersman, Mokrushin, Pettersson, and Yi - 2004

- None of them explores the details of in-kernel (or complex os) mechanisms.
Related work: Formal verification

- Usage of BLAST tool with control flow automata, along with techniques for state-space reduction, applied to the verification of safety properties of OS drivers for the Linux and Microsoft Windows NT kernels.
  - Henzinger, Jhala, Majumdar, and Sutre (2002)
- MAGIC, a tool for automatic verification of sequential C programs against finite state machine specifications.
  - Chaki, Clarke, Groce, Jha, and Veith (2004)
  - MAGIC has been used to verify locking correctness (deadlock-freedom) in the Linux kernel.
Related work: Linux kernel

- lockdep mechanism built into the Linux kernel, capable of identifying errors in using locking primitives that might eventually lead to deadlocks.
- Linux Memory Model
  - Alglave, Maranget, McKenney, Parri, and Stern (2018)
A model for thread synchronization

To the best of our knowledge, none of these techniques ventured into the challenging goal of building a formal model for the understanding and validation of the Linux PREEMPT RT kernel code sections responsible for such low-level operations such as task scheduling, IRQ and NMI management, and their delicate interplay, as done in this paper.
Agenda

- Introduction
- Related Work
- Background on automata theory
- Proposed approach
- Application of the model
- Conclusions and future work
Background

- 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;
    - $f$ is the transition function $= (X \times E) \rightarrow X$;
    - $x_0$ = Initial state;
    - $X_m$ = set of final states.
- The language - or traces - generated/recognized by $G$ is the $L(G)$. 
Graphical format
Modeling of complex systems

- Rather than modeling the system as a single automaton, the modular approach uses **generators** and **specifications**.
  - Generators:
    - Independent subsystems models
    - Generates all chain of events (without control)
  - Specification:
    - Control/synchronization rules of two or more subsystems
    - Blocks some events
- The parallel composition operation synchronizes the generators and specifications.
  - The result is an automaton with all chain of events possible in a controlled system.
Agenda

- Introduction
- Related Work
- Background on automata theory
- Proposed approach
- Application of the model
- Conclusions and future work
Proposed approach

Informal knowledge → Modeling → Validation

Kernel → Tracing → Validation

automaton.dot

perf.data

Does the model matches trace?

No

Yes

OK
Modeling
Example of generators: G05, G01 and G04

`sched_need_resched`

`need_resched`  `sleepable`

`sched_waking`

`sched_set_state_runnable`

`sched_set_state_sleepable`

`runnable`

`thread`

`schedule_entry`

`schedule_exit`

`sched`
Automata & Kernel events
## Model: IRQ events

<table>
<thead>
<tr>
<th>Automaton event</th>
<th>Kernel event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw_local_irq_disable</td>
<td>irq:local_irq_disable</td>
<td>Begin IRQ handler</td>
</tr>
<tr>
<td>hw_local_irq_enable</td>
<td>irq:local_irq_enable</td>
<td>Return IRQ handler</td>
</tr>
<tr>
<td>local_irq_disable</td>
<td>irq:local_irq_disable</td>
<td>Mask IRQs</td>
</tr>
<tr>
<td>local_irq_enable</td>
<td>irq:local_irq_enable</td>
<td>Unmask IRQs</td>
</tr>
<tr>
<td>nmi_entry</td>
<td>irq_vectors:nmi</td>
<td>Begin NMI handler</td>
</tr>
<tr>
<td>nmi_exit</td>
<td>irq_vectors:nmi</td>
<td>Return NMI Handler</td>
</tr>
</tbody>
</table>
# Model: Preemption/Scheduler related events

