Mind the gap:

between real-time Linux and real-time theory

Part I

Daniel Bristot de Oliveira

In the begin

In the begin a program was only a **logical sequence**,
Then gosh said: we can't wait forever, we need to put **time** on this,

Since then we have two problems:
The **logical correctness**, and the **timing correctness**.

In theory…

The systems defined as a set of tasks τ
Each task is a set of variables that defines its timing behavior, e.g.,

$$\tau_i = \{ P, C, D, B, J \}$$

Then, they try to define/develop a scheduler in such way that,
for each task i in τ:

the response time of $\tau_i < D_i$

For task level fixed priority scheduler:

$$\forall\, task\, i \in \tau:$$

$$W_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{P_j} \right\rceil C_j$$

$$R_i = W_i + J_i$$

$$is\, schedulable \Leftrightarrow \forall\, task\, i \in \tau \,|\, R_i < D_i$$

For Early Deadline First

$$\forall\, task\, i \in \tau:$$

$$U_i = \frac{C_i}{P_i}$$

$$is\, schedulable \Leftrightarrow \forall\, task\, i \in \tau \left| \sum U_i < 1 \right.$$

The development of a new scheduler is done with mathematical reasoning.

But generally, they relax in the task model

- The system is fully preemptive;
- Tasks are completely independent;
- Operations are atomic;
- There is no overhead.

We can’t say that these assumptions are not realistic...

But, what is our reality?

Our reality

- The system is not fully preemptive;
- Tasks are not completely independent;
- Operations are not atomic;
- There is overhead.

Math side: But talk is cheap...

Dev side: Read the code, it is there, boy!

Math side: Talk is cheap...

Show me the math!

Towards a Linux task model

- Inside our mind, we have an implicit task model:
  - We know preemption causes latency
  - We know the difference in the behavior of a mutex and the spin lock
  - We know we have interrupts
- But, how do we explain these things without missing details?
  - Natural language is ambiguous…
  - e.g., preemption disabled is bad for latency, right?

Towards a Linux task model

- We need an explicit task model
  - Using a formal language/method
  - Abstracting the code
  - Without losing contact with the terms that we use in practice.

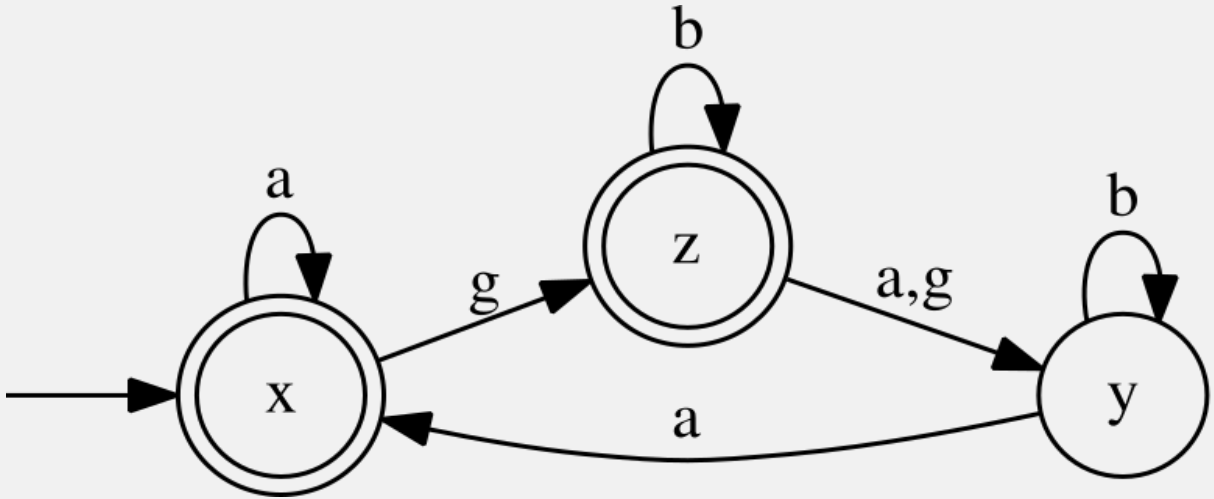
Toward a Linux task model

- Linux developers use tracing features to analyze 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 is can be used in the formalization of system.
- So, why not try to describe Linux using a DES method?

