ROS 2 Real-Time Working Group

ROS 2 Real-Time Working Group

Jul 11, 2021

CONTENTS

The Real-Time Working Group's mission is to advocate for and work on memory management, real-time pub/sub, real-time DDS, and tools that allow tracing, profiling and optimizing.

CHAPTER

ONE

GUIDES

1.1 Real Time Operating System Setup

1.1.1 Real-Time Linux

These are guides to build and configure the Linux kernel using PREEMPT_RT.

How to build your own Linux real-time kernel

Introduction

In this document, several guides are listed for users who want to build their own real-time kernel. That is, users who want to build a specific kernel version, a specific architecture or use a customized kernel configuration.

External guides

- How to setup Linux with PREEMPT_RT properly
  - Description: Official instructions from the Linux Foundation
  - https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/preemptrt_setup
- Building real-time Linux for ROS 2
  - Description: Community contributed step-by-step guide explaining how to build, configure and deploy a kernel with a RT_PREEMPT patch
  - https://docs.ros.org/en/rolling/Tutorials/Building-Realtime-rt_preempt-kernel-for-ROS-2.html

Build a Linux Real-Time kernel using docker

Introduction

This document explains how to build a real-time kernel using a docker container provided by the ROS Real-Time Working Group. The docker container comes with cross-compilation tools installed, and a ready-to-build RT kernel. This should be the preferred option for those users who simply want to use to cross-compile a new kernel.

Supported configuration

For the moment, the tool supports the following options:

- 5.4.0 kernel version and 5.4.86-rt48 patch
- cross-compilation for aarch64
- pre-configured kernel settings
- Raspberry Pi 4 Model B Rev 1.2 (more platforms will be added in the future)

Build and run docker container

For the local build:

```
$ git clone https://github.com/ros-realtime/rt-kernel-docker-builder
$ cd rt-kernel-docker-builder
$ docker build -t rtwg-image .
$ docker run -t -i rtwg-image bash
```

Kernel configuration

By default the kernel is configured with the following options:

- RT preempt real-time kernel
- Fixed operation frequency at 1.0 GHz
- CPU1, CPU2 and CPU3 tickless
- No CPU frequency scaling

This is configured automatically by setting the following options:

```
$ ./scripts/config -d CONFIG_PREEMPT \
$ ./scripts/config -e CONFIG_PREEMPT_RT \
$ ./scripts/config -d CONFIG_NO_HZ_IDLE \
$ ./scripts/config -e CONFIG_NO_HZ_FULL \
$ ./scripts/config -d CONFIG_HZ_250 \
$ ./scripts/config -e CONFIG_HZ_1000 \
$ ./scripts/config -d CONFIG_AUFS_FS \
```

which corresponds to the following

```
# Enable CONFIG_PREEMPT_RT
 -> General Setup
  -> Preemption Model (Fully Preemptible Kernel (Real-Time))
   (X) Fully Preemptible Kernel (Real-Time)

# Enable CONFIG_HIGH_RES_TIMERS
 -> General setup
  -> Timers subsystem
   [*] High Resolution Timer Support
```

(continues on next page)

(continued from previous page)

```
# Enable CONFIG_NO_HZ_FULL
 -> General setup
  -> Timers subsystem
   -> Timer tick handling (Full dynticks system (tickless))
    (X) Full dynticks system (tickless)

# Set CONFIG_HZ_1000
 -> Kernel Features
  -> Timer frequency (1000 HZ)
   (X) 1000 HZ

# Set CPU_FREQ_DEFAULT_GOV_PERFORMANCE [=y]
 -> CPU Power Management
  -> CPU Frequency scaling
   -> CPU Frequency scaling (CPU_FREQ [=y])
    -> Default CPUFreq governor (<choice> [=y])
     (X) performance

# Disable CONFIG_AUFS_FS, otherwise RT kernel build breaks
 x     -> File systems                                                 ␣
↪                                                        x
  x (1)   -> Miscellaneous filesystems (MISC_FILESYSTEMS [=y])
```

Todo:

- `CONFIG_CPU_FREQ=n` or `CONFIG_CPU_FREQ_DEFAULT_GOV_ONDEMAND=y`.
- `CONFIG_CPU_IDLE=n`: Disable transitions to low-power states

