Real-time working group Documentation

Lander Usategui

Apr 28, 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

Build a Linux Real-Time kernel using docker

Introduction

Ubuntu 20.04 x86_64 docker container is used to cross-compile a new kernel. There is a Dockerfile which can be used for that purpose. If you want to build it using gitpod you need to run https://gitpod.io/#https://github.com/ros-realtime/rt-kernel-docker-builder. It will spawn a docker container automatically for you.

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
```

Setup a build environment

Container comes with cross-compilation tools installed, and a ready-to-build RT kernel:

- ARMv8 cross-compilation tools
- Linux source build dependencies
- Linux source buildinfo, i.e. from where config is copied
- Ubuntu RPI4 linux source installed under ~/linux_build
- RT kernel patch downloaded and applied - the nearest to the recent RPI4 Ubuntu kernel

Kernel configuration

Additionally RT kernel configured as

```
$ ./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

# 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])
```

If you need to reconfigure it, run

```
$ make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- menuconfig
```

Kernel build

```
$ 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:

```
gitpod ~/linux_build/linux-raspi-5.4.0 $ ls -la ../*.deb
-rw-r--r-- 1 gitpod gitpod  11278252 Nov 29 14:01 ../linux-headers-5.4.86-rt48_5.4.86-
↪rt48-1_arm64.deb
-rw-r--r-- 1 gitpod gitpod 486149956 Nov 29 14:04 ../linux-image-5.4.86-rt48-dbg_5.4.
↪86-rt48-1_arm64.deb
-rw-r--r-- 1 gitpod gitpod  38504756 Nov 29 14:01 ../linux-image-5.4.86-rt48_5.4.86-
↪rt48-1_arm64.deb
-rw-r--r-- 1 gitpod gitpod   1054624 Nov 29 14:01 ../linux-libc-dev_5.4.86-rt48-1_
↪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&architecture=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
```

Update your system

After that you need to connect to the Internet and update your system

```
$ sudo apt-get update && apt-get upgrade
```

Copy a new kernel to your system and install it

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. There is an extra step in compare to the x86_64 install (why is that?)

```
$ cd /boot
$ sudo ln -s -f vmlinuz-5.4.86-rt48 vmlinuz
$ sudo ln -s -f vmlinuz-5.4.0-1029-raspi vmlinuz.old
$ sudo ln -s -f initrd.img-5.4.86-rt48 initrd.img
$ sudo ln -s -f initrd.img-5.4.0-1029-raspi initrd.img.old
```

(continues on next page)

(continued from previous page)

```
$ 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
```

Verify kernel version

After reboot you should see a new RT kernel installed

```
ubuntu@ubuntu:~$ uname -a
Linux ubuntu 5.4.86-rt48 #1 SMP PREEMPT_RT Sun Nov 15 22:44:33 UTC 2020 aarch64␣
↪aarch64 aarch64 GNU/Linux
```

1.1.2 VXworks

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 QNX

Build QNX (WIP)

The QNX instructions are currently work in progress. You can find a temporary guide here: https://github.com/asobhy-qnx/ros2_documentation/blob/master/source/Installation/Rolling/QNX-Development-Setup.rst

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

1.3.1 How to configure a ROS2 real-time application

TODO

CHAPTER

TWO

REAL-TIME BUILDFARM

2.1 Test environment

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

2.1.1 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

ARM

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

Intel

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

2.1.2 Software

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

ROS2 foxy release

- Prebuilt Debian packages from Installing ROS 2 via Debian Packages

Latest Ubuntu 20.04 LTS (ISO image)

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

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: rttest is a 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 realtime thread, without breaking the realtime behavior
  - Repositories:
    - https://github.com/ros-controls/realtime_tools/tree/foxy-devel

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, dicussions.

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 realtime 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
- The rclc Executor: Domain-specific deterministic scheduling mechanisms for ROS applications on microcontrollers: work-in-progress

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.