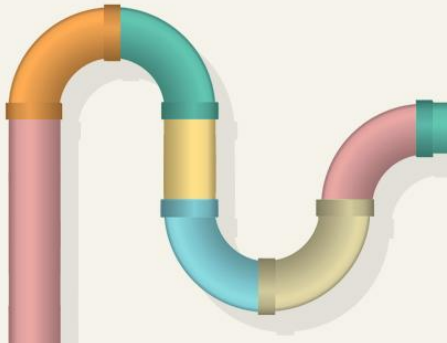
LINUX
PLUMBERS
CONFERENCE

August 24-28, 2020

The usage of PREEMPT_RT in safety-critical systems

what do we need to do?

Lukas Bulwahn



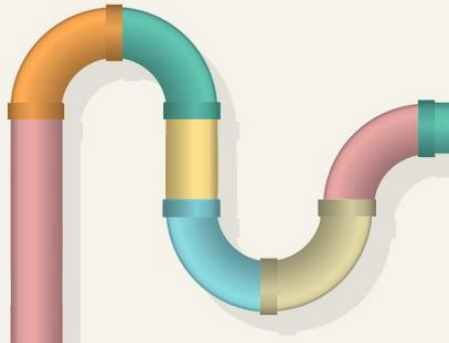
Core Question for this Discussion

Can we use confidently use a kernel with CONFIG_PREEMPT_RT=y in a safety-critical system?

Are we confident that a kernel with CONFIG_PREEMPT_RT=y is a high-quality kernel?

Why are we confident?

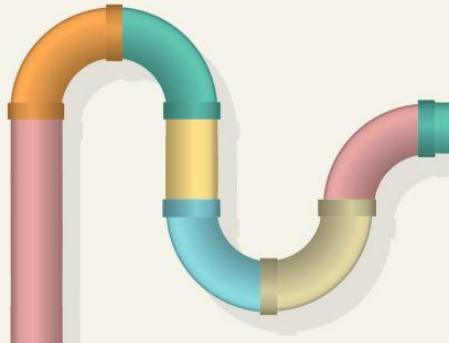
How can we make a third party gain confidence?



Action Items to these Questions

What needs to be done?

What are our recommendations within the Real-time Linux Community to our users?