If you need to reconfigure it, run

```
$ cd linux-raspi-5.4.0/
$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- menuconfig
```

Kernel build

```
$ cd linux-raspi-5.4.0/
$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j `nproc` deb-pkg
```

You need 32GB free disk space to build it, it takes a while, and the results are located here:

```
user@3e9fd281ed2a:~/linux_build/linux-raspi-5.4.0$ ls -la ../*.deb
-rw-r--r-- 1 user user  11462528 Jun 17 09:46 ../linux-headers-5.4.114-rt57_5.4.114-rt57-
↪1_arm64.deb
-rw-r--r-- 1 user user 494790284 Jun 17 09:50 ../linux-image-5.4.114-rt57-dbg_5.4.114-
↪rt57-1_arm64.deb
-rw-r--r-- 1 user user  39756144 Jun 17 09:46 ../linux-image-5.4.114-rt57_5.4.114-rt57-1_
↪arm64.deb
-rw-r--r-- 1 user user   1055224 Jun 17 09:46 ../linux-libc-dev_5.4.114-rt57-1_arm64.deb
```

Deploy

Download and install Ubuntu 20.04 image

Follow these links to download and install Ubuntu 20.04. In the case of the Raspberry PI:

- https://ubuntu.com/download/raspberry-pi
- https://ubuntu.com/download/raspberry-pi/thank-you?version=20.04.2&architecture=server-arm64+raspi
- https://ubuntu.com/tutorials/create-an-ubuntu-image-for-a-raspberry-pi-on-ubuntu#2-on-your-ubuntu-machine

```
# initial username and password
ubuntu/ubuntu
```

Copy a new kernel to your system and install it

Todo: Add instructions explaining how to move the files to the Raspberry PI

Assumed you have already copied all *.deb packages to your `$HOME/ubuntu` directory

```
$ cd $HOME/ubuntu
$ sudo dpkg -i *.deb
```

Now it is necessary to adjust vmlinuz and initrd.img links. First, we locate the kernel that we are using:

```
ubuntu@ubuntu:/boot$ uname -a
Linux ubuntu 5.4.0-1028-raspi #31-Ubuntu SMP PREEMPT Wed Jan 20 11:30:45 UTC 2021␣
↪aarch64 aarch64 aarch64 GNU/Linux
```

We check the real-time kernel version that we installed, in this case is `5.4.114-rt57`:

```
ubuntu@ubuntu:~$ ls /boot/
System.map-5.4.0-1028-raspi  config-5.4.0-1028-raspi  dtb                   dtbs       ␣
↪initrd.img-5.4.0-1028-raspi  initrd.img.old           vmlinuz-5.4.0-1036-raspi
System.map-5.4.0-1036-raspi  config-5.4.0-1036-raspi  dtb-5.4.0-1036-raspi  firmware   ␣
↪initrd.img-5.4.0-1036-raspi  vmlinuz                 vmlinuz-5.4.114-rt57
System.map-5.4.114-rt57      config-5.4.114-rt57      dtb-5.4.114-rt57      initrd.img ␣
↪initrd.img-5.4.114-rt57      vmlinuz-5.4.0-1028-raspi  vmlinuz.old
```

Now we replace the old kernel with the new real-time one:

```
$ cd /boot
$ sudo ln -s -f vmlinuz-5.4.114-rt57 vmlinuz
$ sudo ln -s -f vmlinuz-5.4.0-1028-raspi vmlinuz.old
$ sudo ln -s -f initrd.img-5.4.114-rt57 initrd.img
$ sudo ln -s -f initrd.img-5.4.0-1028-raspi initrd.img.old
$ sudo cp vmlinuz firmware/vmlinuz
$ sudo cp vmlinuz firmware/vmlinuz.bak
$ sudo cp initrd.img firmware/initrd.img
$ sudo cp initrd.img firmware/initrd.img.bak

$ sudo reboot
```

