Mind the gap:

between real-time Linux and real-time theory

Part I

Daniel Bristot de Oliveira

In the beginning

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 $\tau_i < D_i$

redhat.

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

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

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

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

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.

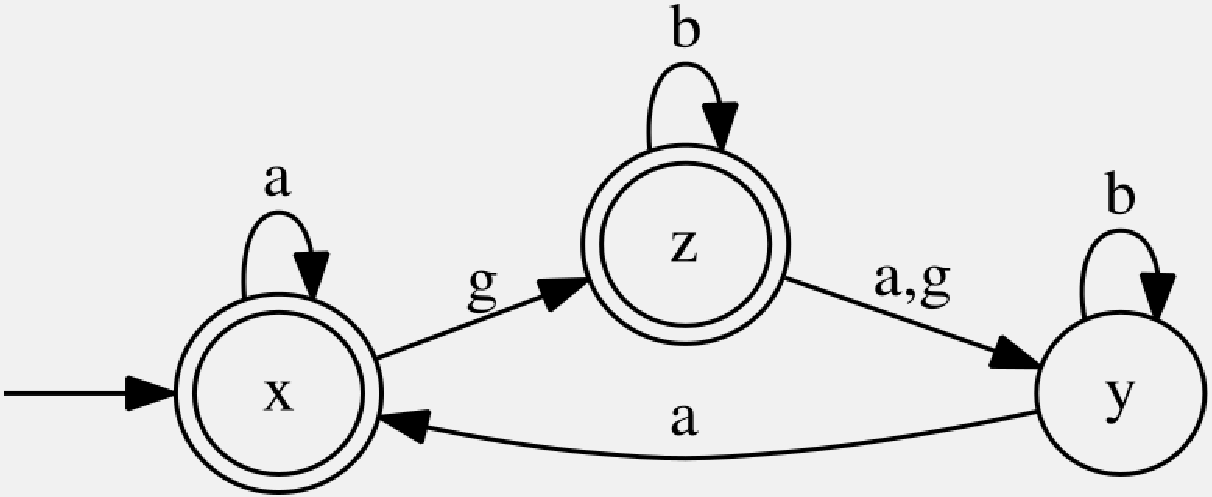
Toward a Linux task model

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

Graphical format

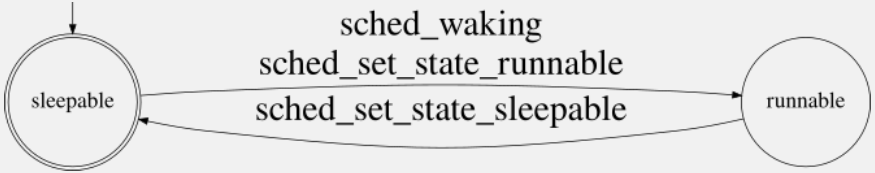


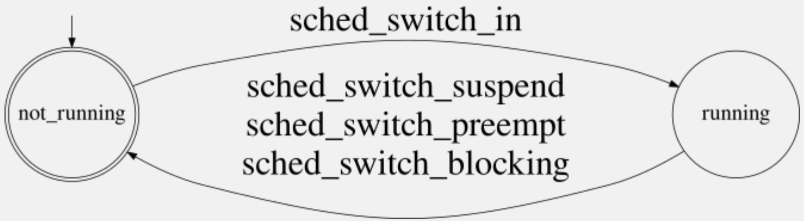
Modeling of complex systems

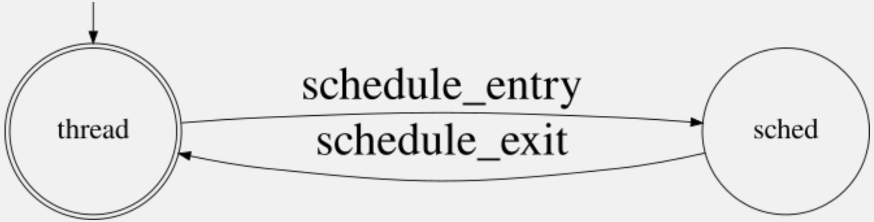
- Rather than modeling the system as a single automaton, the modular approach uses **generators** and **specifications**.
  - Generators:
    - Independent subsystems models
    - Generates all chain of events (without control)
  - Specification:
    - Control/synchronization rules of two or more subsystems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

