Advanced Execution Management with ROS 2

Dr. Ralph Lange
Bosch Corporate Research
int main(int argc, char* argv[]) {
    ros::init(argc, argv, "my_node");
    ros::NodeHandle nh;
    // Init some stuff
    ros::spin();
    return 0;
}
Execution Management in ROS 1

User code

roscpp
TCPROS, UDPROS

rospy
TCPROS, UDPROS

roslib

TCP, UDP
Execution Management in ROS 1

int main(int argc, char* argv[]) {
    rclcpp::init(argc, argv);
    rclcpp::Node::SharedPtr node = ... 
    rclcpp::spin(node);
    return 0;
}
$ ros2 component standalone demo_nodes_cpp demo_nodes_cpp::Listener

[INFO] [1607681830.972658378] [standalone_container_bc0e6d7a7a2e]: Load Library: /opt/ros/foxy/lib/libtopics_library.so
[INFO] [1607681830.974177416] [standalone_container_bc0e6d7a7a2e]: Found class: rclcpp_components::NodeFactoryTemplate<demo_nodes_cpp::Listener>
[INFO] [1607681830.974225889] [standalone_container_bc0e6d7a7a2e]: Instantiate class: rclcpp_components::NodeFactoryTemplate<demo_nodes_cpp::Listener>

[INFO] [160768184.120730739] [listener]: I heard: [Hello World: 1]
[INFO] [160768188.111974007] [listener]: I heard: [Hello World: 2]
[INFO] [160768188.122591272] [listener]: I heard: [Hello World: 3]
[INFO] [160768188.114233723] [listener]: I heard: [Hello World: 4]
[INFO] [160768188.112173517] [listener]: I heard: [Hello World: 5]
[INFO] [160768188.119666995] [listener]: I heard: [Hello World: 6]
Executor

\[ \ldots \]

```cpp
rclcpp::executors::SingleThreadedExecutor executor;
exector.add_node(node);
exector.spin();
\ldots
```
Agenda

- Objectives behind Executor design
- Default scheduling semantics – and its issues

- Static Executor
- Callback-group-level Executor

- Determinism – and particularly FIFO ordering

- rclc Executor (micro-ROS)
Executor Design

- rclcpp
- rclpy
- ... (User code)

- rcl – ROS Client Support Lib

- rmw – middleware interface

- rmw adapter

- FastDDS, Cyclone, Connext, ...

© Robert Bosch GmbH 2020. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Executor Design

- onGoal
- nextCmd
- processOdom

Executor

1) wait
2) take
3) execute

User code

rclcpp
rclpy
...

rmw – middleware interface

rmw adapter

FastDDS, Cyclone, Connext, ...

rcl – ROS Client Support Lib

User code
Executor Design

Design objectives
- Avoid additional queue in client library
- Utilize DDS QoS mechanisms
  - Lifespan, history, priorities, ...

⚠️ Decision on processing order is distributed to middleware and client lib

```
User code

rclcpp  rclpy ...

rcl – ROS Client Support Lib

rmw – middleware interface

rmw adapter

FastDDS, Cyclone, Connext, ...

```

/goal  /cmd  /odom
Scheduling Semantics

Executor

timer ready?

<table>
<thead>
<tr>
<th>yes</th>
<th>yes</th>
<th>yes</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

collect_entities()

wait in middleware

/pgoal 0 /p_cmd /p_odom
Scheduling Semantics

Non-preemptive priority + round-robin

- timer ready?
  - no
    - topic in wait-set?
      - no
        - service in wait-set?
          - no
            - service reply in wait-set?
              - no
                - collect_entities()
                  - yes
                    - wait in middleware

- yes
  - Execute callback
  - Clear in wait-set

- Take message
Non-preemptive priority + round-robin

No FIFO processing in case of congestions!