Configure boot options

Inside the Raspberry PI, add the following at the end of the line in `/boot/firmware/cmdline.txt`:

```
$ sudo vim /boot/firmware/cmdline.txt
#  dwc_otg.fiq_fsm_enable=0 dwc_otg.fiq_enable=0 dwc_otg.nak_holdoff=0 dwg_otg.speed=1␣
↪rcu_nocbs=0 nohz_full=1-3 isolcpus=1-3 audit=0 watchdog=0 skew_tick=1
```

Here is an explanation of what each option will do:

- `dwc_otg.fiq_fsm_enable=0 dwc_otg.fiq_enable=0 dwc_otg.nak_holdoff=0`: solves an issue causing a high CPU usage from the USB driver (see https://www.osadl.org/Single-View.111+M5c03315dc57.0.html)
- `rcu_nocbs=0`: relocates RCU callbacks to kernel threads
- `nohz_full=1-3`: makes CPU1, CPU2 and CPU3 tickless
- `isolcpus=1-3`: isolates CPU1, CPU2 and CPU3. No process will be automatically scheduled to these CPUs.
- `audit=0`
- `watchdog=0`: disables the watchdog timer
- `skew_tick=1`

TODO: explain all the boot options used

For more information see:

- https://linux.enea.com/4.0/documentation/book-enea-linux-realtime-guide.pdf

Verify that eveything is correctly configured

After reboot you should see a new RT kernel installed

```
ubuntu@ubuntu:/boot$ uname -a
Linux ubuntu 5.4.114-rt57 #1 SMP PREEMPT_RT Thu Jun 17 09:21:41 UTC 2021 aarch64 aarch64␣
↪aarch64 GNU/Linux
```

Check that fiq is actually disabled:

```
ubuntu@ubuntu:~$ dmesg | grep -i fiq
[    0.000000] Kernel command line:  coherent_pool=1M 8250.nr_uarts=1 snd_bcm2835.enable_
↪compat_alsa=0 snd_bcm2835.enable_hdmi=1 bcm2708_fb.fbwidth=0 bcm2708_fb.fbheight=0␣
↪bcm2708_fb.fbswap=1 smsc95xx.macaddr=DC:A6:32:A7:32:00 vc_mem.mem_base=0x3ec00000 vc_
↪mem.mem_size=0x40000000  net.ifnames=0 dwc_otg.lpm_enable=0 console=ttyS0,115200␣
↪console=tty1 root=LABEL=writable rootfstype=ext4 elevator=deadline rootwait fixrtc dwc_
↪otg.fiq_fsm_enable=0 dwc_otg.fiq_enable=0 dwc_otg.nak_holdoff=0 dwg_otg.speed=1 rcu_
↪nocbs=0 nohz_full=1-3 isolcpus=1-3 quiet splash
[    1.771203] dwc_otg: FIQ disabled
[    1.771212] dwc_otg: FIQ split-transaction FSM disabled
```

Check that interrupts, except timers, are only handled by CPU0:

```
ubuntu@ubuntu:~$ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  1:          0          0          0          0     GICv2  25 Level     vgic
  3:     306043        106        104        103     GICv2  30 Level     arch_timer
```

(continues on next page)

(continued from previous page)

```
  4:          0          0          0          0     GICv2  27 Level     kvm guest vtimer
 11:      41860          0          0          0     GICv2  65 Level     fe00b880.mailbox
 15:          0          0          0          0     GICv2 150 Level     fe204000.spi
 16:       1143          0          0          0     GICv2 125 Level     ttyS0
 17:          0          0          0          0     GICv2 149 Level     fe804000.i2c
 20:          0          0          0          0     GICv2 114 Level     DMA IRQ
 22:          0          0          0          0     GICv2 116 Level     DMA IRQ
 23:        342          0          0          0     GICv2 117 Level     DMA IRQ
 27:         47          0          0          0     GICv2  66 Level     VCHIQ doorbell
 28:      20553          0          0          0     GICv2 158 Level     mmc1, mmc0
 29:          0          0          0          0     GICv2  48 Level     arm-pmu
 30:          0          0          0          0     GICv2  49 Level     arm-pmu
 31:          0          0          0          0     GICv2  50 Level     arm-pmu
 32:          0          0          0          0     GICv2  51 Level     arm-pmu
 34:        840          0          0          0     GICv2 189 Level     eth0
 35:        475          0          0          0     GICv2 190 Level     eth0
 41:          0          0          0          0     GICv2 175 Level     PCIe PME, aerdrv
 42:         45          0          0          0  BRCM STB PCIe MSI 524288 Edge      ␣
↪xhci_hcd
IPI0:        31         14         14         14       Rescheduling interrupts
IPI1:         0        277        277        278       Function call interrupts
IPI2:         0          0          0          0       CPU stop interrupts
IPI3:         0          0          0          0       CPU stop (for crash dump)␣
↪interrupts
IPI4:         0          0          0          0       Timer broadcast interrupts
IPI5:     21717          8          8          6       IRQ work interrupts
IPI6:         0          0          0          0       CPU wake-up interrupts
Err:          0
```