<table>
<thead>
<tr>
<th>Automaton event</th>
<th>Kernel event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preempt_disable</td>
<td>sched:sched_preempt_disable</td>
<td>Disable preemption</td>
</tr>
<tr>
<td>preempt_enable</td>
<td>sched:sched_preempt_enable</td>
<td>Enable preemption</td>
</tr>
<tr>
<td>preempt_disable_sched</td>
<td>sched:sched_preempt_disable</td>
<td>Disable preemption to call the scheduler</td>
</tr>
<tr>
<td>preempt_enable_sched</td>
<td>sched:sched_preempt_enable</td>
<td>Enables preemption returning from the scheduler</td>
</tr>
<tr>
<td>schedule_entry</td>
<td>sched:sched_entry</td>
<td>Begin of the scheduler</td>
</tr>
<tr>
<td>schedule_exit</td>
<td>sched:sched_exit</td>
<td>Return of the scheduler</td>
</tr>
<tr>
<td>sched_need_resched</td>
<td>sched:set_need_resched</td>
<td>Set need resched</td>
</tr>
</tbody>
</table>
## Model: State of threads events

<table>
<thead>
<tr>
<th>Automaton event</th>
<th>Kernel event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_waking</td>
<td>sched:sched_waking</td>
<td>Activation of a thread</td>
</tr>
<tr>
<td>sched_set_stateRunnable</td>
<td>sched:sched_set_state</td>
<td>Thread is runnable</td>
</tr>
<tr>
<td>sched_set_stateSleepable</td>
<td>sched:sched_set_state</td>
<td>Thread can go to sleepable</td>
</tr>
</tbody>
</table>
## Model: Context switch events

<table>
<thead>
<tr>
<th>Automaton event</th>
<th>Kernel event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_switch_in</td>
<td>sched:sched_switch</td>
<td>Switch in of the thread under analysis</td>
</tr>
<tr>
<td>sched_switch_suspend</td>
<td>sched:sched_switch</td>
<td>Switch out due to a suspension of the thread under analysis</td>
</tr>
<tr>
<td>sched_switch_preempt</td>
<td>sched:sched_switch</td>
<td>Switch out due to a preemption of the thread under analysis</td>
</tr>
<tr>
<td>sched_switch_blocking</td>
<td>sched:sched_switch</td>
<td>Switch out due to a blocking of the thread under analysis</td>
</tr>
<tr>
<td>sched_switch_in_o</td>
<td>sched:sched_switch</td>
<td>Switch in of another thread</td>
</tr>
<tr>
<td>sched_switch_out_o</td>
<td>sched:sched_switch</td>
<td>Switch out of another thread</td>
</tr>
</tbody>
</table>
## Model: Mutex events

<table>
<thead>
<tr>
<th>Automaton event</th>
<th>Kernel event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex_lock</td>
<td>lock:rt_mutex_lock</td>
<td>Requested a RT Mutex</td>
</tr>
<tr>
<td>mutex_blocked</td>
<td>lock:rt_mutex_block</td>
<td>Blocked in a RT Mutex</td>
</tr>
<tr>
<td>mutex_acquired</td>
<td>lock:rt_mutex_acquired</td>
<td>Acquired a RT Mutex</td>
</tr>
<tr>
<td>mutex_abandon</td>
<td>lock:rt_mutex_abandon</td>
<td>Abandoned the request of a RT Mutex</td>
</tr>
</tbody>
</table>
Model: Write lock events

<table>
<thead>
<tr>
<th>Automaton event</th>
<th>Kernel event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write_lock</td>
<td>lock:rwlock_lock</td>
<td>Requested a R/W Lock or Sem as writer</td>
</tr>
<tr>
<td>write_blocked</td>
<td>lock:rwlock_block</td>
<td>Blocked in a R/W Lock or Sem as writer</td>
</tr>
<tr>
<td>write_acquired</td>
<td>lock:rwlock_acquired</td>
<td>Acquired a R/W Lock or Sem as writer</td>
</tr>
<tr>
<td>write_abandon</td>
<td>lock:rwlock_abandon</td>
<td>Abandoned a R/W Lock or Sem as writer</td>
</tr>
</tbody>
</table>
## Model: Read lock events

