ROS 2 Executor: How to make it efficient, real-time and deterministic?

Callback-group-level Executor

Dr. Ralph Lange
Bosch Corporate Research
Let’s prioritize the critical path ...
Critical Path in the Reference System
Distribution to Executors

Executor 1
- Front Lidar Driver
- Front Points Transformer
- Euclidean Cluster Settings
- Intersection Output

Executor 2
- Rear Lidar Driver
- Rear Points Transformer
- Point Cloud Fusion
- Ray Ground Filter
- Euclidean Cluster Detector
- Object Collision Estimator

Executor 3
- Point Cloud Map
- Point Cloud Map Loader
- Voxel Grid DownSampler
- NDT Localizer
- MPC Controller
- Vehicle Interface

Non-critical nodes are assigned to Executor 4
- Visualizer
- Lanelet2 Map
- Lanelet2 Global Planner
- Lanelet2 Map Loader
- Parking Planner
- Behavior Planner
- Lane Planner
- Vehicle DBW System
Linux Schedulers and Priorities

SCHED_FIFO and SCHED_RR
- Defined in POSIX standard
- Preemption according to priority levels
- FIFO or round-robin (configurable quantums)

SCHED_OTHER
- Completely Fair Scheduler (CFS) by Ingo Molnár
- Since Linux 2.6.23
- Actually, three policies SCHED_NORMAL (aka SCHED_OTHER), SCHED_BATCH, and SCHED_IDLE

Real-time throttling:
- Non-RT schedulers get at least 50 ms per second to avoid that system hangs
## Linux Schedulers and Priorities

```bash
ps ax --format uname,pid,cmd,pri,rtprio,nice
```
Prioritization of Executors

▶ Prioritization with nice levels in SCHED_OTHER

SingleThreadedExecutor executor1;
executor1.add_node(front_lidar_driver_node);
[... ] // Add more nodes.

auto executor1_thread = std::thread(
    [&]() {
        int nice = -5; // -20 to 19
        setpriority(PRIIO_PROCESS, gettid(), nice);
        executor1.spin();
    });

[... ] // Create other threads
executor1_thread.join();

▶ Real-time prioritization with SCHED_FIFO

SingleThreadedExecutor executor1;
executor1.add_node(front_lidar_driver_node);
[... ] // Add more nodes.

auto executor1_thread = std::thread(
    [&]() {
        executor1.spin();
    });

auto handle1 = executor1_thread.native_handle();
sched_param params; int policy;
pthread_getschedparam(handle1, &policy, &params);
params.sched_priority = 20; // 1 to 99
pthread_setschedparam(handle1, SCHED_FIFO, &params);

[... ] // Create other threads
executor1_thread.join();
Prioritization of Executors

- Prioritization with nice levels in SCHED_OTHER

```cpp
SingleThreadedExecutor executor1;
executor1.add_node(front_lidar_driver_node);
[...] // Add more nodes.

auto executor1_thread = std::thread(
    [&](){
        int nice = -5;  // -20 to 19
        setpriority(PRIO_PROCESS, gettid(), nice);
        executor1.spin();
    });

[...] // Create other threads
executor1_thread.join();
```

- Real-time prioritization with SCHED_FIFO

```cpp
SingleThreadedExecutor executor1;
executor1.add_node(front_lidar_driver_node);
[...] // Add more nodes.

auto executor1_thread = std::thread(
    [&](){
        executor1.spin();
    });

auto handle1 = executor1_thread.native_handle();
sched_param params;  int policy;
pthread_getschedparam(handle1, &policy, &params);
params.sched_priority = 20;  // 1 to 99
pthread_setschedparam(handle1, SCHED_FIFO, &params);
[...] // Create other threads
executor1_thread.join();
```

- Not immediately possible for MultiThreadedExecutor
- See PiCAS talk by Hyunjong for a nice API
Distribution to Executors

Executor 1 (nice: -5)
- Front Lidar Driver
- Front Points Transformer
- Euclidean Cluster Settings

Executor 2 (nice: -5)
- Rear Lidar Driver
- Rear Points Transformer
- Point Cloud Fusion

Executor 3 (nice: -5)
- Intersection Output
- Ray Ground Filter
- Euclidean Cluster Detector
- Object Collision Estimator

Non-critical nodes are assigned to Executor 4 (nice: +5)

- Point Cloud Map
- Point Cloud Map Loader
- Voxel Grid Downsample
- NDT Localizer
- Behavior Planner
- Lane Planner
- Parking Planner
- Lanel2 Global Planner
- Lanel2 Map Loader
- MPC Controller
- Vehicle Interface
- Vehicle DBW System

© Robert Bosch GmbH 2021. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Results for the Reference System (4 CPUs)
Results for the Reference System (2 CPUs)
Callback-group-level Executor
Callback-group-level Executor is NOT another Executor
Motivation and Idea

Motion control node

- Parameter services: ParameterService::*
- Lifecycle services: LifecycleNode::*
- Goal subscription: store_new_goal
- Sensor subscription: detect_sensor_timeout
- Do control step: do_control_step

Motion command
Motivation and Idea

Motion control node

- ParameterService::*
- LifecycleNode::*
- store_new_goal
- detect_sensor_timeout
- do_control_step

Executor 1:
- Low prio thread

Executor 2:
- High prio thread

parameter services
lifecycle services
goal subscription
sensor subscription
motion command
Executor API

**base class Executor**

- add_node
- remove_node

**base class Executor**

- add_callback_group
- remove_callback_group
- add_node
- remove_node
- get_all_callback_groups
- get_manually_added_callback_groups
- get_automatically_added_callback_groups
  - from_nodes
Executor API (cont’d)

Many thanks to Pedro Pena (peterpena) and William Woodall (wjwwood) who brought the callback-group-executor prototype mainline!

Executor

- SingleThreaded Executor
- MultiThreaded Executor
- StaticSingleThreaded Executor
- Events Executor

See today’s talk by Alberto Soragna
Critical Path in the Reference System
Distribution to Executors

Executor 1 (nice: -5)
- Front Lidar Driver
- Front Points Transformer
- Euclidean Cluster Settings
- Intersection Output

Executor 2 (nice: -5)
- Rear Lidar Driver
- Rear Points Transformer
- Point Cloud Fusion

Executor 3 (nice: -5)
- Ray Ground Filter
- Euclidean Cluster Detector
- Object Collision Estimator

Non-critical nodes and callback groups are assigned to Executor 4 (nice: +5)
- Point Cloud Map
- Visualizer
- Lanelet2 Map
- Lanelet2 Global Planner
- Lanelet2 Map Loader
- Parking Planner
- Voxel Grid Downsample
- NDT Localizer
- Behavior Planner
- Lane Planner
- MPC Controller
- Vehicle Interface
- Vehicle DBW System
Results for the Reference System (4 CPUs)
Results for the Reference System (2 CPUs)
Demo Package
examples_rclcpp_cbغ_executor
Package examples_rclcpp_cbg_executor

Ping Node

- Counter msg every `ping_period` secs
- Measuring receive rate and latency

Pong Node

- /high_ping
- /high_pong
- Burn CPU cycles for `high_busyloop` secs

Source code at https://github.com/ros2/examples/

With SCHED_FIFO and core pinning:
- Executor with high prio thread
- Executor with low prio thread
Package examples_rclcpp_cbg_executor

Source code at https://github.com/ros2/examples/

```
high_busyloop = 0.01 s
low_busyloop = 0.04 s
```
Looking forward to your questions!

Dr. Ralph Lange
Bosch Corporate Research
ralph.lange@de.bosch.com
github.com/ralph-lange