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 = $(X \text{ x } E) \rightarrow X$;
    - x_0 = Initial state;
    - X_m = set of final states.
- The language - or traces - generated/recognized by G is the L(G).

Graphical format

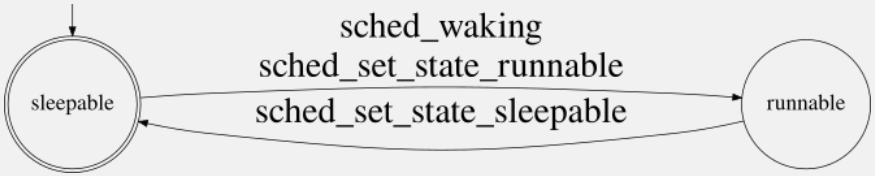


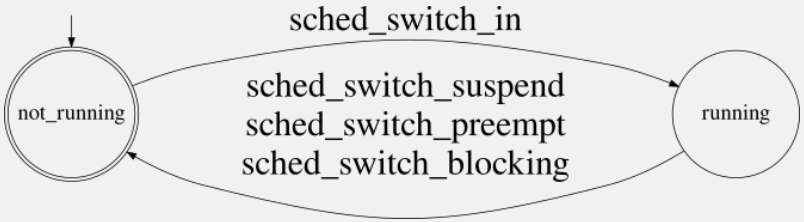
Modeling of complex systems

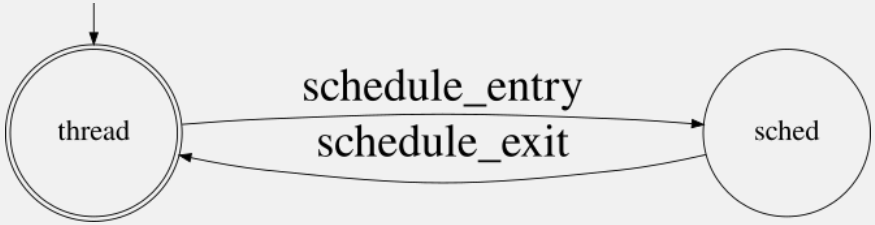
- Rather than modeling the system as a single automaton, the modular approach uses **generators** and **specifications**.
  - Generators:
    - Independent subsystems models
    - Generates all chain of events (without control)
  - Specification:
    - Control/synchronization rules of two or more subsystems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