<table>
<thead>
<tr>
<th>Automaton event</th>
<th>Kernel event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_lock</td>
<td>lock:rwlock_lock</td>
<td>Requested a R/W Lock or Sem as reader</td>
</tr>
<tr>
<td>read_blocked</td>
<td>lock:rwlock_block</td>
<td>Blocked in a R/W Lock or Sem as reader</td>
</tr>
<tr>
<td>read_acquired</td>
<td>lock:rwlock_acquired</td>
<td>Acquired a R/W Lock or Sem as reader</td>
</tr>
<tr>
<td>read_abandon</td>
<td>lock:rwlock_abandon</td>
<td>Abandoned a R/W Lock or Sem as reader</td>
</tr>
</tbody>
</table>
Generators and Specifications
Components: Generators

<table>
<thead>
<tr>
<th>Name</th>
<th>States</th>
<th>Events</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>G01 Sleepable or runnable</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G02 Context switch</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>G03 Context switch other thread</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G04 Scheduling context</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G05 Need resched</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G06 Preempt disable</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>G07 IRQ Masking</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G08 IRQ handling</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G09 NMI</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>G10 Mutex</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>G11 Write lock</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>G12 Read lock</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Name</td>
<td>States</td>
<td>Events</td>
<td>Transitions</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>S01 Sched in after wakeup</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>S02 Resched and wakeup sufficiency</td>
<td>3</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>S03 Scheduler with preempt disable</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>S04 Scheduler doesn't enable preemption</td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>S05 Scheduler with interrupt enabled</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>S06 Switch out then in</td>
<td>2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>S07 Switch with preempt/irq disabled</td>
<td>3</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>S08 Switch while scheduling</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>S09 Schedule always switch</td>
<td>3</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>S10 Preempt disable to sched</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>S11 No wakeup right before switch</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>S12 IRQ context disable events</td>
<td>2</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>S13 NMI blocks all events</td>
<td>2</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>S14 Set sleepable while running</td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>S15 Don't set runnable when scheduling</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>S16 Scheduling context operations</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Components: Specifications (part 2)

<table>
<thead>
<tr>
<th>Name</th>
<th>States</th>
<th>Events</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>S17 IRQ disabled</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>S18 Schedule necessary and sufficient</td>
<td>7</td>
<td>9</td>
<td>22</td>
</tr>
<tr>
<td>S19 Need resched forces scheduling</td>
<td>7</td>
<td>27</td>
<td>59</td>
</tr>
<tr>
<td>S20 Lock while running</td>
<td>2</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>S21 Lock while preemptive</td>
<td>2</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>S22 Lock while interruptible</td>
<td>2</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>S23 No suspension in lock algorithms</td>
<td>3</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>S24 Sched blocking if blocks</td>
<td>3</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>S25 Need resched blocks lock ops</td>
<td>2</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>S26 Lock either read or write</td>
<td>3</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>S27 Mutex doesn't use rw lock</td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>S28 RW lock does not sched unless block</td>
<td>4</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>S29 Mutex does not sched unless block</td>
<td>4</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>S30 Disable IRQ in sched implies switch</td>
<td>5</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>S31 Need resched preempts unless sched</td>
<td>3</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>S32 Does not suspend in mutex</td>
<td>3</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>S33 Does not suspend in rw lock</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>
Model

- The final model has:
  - 13906 states;
  - 31708 transitions;
  - It would be impossible to model it directly.

- Using the modular approach, the final model is composed of:
  - 34 events;
  - 12 generators;
  - 33 specifications.
    - The most complex module (a specification) has 7 states!
Model Verification
Model Validation: *perf task model*

![Diagram showing the process of model validation involving perf interface, Trace to Event Interpreter, and Graphviz / Automaton. The steps include determining if the operation is ok, processing raw tracepoint, event accept/deny, and restarting the process as needed.](image-url)
### perf task_model output