Check that soft-interrupts, except timers, are only handled by CPU0:

```
ubuntu@ubuntu:~$ cat /proc/softirqs
                    CPU0       CPU1       CPU2       CPU3
          HI:          2          0          0          0
       TIMER:     343845        105        103        103
      NET_TX:        165          0          0          0
      NET_RX:       1628          0          0          0
       BLOCK:       9192          0          0          0
    IRQ_POLL:          0          0          0          0
     TASKLET:       3728          0          0          0
       SCHED:          0          0          0          0
     HRTIMER:      60501          0          0          0
         RCU:          0          0          0          0
```

Check that all the CPU cores are operating at 1000MHz:

```
# reset cpufreq stat counters
ubuntu@ubuntu:~$ echo '1' | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/stats/reset
1
ubuntu@ubuntu:~$ cpufreq-info -s -m
600 MHz:0.00%, 700 MHz:0.00%, 800 MHz:0.00%, 900 MHz:0.00%, 1000 MHz:100.00%, 1.10 GHz:0.
↪00%, 1.20 GHz:0.00%, 1.30 GHz:0.00%, 1.40 GHz:0.00%, 1.50 GHz:0.00%
```

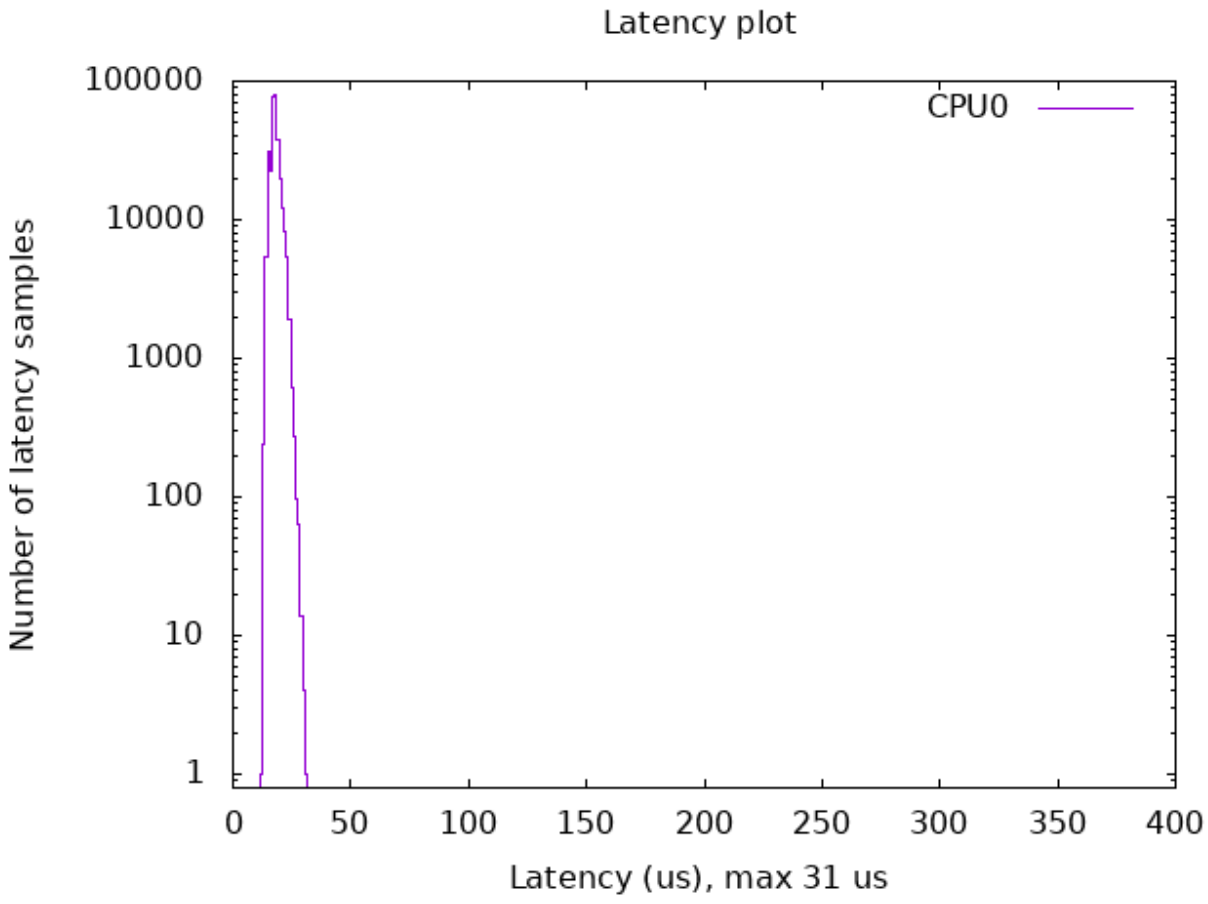
Benchmark

Finally, we can benchmark the real-time performance of the configured kernel with the platform we are using. A common benchmark is to measure the interrupt latency using a tool named cyclictest.

For example you run a latency test imposing CPU and I/O stress in the system and verifying that the latency test results in good performance.

```
$ taskset -c 0 stress -c 1 &
$ taskset -c 1 stress -c 1 &