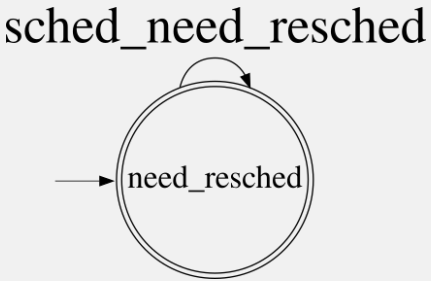
Example of models

Generators of events

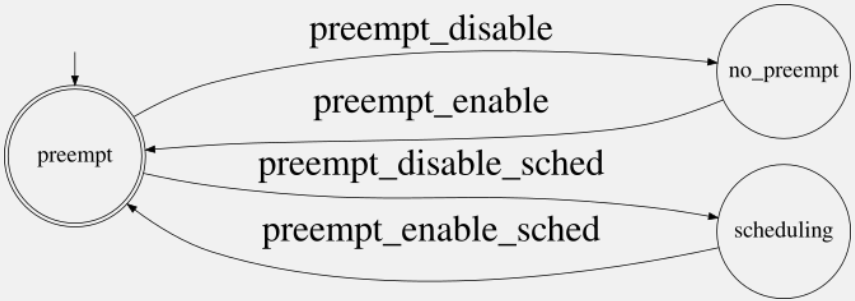


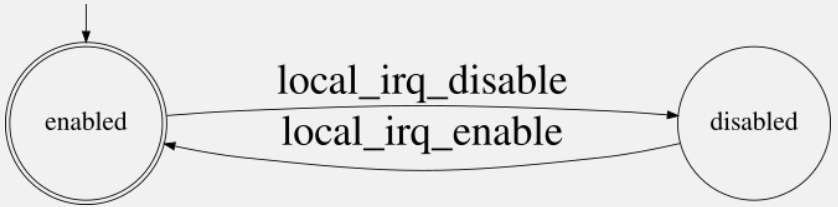