1: Reference model: isorc.dot
2: +----> +=thread of interest - .=other threads
3: | +--> T=Thread - I=IRQ - N=NMI
4: | |
5: | | TID | timestamp | cpu | event | state | safe?
6: . T   8 436.912532   [000]     preempt_enable -> q0 safe
7: . T   8 436.912534   [000]    local_irq_disable -> q8102
8: . T   8 436.912535   [000]    preempt_disable -> q19421
9: . T   8 436.912535   [000]     sched_waking -> q99
10:. . T 8 436.912535   [000]    sched_need_resched -> q14076
11:. . T 8 436.912535   [000]    local_irq_enable -> q1965
12:. . T 8 436.912536   [000]    preempt_enable -> q12256
13:. . T 8 436.912536   [000]  preempt_disable_sched -> q18615,q23376
14:. . T 8 436.912536   [000]    schedule_entry -> q16926,q17108,q2649
15:. . T 8 436.912537   [000]    local_irq_disable -> q11700,q14046,q21391
16:. . T 8 436.912537   [000]    sched_switch_out_o -> q10337,q20018,q21933
17:. . T 8 436.912537   [000]    sched_switch_in -> q10268,q20126
18: + T 1840 436.912537   [000]    local_irq_enable -> q20036
19: + T 1840 436.912538   [000]    schedule_exit -> q21033
Agenda

- Introduction
- Related Work
- Background on automata theory
- Proposed approach
- Application of the model
- Conclusions and future work
Example of application of the Model: Analysis of activation of the highest priority thread.
Highest priority task activation: Generators

sched_need_resched

need_resched -> sched_waking

sleepable

sched_set_state_runnable

sched_set_state_sleepable

runnable

thread

schedule_entry

sched

schedule_exit
Highest priority task activation: Generators
Highest priority task activation: Generators

- **no_irq**
  - local_irq_disable
  - local_irq_enable
  - hw_local_irq_disable
  - hw_local_irq_enable

- **irq_disabled**
- **irq_running**
Highest priority task activation: Generators

- **running**
  - Transition: `sched_switch_out_o` to **preempted**
  - Transition: `sched_switch_in_o`

- **not_running**
  - Transition: `sched_switch_in`
  - Transition: `sched_switch_suspend`
  - Transition: `sched_switch_preempt`
  - Transition: `sched_switch_blocking`
  - Transition: Back to **running**
Highest priority task activation: Specifications
Highest priority task activation: Specifications

- \text{schedule\_entry}
- \text{schedule\_exit}

\text{can\_sched} \quad \text{local\_irq\_disable} \quad \text{local\_irq\_enable} \quad \text{cant\_sched}

\text{cant\_sched} \quad \text{preempt\_disable\_sched} \quad \text{preempt\_enable\_sched} \quad \text{can\_sched}
Highest priority task activation: Specifications

- sched_switch_in
- sched_switch_in_o
- sched_switch_suspend
- sched_switch_preempt
- sched_switch_out_o
- sched_switch_blocking

**Diagram:**
- **thread**
  - schedule_entry
  - schedule_exit
  - sched

45
Highest priority task activation: Specifications

sched_switch_in
sched_switch_suspend
sched_switch_preempt
sched_switch_in_o
sched_switch_out_o
sched_switch_blocking

disabled

local_irq_enable
preempt_enable_sched

local_irq_disable
preempt_disable_sched

local_irq_disable
preempt_disable_sched

local_irq_enable
preempt_enable_sched

enabled

p_xor_i
Highest priority task activation: Specifications
Other results

- By modeling the *expected* behavior, we can catch cases in which the kernel does not behave as expected.
  - We found two problems on kernel
    - One unexpected call to schedule():
      - Schedule called in vain.
        - Resulted in a kernel patch.
    - Perf & Ftrace missing events:
      - It is a problem in the trace recursion control
        - Patch suggested and under discussion
Agenda

- Introduction
- Related Work
- Background on automata theory
- Proposed approach
- Early results
- Conclusions and future work
Conclusion

- The definition of the operations that affect the timing behavior of threads are fundamental for the improvement of real-time Linux.
- Linux is complex! But the complexity was successfully “broken” by using the modular approach.
- The model is useful to understand the kernel dynamics.
- But also to find problems in the kernel!
Future work

- Model the multicore behavior.
- Try to fit the approach in existing schedulability analysis methods.
- Create more efficient model checker for Linux.
  - To find regressions: both timing and logical regressions.
  - This was discussed with developers and it is a wish.
All is OPEN

- All the source code (Kernel, model,...) are available at the Linux Task Model page:
  - http://bristot.me/linux-task-model/
Thanks!
Questions?