Discussion

Requirements

- End-to-end latency guarantees
- Support for mixed real-time criticality
- Parallelization
- Determinism

On-going works

- Runtime overhead by layered design
  - Costly wait-set operations
- Mapping to OS scheduling mechanisms
- FIFO ordering (by message timestamps)
Multi-Threaded Executor

1) wait
2a) take
2b) take
3a) execute
3b) execute

User code
rclcpp
rclpy
...
rmw adapter
FastDDS, Cyclone, Connext, ...
rmw – middleware interface
rcl – ROS Client Support Lib

/multiThreadedExecutor
/onGoal
/nextCmd
/processOdom

/doc
/cmd
/odom

2a) take
1) wait
2b) take
auto secondGroup = create_callback_group(type);

where type is rclcpp::CallbackGroupType::MutuallyExclusive
or rclcpp::CallbackGroupType::Reentrant
rclcpp::SubscriptionOptionsWithAllocator<> options;
options.callback_group = secondGroup;

mySub = create_subscription<..>("/odom",
   rclcpp::SensorDataQoS(), processOdom, options);
...
myTimer = create_wall_timer(100ms, myCallback, secondGroup);
Static Single-Threaded Executor

StaticSingleThreadedExecutor

- onGoal
- nextCmd
- processOdom

`/goal` → 1
`/cmd` → 0
`/odom` → 1

flowchart

- collect_entities()
- timer ready?
- topic in wait-set?
- service in wait-set?
- service reply in wait-set?
- wait in middleware

P_Goal
P_Cmd

Take message
Execute callback
Clear in wait-set

© Robert Bosch GmbH 2020. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Static Single-Threaded Executor

```cpp
StaticSingleThreadedExecutor executor;
executor.add_node(node);

void nextCmd(Cmd msg)
{
    if (msg == "activate_process_odom")
    {
        processOdomSub_ = create_subscription(…);
    }
}
```
Callback-group-level Executor

is NOT another Executor
Callback-group-level Executor

Support mixed real-time criticality in a node

- Refines interface of Executor to callback groups
- Prototype presented by me at ROSCon 2018
- Recently brought mainline by Pedro Pena and William Woodall (many thanks!)
- Implemented for all Executors in rclcpp now
- Available in Rolling release
The cbg_executor_demo Package

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

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

- /low_ping
- /low_pong
- Burn CPU cycles for low_busyloop secs
- Executor with low prio thread

Source code at https://github.com/boschresearch/ros2_demos
The cbg_executor_demo Package

Source code at https://github.com/boschresearch/ros2_demos

high_busyloop = 0.01\,s
low_busyloop = 0.04\,s
... but no solution for determinism or at least FIFO ordering
Design Revisited

Decision on processing order is distributed to middleware and client lib

Executor

User code

rclcpp

rclpy

... rcl – ROS Client Support Lib

rmw – middleware interface

rmw adapter

FastDDS, Cyclone, Connext, ...

/goal

/cmd

/odom

onGoal nextCmd processOdom
Design Revisited

Ideas:

1. Decide completely in middleware
   - Lack of application knowledge

2. Additional queue in client library
   - Thwarts middleware QoS

3. Comprehensive view on middleware
   - Expensive synchronization

Many subtle technical issues:
- Memory management
- Integration of timers
- Access to DDS metadata
Message Info (since Foxy)

Ideas:

1. Decide completely in middleware
   – Lack of application knowledge

2. Additional queue in client library
   – Thwarts middleware QoS

3. Comprehensive view on middleware
   – Expensive synchronization

Many subtle technical issues:

- Memory management
- Integration of timers
- Access to DDS metadata
EventsExecutor (Proof of Concept)

Thread at https://discourse.ros.org/t/ros2-middleware-change-proposal/

- Improved performance
- FIFO ordering
- Possible to use DDS listeners
- Event queue or work queue?
rclc Executor for micro-ROS
Typical Execution Patterns

- Control loops

- Data fusion

- Prioritized paths
Key Concepts of rcllc Executor

- Individual registration of each callback
  - Not uncommon in deeply embedded software

- User-defined processing sequence

- Custom trigger conditions

- Optional: LET semantics

Source code at https://github.com/ros2/rcllc/
Conclusions on Execution Management in ROS 2

- Very different semantics compared to ROS 1
  - No FIFO ordering in case of congestions

- Decision on processing order is distributed to middleware and client library
  - Key questions: Determinism? Integration with middleware QoS?
  - On-going discussion – join middleware and real-time working group

- Several new concepts available in Foxy and Rolling
Looking forward to your questions!

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