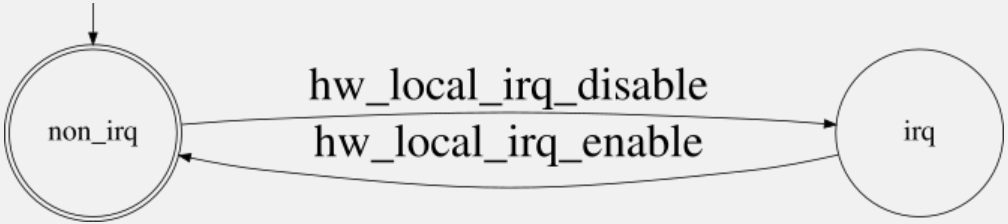




Generators of events

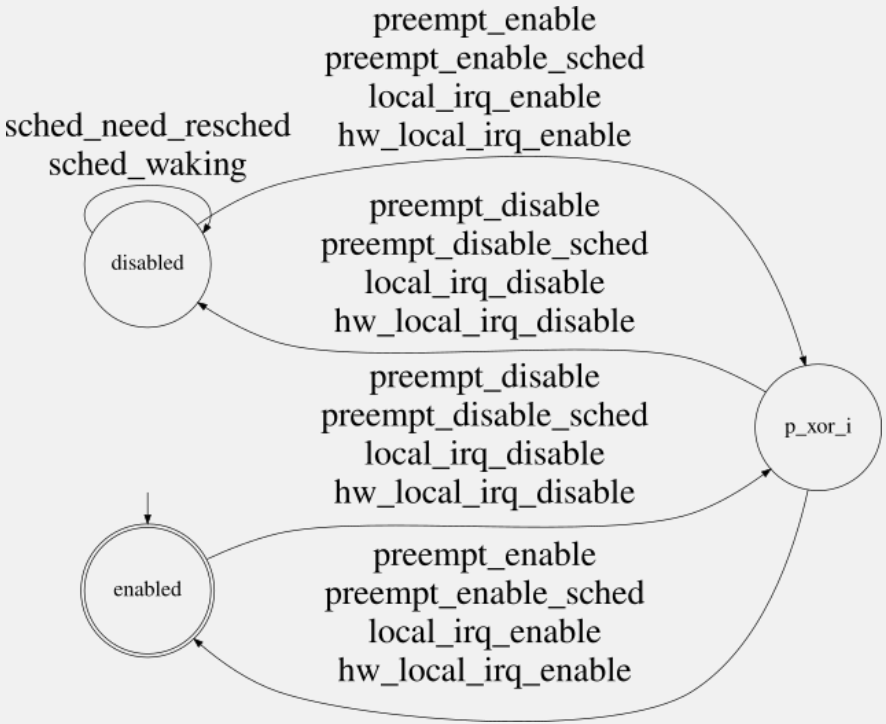




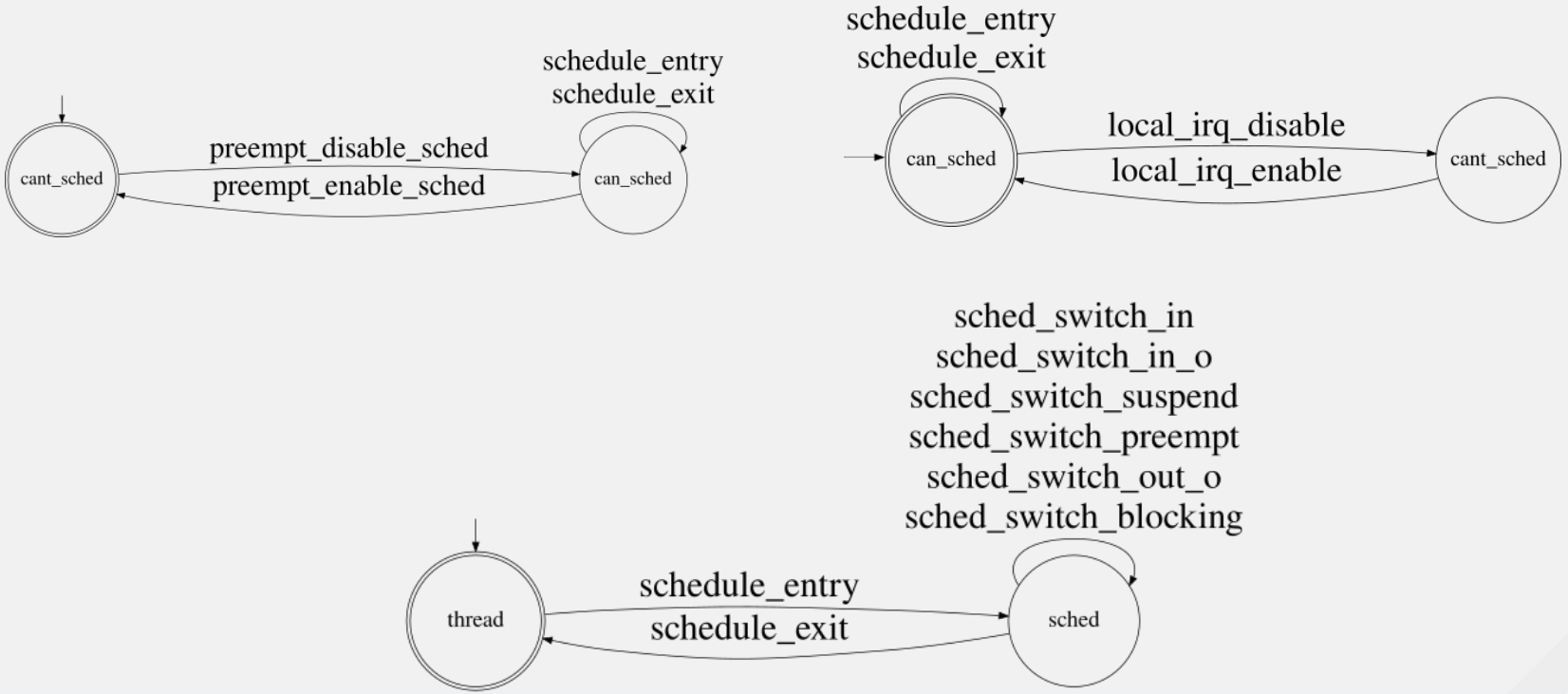


Boia,
This is boring...
de!