Example of models

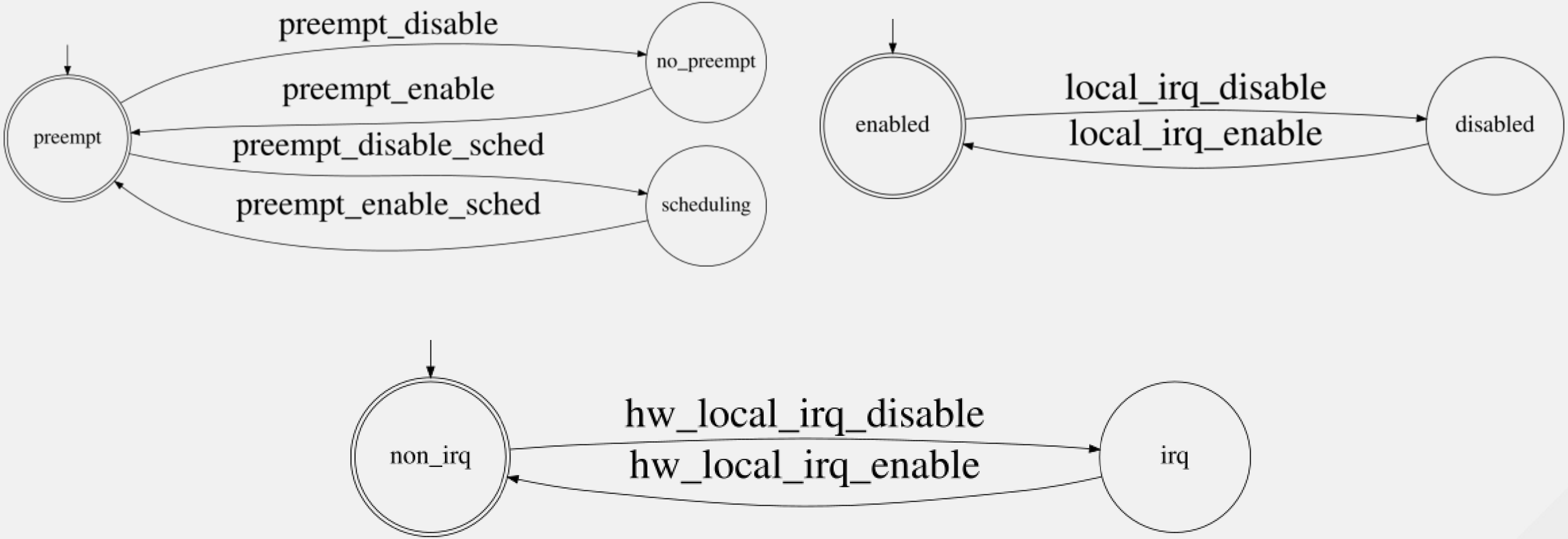
Generators of events





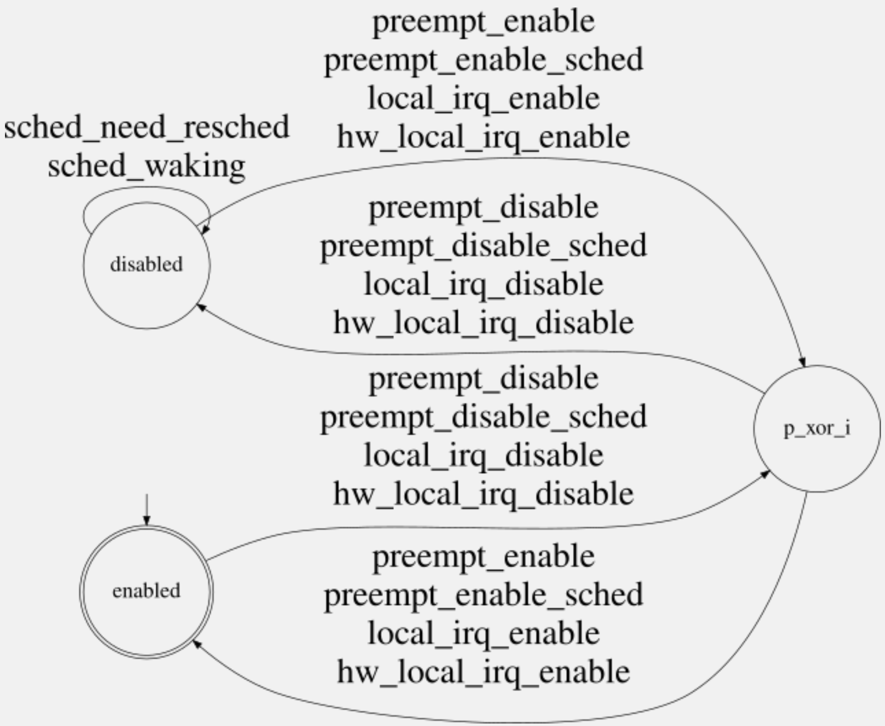


Generators of events

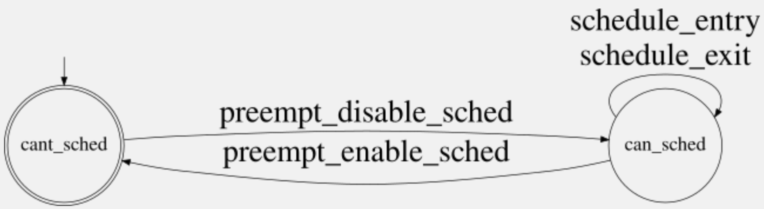


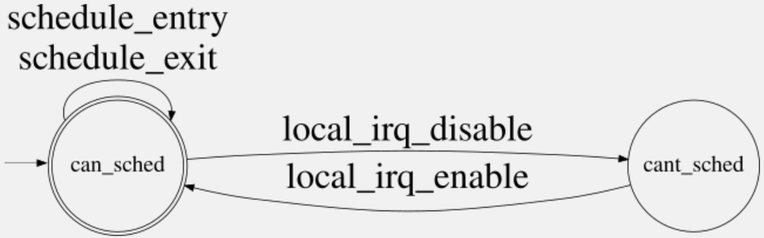
Eita, boia,
This is boring...