$ taskset -c 2 stress -c 1 &
$ taskset -c 3 stress -c 1 &
$ taskset -c 0 stress -i 1 &
$ taskset -c 1 stress -i 1 &
$ taskset -c 2 stress -i 1 &
$ taskset -c 3 stress -i 1 &
$ taskset -c 0 cyclictest -p 90 -m -t1 -n -D 3h -i 200 -a 1 -h500 -q
```

In order to generate a latency plot you can use the OSADL script.



Real-time Raspberry PI images

Introduction

For those users who simply want to test ROS 2 real-time applications in a Raspberry Pi, some ready-to-use images are provided.

Raspberry PI images

TODO

1.1.2 Build VxWorks

Your can find more information about how to use ROS 2 with VxWorks here:

- https://github.com/Wind-River/vxworks7-layer-for-ros2

1.1.3 Build QNX

You can find the ROS 2 official QNX build instructions here: https://ros2-qnx-documentation.readthedocs.io/en/latest/

1.2 How to configure a RMW implementation

1.2.1 Fast-DDS

TODO

1.2.2 Cyclone-DDS

TODO

1.2.3 Connext-DDS

TODO

1.2.4 Iceoryx

TODO

1.3 How to configure a ROS2 real-time application

TODO

CHAPTER

TWO

TEST ENVIRONMENT

This document describes a test environment used for the ROS2 real-time tests

2.1 Hardware

Two hardware platforms:

- Hardware architecture
  - Intel x86_64
  - ARM v8
- Number of CPU cores => 4
- Amount of RAM => 8 GB
- Supports Ubuntu 20.04 LTS release

2.1.1 ARM

- Raspberry Pi 4 8 GB RAM, 4 CPU cores or similar

2.1.2 Intel

- any Intel PC with 8 GB RAM, 4 CPU cores
- UP squared 8 GB RAM, 4 CPU cores

2.2 Software

We use ROS2 Foxy release and Ubuntu 20.04 which is a Tier 1 platform as described in the Release information.