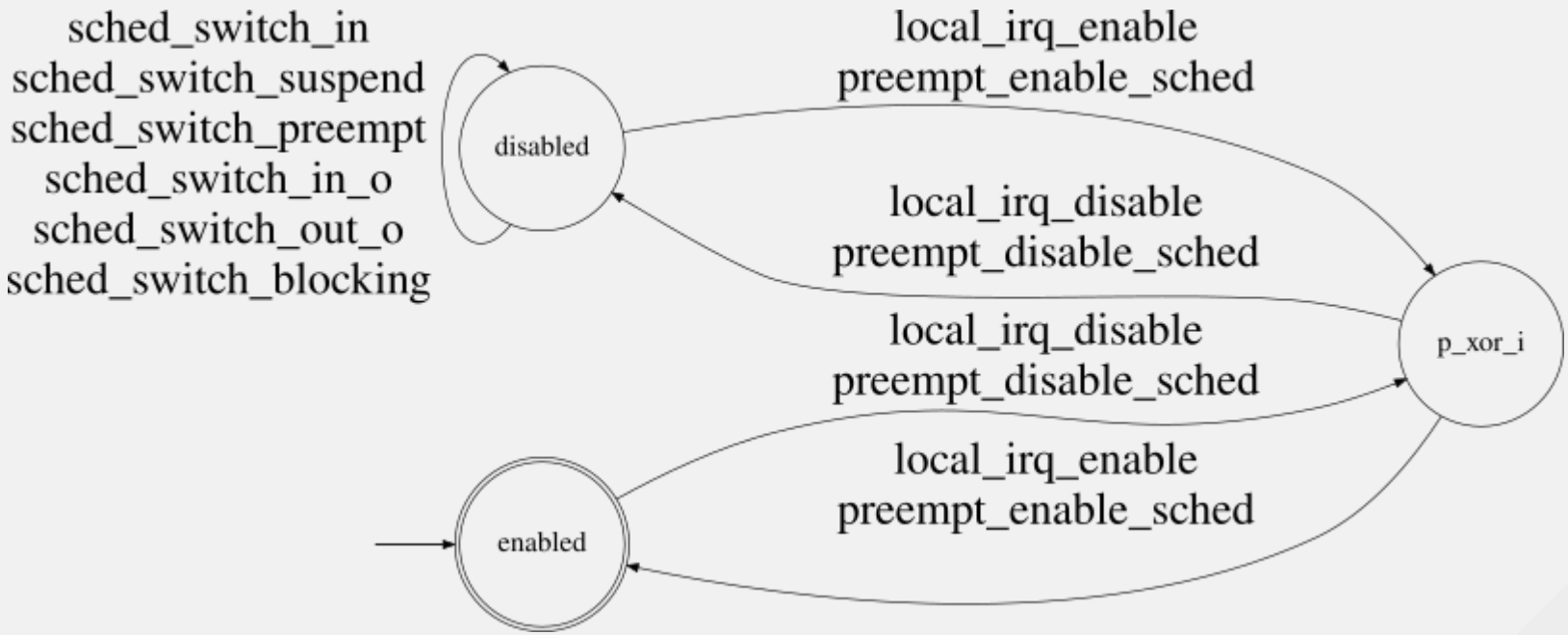
Specifications: Sufficiency conditions



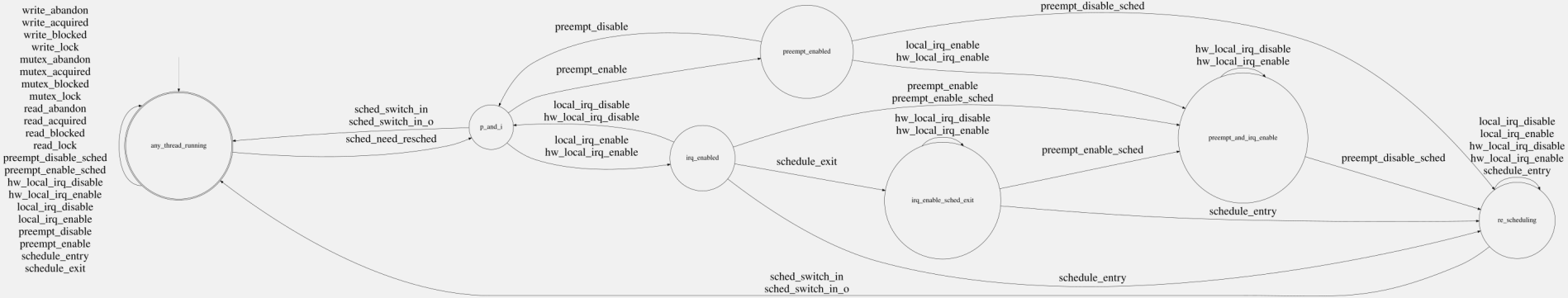
Specifications: Sufficiency conditions



Specifications: Sufficiency conditions



Specifications: Necessary condition



Synchronizing the modules, we have the model

The complete model has:

- 12 generators + 33 specifications
- 34 different events
- 13906 states!

The benefit of this:

- Validating the model against the kernel, and vice-versa, is O(1)
- One kernel event generates one automata trasition.

Nice! But what do we do with this information?