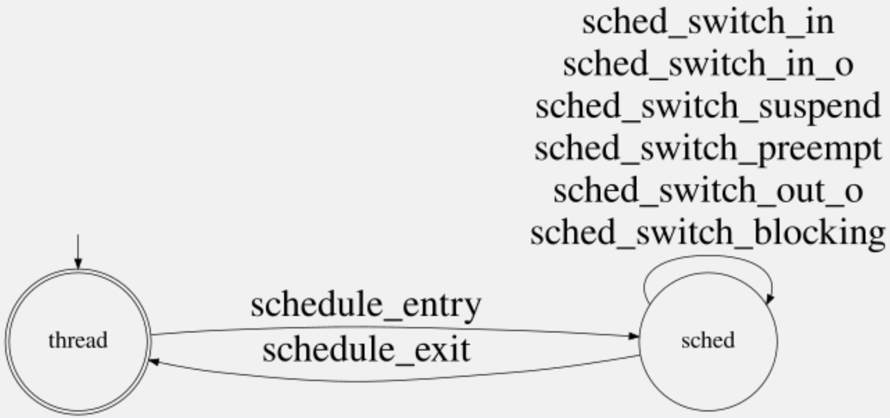
Specifications: Sufficiency conditions



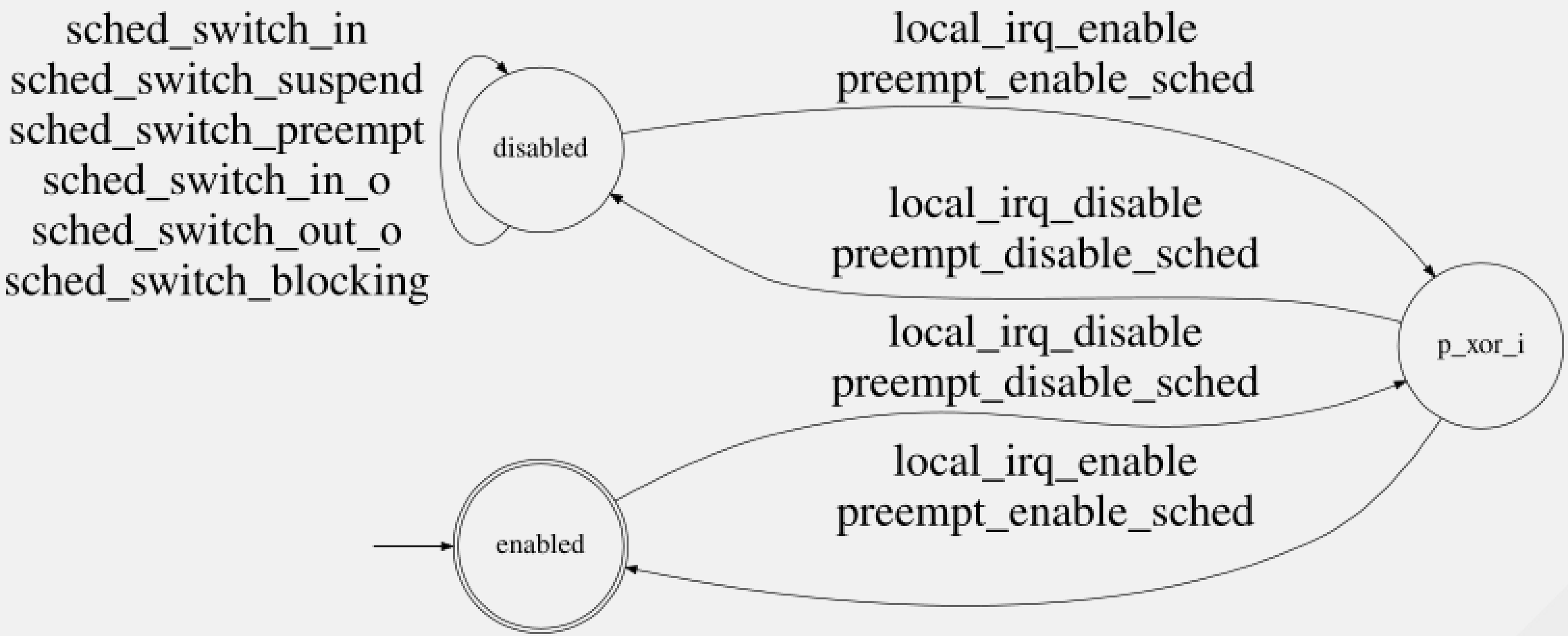
Specifications: Sufficiency conditions



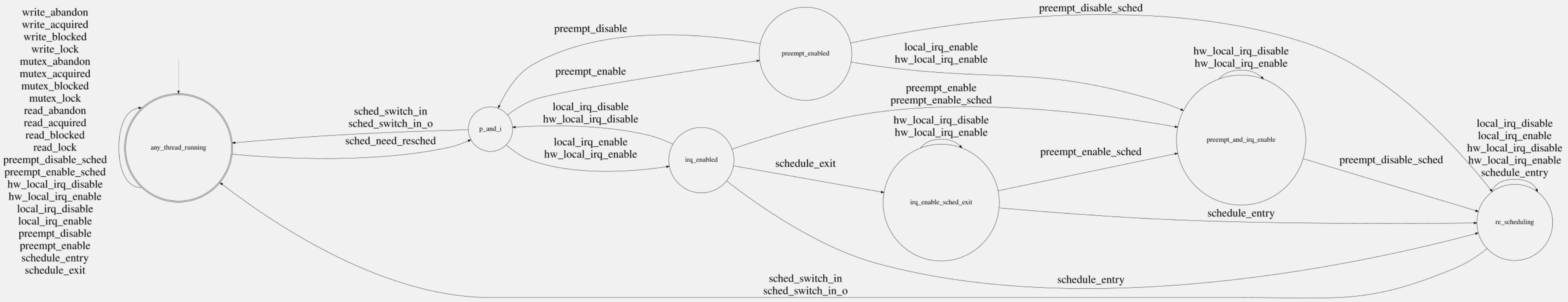




Specifications: Sufficiency conditions



Specifications: Necessary condition



Synchronizing the modules, we have the model

The complete model has:

- 12 generators + 33 specifications
- 34 different events
- > 10000 states!

The benefit of this:

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

Nice! But what do we do with this information?

What can we do with the model

From academic side:

- Understand the kernel dynamics.
- Develop of a theoretical system model for Linux.
- And... rework or develop new algorithms for Linux.

From development side:

- **A rutime model checker for the kernel - think of a lockdep for preemption.**
- **A new set of metrics - isolated metrics.**
- Static code analysis based in the assumptions - think of using coccinelle to find PREEMPT_RT bugs.

Questions?