2.2.1 ROS2 foxy release

- Prebuilt Debian packages from Installing ROS 2 via Debian Packages

2.2.2 Latest Ubuntu 20.04 LTS (ISO image)

- Raspberry Pi4
- Intel UP squared
- Intel Ubuntu 20.04.2.0 LTS

2.2.3 Latest Stable PREEMPT_RT Kernel

- PREEMPT_RT Kernel is built using these instructions

CHAPTER

THREE

SUBPROJECT LIST

The following subprojects are owned by Real-Time Working Group:

- `rt-kernel-docker-builder`
  - Description: Build and setup RT kernel for the ROS2 testing
  - Repositories
    - https://github.com/ros-realtime/rt-kernel-docker-builder

CHAPTER

FOUR

RELATED PROJECTS

4.1 Performance measurements

- `performance_test`
  - Description: Tool to test the performance of pub/sub based communication frameworks
  - Repositories
    - https://gitlab.com/ApexAI/performance_test
- `ros2-performance`
  - Description: iRobot ROS2 performance evaluation framework
  - Repositories
    - https://github.com/irobot-ros/ros2-performance
- `buildfarm_perf_tests`
  - Description: Performance tests which run regularly on the ROS 2 buildfarm
  - Repositories
    - https://github.com/ros2/buildfarm_perf_tests
- `TwoWaysMeasurement`
  - Description: Tool to test the real-time performance in a ping-pong scenario
  - Repositories
    - https://github.com/y-okumura-isp/TwoWaysMeasurement
- `ros2_timer_latency_measurement`
  - Description: Tool to measure the accuracy of the ROS 2 timer
  - Repositories
    - https://github.com/hsgwa/ros2_timer_latency_measurement

4.2 Real-time utilities

- `realtime_support`
  - Description: Minimal real-time testing utility for measuring jitter and latency
    - `rttest`: Minimal tool for instrumenting and running tests for synchronous real-time systems
    - `tlsf_cpp`: C++ stdlib-compatible wrapper around `tlsf allocator` and ROS2 examples
  - Repositories
    - https://github.com/ros2/realtime_support
- `ros2_tracing`
  - Description: Tracing tools for ROS 2
  - Repositories
    - https://gitlab.com/ros-tracing/ros2_tracing
    - https://gitlab.com/ros-tracing/tracetools_analysis
- `osrf_testing_tools_cpp`
  - Description: This repository contains testing tools for C++, and is used in OSRF projects. The memory_tools API lets you intercept calls to dynamic memory calls like `malloc` and `free`, and provides some convenience functions for differentiating between expected and unexpected calls to dynamic memory functions.
  - Repositories:
    - https://github.com/osrf/osrf_testing_tools_cpp
- `apex_test_tools`
  - Description: The package Apex.OS Test Tools contains test helpers
  - Repositories:
    - https://gitlab.com/ApexAI/apex_test_tools
- `apex_containers`
  - Description: A collection of C++ containers suitable for real-time systems
  - Repositories:
    - https://gitlab.com/ApexAI/apex_containers
- `realtime_tools`
  - Description: Contains a set of tools that can be used from a hard real-time thread, without breaking the real-time behavior
  - Repositories:
    - https://github.com/ros-controls/realtime_tools/tree/foxy-devel
- `rclc`
  - Description: ROS Client Library for the C language
  - Repositories:
    - https://github.com/ros2/rclc
- `micro-ROS`

  - Description: ROS 2 based framework targeting embedded and deeply embedded robot components with extremely constrained computational resources
  - Repositories:
    * https://micro.ros.org/
    * https://github.com/micro-ROS

4.3 Real-time demos

- `pendulum_control`
  - Description: Real-time inverted pendulum control demo
  - Repositories
    * https://github.com/ros2/demos/tree/master/pendulum_control
    * https://docs.ros.org/en/foxy/Tutorials/Real-Time-Programming.html
- `pendulum`
  - Description: Inverted pendulum demo inspired by `pendulum_control`
  - Repositories
    * https://github.com/ros2-realtime-demo/pendulum
- `e2e_demo`
  - Description: End-to-end latency demo
  - Repositories
    * https://github.com/hsgwa/e2e_demo

CHAPTER

FIVE

RESOURCES

This document contains a compilation of ROS and real-time related documents, articles, and discussions.