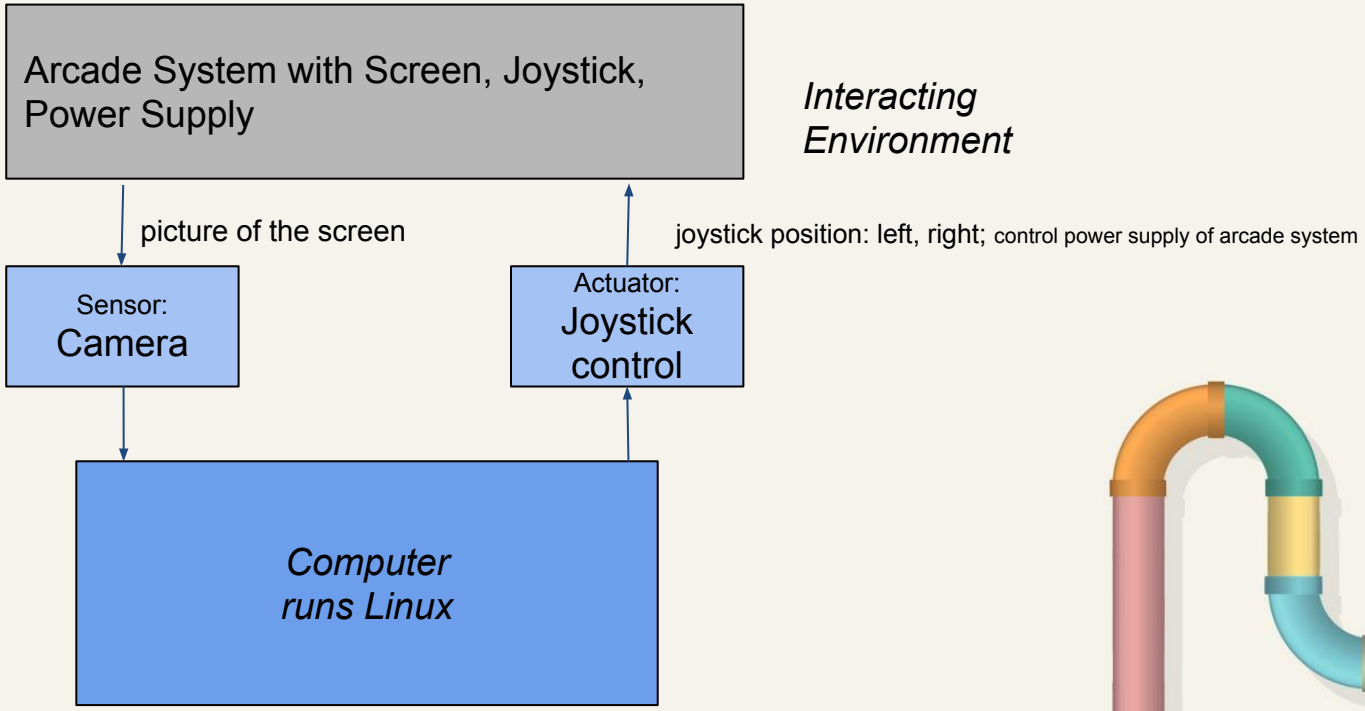


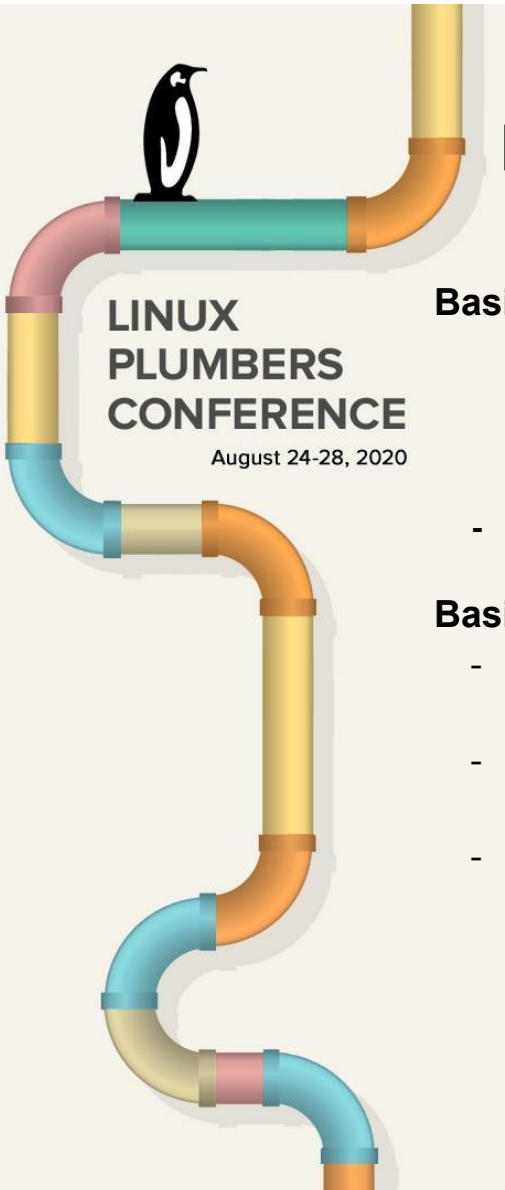
LINUX PLUMBERS CONFERENCE

August 24-28, 2020

Example System *(Purely Illustrative)*

We are going to design a robot playing an Arcade game through physical interaction watching the screen and using the joystick.
We are designing the robot, not the Arcade system/game.





Basic System & System Assumption

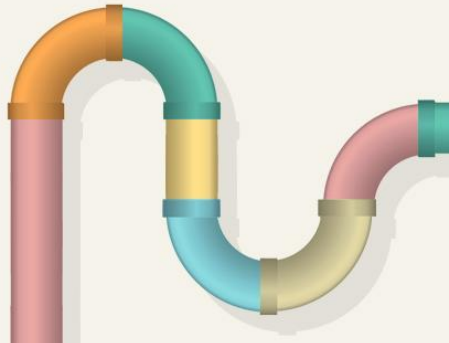
Basic system & environment loop:

- Sensor can see the screen (with a camera) and provides picture to computer
- Computer can compute the commands for the actuator, joystick left or joystick right
- Actuator obtains command and executes it.
- Arcade Game behaves accordingly and shows a new screen.

- **Hazardous Event: We reach the Game Over Screen.**

Basic System Assumption:

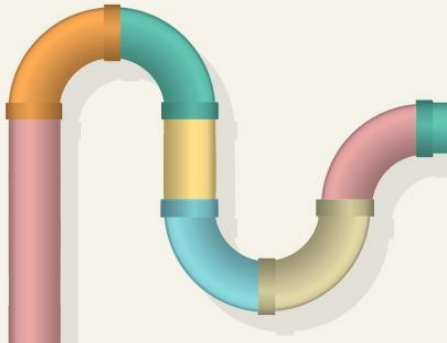
- There is no possibility that the robot through its internal working can influence the software behaviour of the Arcade machine other than through named Actuators in the physical world.
- If the whole control loop (Sensor, Computer, Actuator) reacts within less than 10 ms and the computer always sends the right command, we can never lose in the arcade system.
- If the robot performs badly, we will reach the Game Over Screen.
  i.e., If it sends the wrong commands, we will eventually lose the game and reach the Game Over Screen.



Software Design Assumption

Computer runs some version and configuration of the Linux Kernel.

- Application deciding the commands to be sent is running on Linux.
- Application relies on glibc and the kernel working as intended.
- Application provides the right command if glibc and kernel work as intended.



LINUX PLUMBERS CONFERENCE
August 24-28, 2020

Two Possible System Variants

System Variant A

Linux Computer:

- can cause harm with wrong commands sent
- cannot cause harm with commands sent late
- but still basic interest to send commands in time.

"Safety-critical and real-time, but real-time is not safety-critical"

Focus for this session here!