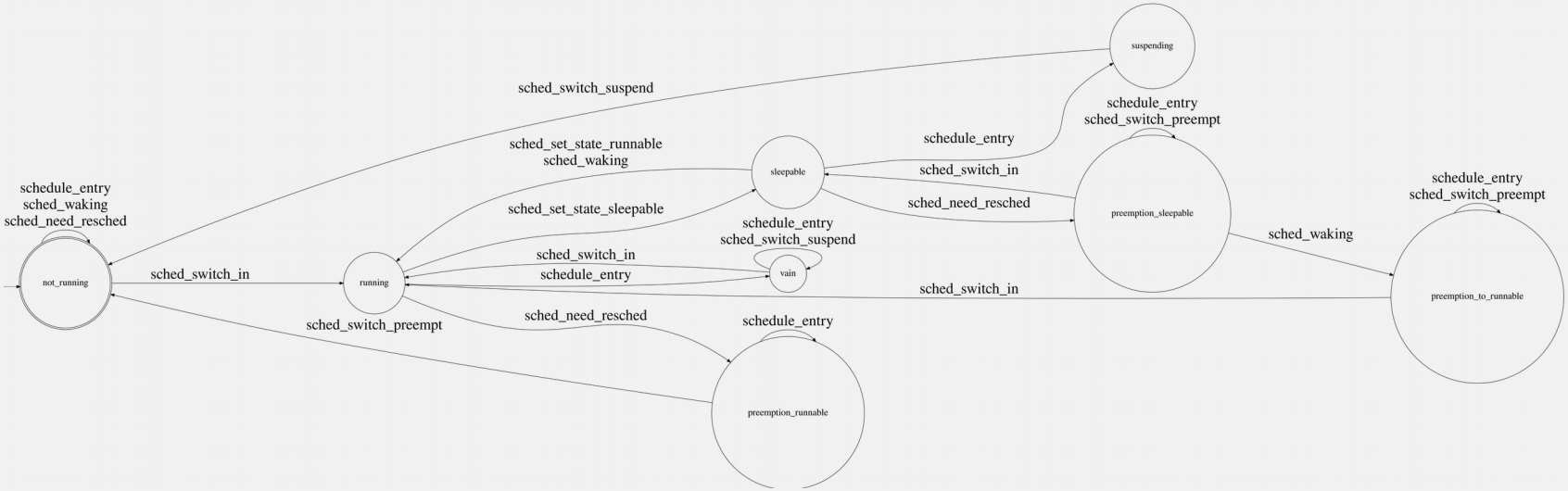
In the begin...

We have two problems:

- The **logical correctness**,
- The **timing correctness**.

The model helps in both cases.