5.1 ROS 2 design

- Introduction to Real-time Systems
- Proposal for Implementation of Real-time Systems in ROS 2

5.2 Tutorials / Guides

- Real-time programming in ROS 2
- Building real-time Linux for ROS 2

5.3 ROSCon

5.3.1 ROSCon 2015

- Real-time Performance in ROS 2 Slides Video

5.3.2 ROSCon 2016

- Evaluating the resilience of ROS2 communication layer Slides Video

5.3.3 ROSCon 2017

- Determinism in ROS Slides Video

5.3.4 ROSCon 2018

- Middleware Performance Testing Slides Video
- ROS 2 on Autonomous Vehicles Slides Video
- ROSCon 2018: Mixed Real-Time Criticality with ROS2 - the Callback-group-level Executor slides video

5.3.5 ROSCon 2019

- ROS 2 ON VXWORKS slides video
- ROS2 Real-Time Behavior: Static Memory Allocation video
- Doing Real-Time with ROS 2: Capabilities and Challenges

5.4 Articles

- Exploring the performance of ROS2
- Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications
- Response-Time Analysis of ROS 2 Processing Chains under Reservation-Based Scheduling
- Latency Overhead of ROS2 for Modular Time-Critical Systems
- Exploring Real-Time Executor on ROS 2
- Distributed and Synchronized Setup towards Real-Time Robotic Control using ROS2 on Linux
- Jan Staschulat, Ingo Lütkebohle, Ralph Lange. The rclc Executor: Domain-specific deterministic scheduling mechanisms for ROS applications on microcontrollers, EMSOFT 2020.
- Jan Staschulat, Ralph Lange, Dakshina Narahari Dasari. Budget-based real-time Executor for Micro-ROS. CoRR arXiv:2105.05590, May 2021
- L. Puck et al. Distributed and Synchronized Setup towards Real-Time Robotic Control using ROS2 on Linux

CHAPTER

SIX

CONTACT

6.1 Meetings

- Regular WG Meeting: every other Tuesday at 7 AM Pacific time, see the ROS Events calendar
- To receive meeting invitations, join `ros-real-time-working-group-invites`
- Meeting notes are kept under ROS 2 Real-time Working Group Agenda
- Meetings are recorded and available in ROS 2 Real-time Working Group Agenda.
- Meetings are open to the public, and anyone is welcome to join

6.2 Communication Channels

- ROS discourse tag `wg-real-time`
- Chat in the Real-time WG Room on Matrix

CHAPTER

SEVEN

HOW TO CONTRIBUTE

7.1 Standards for subprojects

Subprojects must meet the following criteria (and the WG agrees to maintain them upon adoption).

- Build passes against ROS 2 master
- The ROS 2 standard linter set is enabled and adhered to
- If packages are part of nightly builds on the ROS build farm, there are no reported warnings or test failures
- Quality builds are green (address sanitizer, thread sanitizer, clang thread safety analysis)
- Test suite passes
- Code coverage is measured, and non-decreasing level is enforced in PRs
- Issues and pull requests receive prompt responses
- Releases go out regularly when bugfixes or new features are introduced
- The backlog is maintained, avoiding longstanding stale issues

7.2 Adding new subprojects

To request introduction of a new subproject, add a list item to the "Subprojects" section and open a Pull Request to this repository, following the default Pull Request Template to populate the text of the PR.

PR will be merged on unanimous approval from Approvers.

7.3 Subproject changes

Modify the relevant list item in the "Subprojects" section and open a Pull Request to this repository, following the default Pull Request Template to populate the text of the PR.

PR will be merged on unanimous approval from Approvers.

7.4 Deprecating subprojects

Projects cease to be useful, or the WG can decide it is no longer in their interest to maintain. We do not commit to maintaining every subproject in perpetuity.

To suggest removal of a subproject, remove the relevant list item in the "Subprojects" section and open a Pull Request in this repository, following instructions in the Pull Request Template to populate the text of the PR.

PR will be merged on unanimous approval from Approvers.

If the repositories of the subproject are under the WG's GitHub organization, they will be transferred out of the organization or deleted at this time.

CHAPTER

EIGHT

ROADMAP

This page describes planned work for the ROS 2 Real-Time Working Group.