System Variant B

Linux Computer:

- can cause harm with wrong commands sent
- can cause harm with commands sent late

"Safety-critical and real-time, and real-time is safety-critical"

Focus for a later BoF or LPC 2021 if interest

System Variant A

The System A has the possibility to simply pull the plug on the Arcade Computer.

- Actuator can control joystick {left, right};
- **Actuator can pull the plug on the Arcade, and the Arcade is immediately shut-off.**

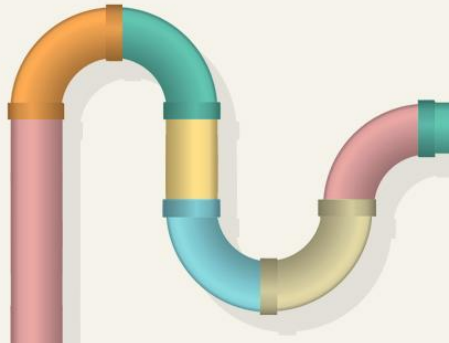
Refined System Design:

- Computer sends the command left or right or no command to the Actuator.
- If no command reaches the Actuator within a 10-ms time window of the last command, the Actuator pulls the plug.
- The Actuator has its own reliable clock.
- The Actuator clock never runs late compared to the clock in the real world.

The overall system and the computer and its application can be hazardous.

- If a command is provided, it needs to be the right one (otherwise we might lose the Arcade game)
- However, the Computer providing a command not *in time* to the Actuator is not hazardous, as the Actuator will pull the plug and the Arcade is off (so, no Game Over Screen).

It still annoying if our robot switches off the arcade system “very/too often”.



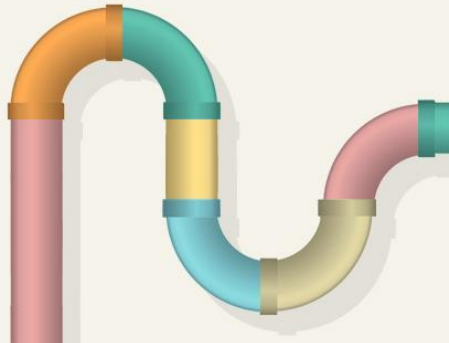
Recommendations on Kernel Selection for this Design

What are our expert recommendations for Selecting the Kernel (version, options) within the Linux Real-time Community? What is our rationale?

Is there a clear consensus among the community on those recommendations?

If not, why do we have different standpoints?

Which points deserve a technical investigation to make recommendations?



Quality Comparison

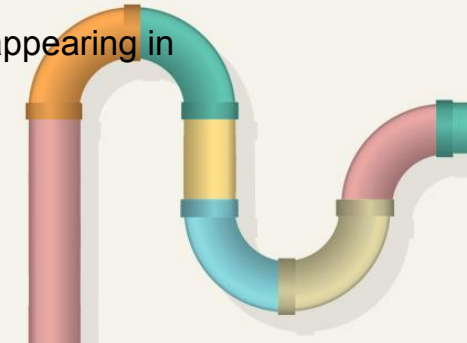
A kernel with CONFIG_PREEMPT_RT=y is

A. Better in quality (meeting functional requirements, stability, reliability, security) than
B. As good quality as
C. Worse in quality than
D. different in quality, but I do not know how, to

the same kernel with CONFIG_PREEMPT_RT=n.

In other words:

Are we risking to observe less or more (in numbers or in severity) bugs appearing in our system with CONFIG_PREEMPT_RT=y
compared to the CONFIG_PREEMPT_RT=n system?

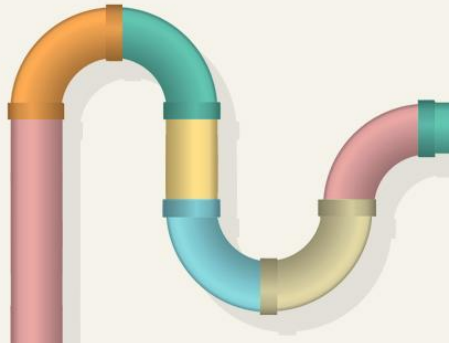


Further Discussion

Which arguments can we provide to support our claim?

Where and how can we find “hard facts”, i.e., empirical, measurable, or quantifiable evidence, for our claims and arguments?

What need we see most important be done to claim a kernel with CONFIG_PREEMPT_RT=y to meet the same quality expectations as a kernel with CONFIG_PREEMPT_RT=n?



Kernel Version Selection

Assumptions on Real-time Linux Kernel Releases

- “PREEMPT_RT is mainline with kernel release v5.10 :)”

CONFIG_PREEMPT_RT=y is supported on the computer’s architecture without any need for further out-of-tree patches.

- “v5.10.y is a well-maintained LTS release, just as all before”

What is our recommendation wrt. kernel version to get the highest-quality RT kernel?

A. Use the “oldest” v4.4-rt stable
B. Use v5.10 LTS once RT is mainline
C. Always update and use the latest kernel release after v5.10
D. It does not matter.
E. I do not know.