Calling scheduler

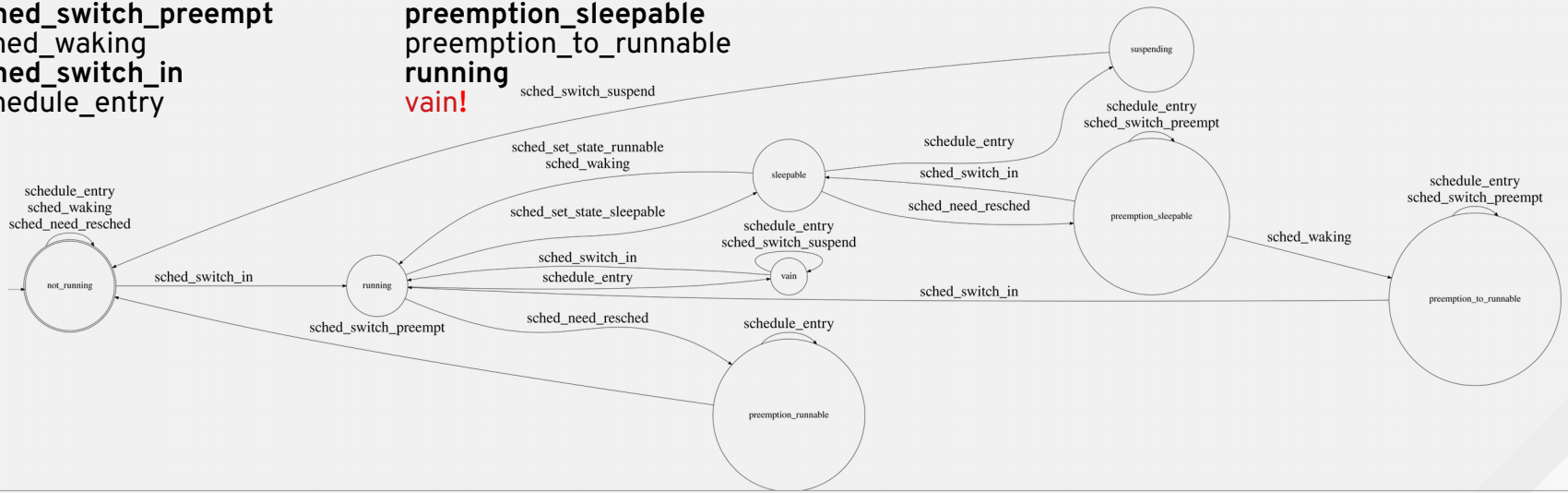


Reference tracing:

```
 1:   ktimersoftd/0   8 [000]   784.425631:  sched:sched_switch: ktimersoftd/0:8 [120] R ==> kworker/0:2:728 [120]
 2:     kworker/0:2 728 [000]   784.425926:  sched:sched_set_state: sleepable
 3:   kworker/0:2   728 [000]   784.425932:  sched:sched_waking: comm=kworker/0:1 pid=724  prio=120 target_cpu=000
 4:     kworker/0:2 728 [000]   784.425936:  sched:set_need_resched: comm=kworker/0:2 pid=728
 5:     kworker/0:2 728 [000]   784.425941:  sched:sched_entry: at preempt_schedule_common
 6:     kworker/0:2 728 [000]   784.425945:  sched:sched_switch: kworker/0:2:728 [120] R ==> kworker/0:1:724 [120]
 7:  irq/14-ata_piix 86 [000]   784.426515:  sched:sched_waking: comm=kworker/0:2 pid=728 prio=120 target_cpu=000
 8:     kworker/0:1 724 [000]   784.426610:  sched:sched_switch: kworker/0:1:724 [120] t ==> kworker/0:2:728 [120]
 9:     kworker/0:2 728 [000]   784.426616:  sched:sched_entry: at schedule
10:     kworker/0:2 728 [000]   784.426619:  sched:sched_switch: kworker/0:2:728 [120] R ==> kworker/0:2:728 [120]
```

Calling scheduler

Event	State
sched_switch_in	**running**
sched_set_state_sleepable	sleepable
sched_need_resched	**preemption_sleepable**
schedule_entry	preemption_sleepable
sched_switch_preempt	**preemption_sleepable**
sched_waking	preemption_to_runnable
sched_switch_in	**running**
schedule_entry	vain**!**



Logical correctness for task model

- Example of patch catch'ed with the model
  - [PATCH RT] sched/core: Avoid__schedule() being called twice, the second in vain
- I am doing the model verification in user-space now:
  - Using perf + (sorry, peterz) tracepoints
  - It works, but requires a lot of memory/data transfer:
    - Single core, 30 seconds = 2.5 GB of data
    - We don't need all the data, only from a safe state to the problem.
  - It performs well, because the automata verification is O(1).
  - But still, the amount of data is massive.

Should I move it to kernel?

- Think of a lockdep for PREEMPT_RT model:
  - If an unexpected event takes place, we explain why
  - Enabled in compilation time
  - Running in kernel would avoid copying data/keeping data after reaching a safe state
- This is helpful for safe critical systems
  - CI
  - We might face more problems with merge with the non-rt
  - It observes more than just latency

Timing correctness

- The latency is good!
- But the model provides way to decompose the latency
  - Preempt & IRQ disabled sections...
  - Scheduling overhead...
  - Locking overhead...
  - The response time...
  - These all helps to better identify the characteristics of -RT
    - And to find regressions in a more fine-grained way.

Ok, but this is a longer subject, we will talk about it at plumbers.

Resume

- There will be gains having more academic people working with “things that connect well” with Linux.
- The PREEMPT RT simplifies the task model enough to turn possible the modeling
- Using DES/Automata was not that hard as it seems.
- It is an ongoing work.
- The model opens other possibilities for the verification of the kernel-rt.

Thanks!


redhat


Sant'Anna
School of Advanced Studies – Pisa


UNIVERSIDADE FEDERAL
DE SANTA CATARINA