Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Петрозаводский государственный университет
Институт математики и информационных технологий

09.03.04 - Программная инженерия

Отчет о прохождении производственной практики

Выполнила студентка группы 22207:
Квист Татьяна Денисовна

Направление подготовки:
Системное и прикладное программное обеспечение

Место прохождения практики:
Кафедра информатики и математического обеспечения, Центр Искусственного Интеллекта

Сроки прохождения практики:
30.05-09.06

Руководитель практики:
к.т.н., доцент
Богоявленская Ольга Юрьевна

Оценка _____

Дата _____

Петрозаводск — 2022

Содержание

Введение

Цель практики: реализовать пример использования механизма publisher/subscriber в ROS.

Задачи учебной практики:

1. познакомиться с ROS;
2. развернуть систему для работы с ROS;
3. настроить окружение для работы;
4. реализовать пример использования механизма publisher/subscriber;
5. сравнить реализацию механизма в ROS 2 и ROS 1.

1 ROS

ROS (Robot Operating System) или операционная система для роботов — это экосистема для программирования роботов, предоставляющая функциональность для распределённой работы.

ROS обеспечивает стандартные службы операционной системы, такие как: аппаратную абстракцию, низкоуровневый контроль устройств, реализацию часто используемых функций, передачу сообщений между процессами, и управление пакетами. ROS основан на архитектуре графов, где обработка данных происходит в узлах, которые могут получать и передавать сообщения между собой. Библиотека ориентирована на Unix-подобные системы (Ubuntu Linux включен в список «поддерживаемых», в то время как другие варианты, такие как Fedora и Mac OS X, считаются «экспериментальными»).

1.1 Принципы разработки на ROS

ROS сосредоточена на максимизации повторного использования кода при разработке. Основные характеристики позволяющие это реализовать:

- Распределенные процессы: структура ROS создана в виде минимальных единиц исполняемых процессов (нодов, узлов), и каждый процесс выполняется изолированно. Взаимодействие разных узлов происходит только на уровне обмена сообщениями.

- Управление пакетами: несколько процессов, имеющих общую задачу объединяются в пакеты. Управление пакетами подразумевает набор утилит, позволяющих автоматически скачивать, устанавливать и удалять пакеты. Пакетный менеджер гарантирует работоспособность и целостность установленных пакетов.

- Публичные репозитории и документация: каждый доступный пакет публикуется в публичном репозитории. Документация пакетов, публикуется в единой системе, которая упрощает поиск необходимых пакетов.

- Единое API: при разработке программы, использующей ROS, получается простое и легко встраиваемое API. В примерах программ использование API не сильно отличается от языка (C ++ или Python). При этом нет разницы при использовании API на каком языке была написана программа.

- Поддержка различных языков программирования: программа ROS предоставляет клиентские библиотеки для поддержки различных языков программирования. Наиболее популярны Python, C ++, а также такие языки, как Lisp, Java, C#, Lua и Ruby.

1.2 Основные термины

Для того, чтобы приступить к настройке системы, нужно сначала понять что и как работает.

- Узел — программная единица системы на ROS. Все узлы системы работают независимо и параллельно.

- Пакет — набор узлов, объединенных по общему назначению.

- Тема — способ асинхронного обмена сообщениями между узлами.

- Сервис — клиент-серверный вариант взаимодействия узлов.

- Действие — способ выполнения продолжительных действий с переодическим обновлением результата (объединение сервисов и тем).

1.3 Применение

2 Развёртывание

Так как Ubuntu Linux включен в список "поддерживаемых" , то для разработки я выбрала именно его.

2.1 Docker

Для того, чтобы развернуть Ubuntu я решила использовать docker. Docker – это программная платформа для быстрой разработки, тестирования и развертывания приложений. Docker упаковывает ПО в стандартизованные блоки, которые называются контейнерами. Каждый контейнер включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения.

Я написала Dockerfile для установки образа ROS Galactic, обновления системы, установки необходимых пакетов для подключения к системе и подгрузке данных из Интернета и командной оболочки fish. Я собрала образ docker-контейнера и запустила. Затем с помощью docker exec я зашла в командную оболочку контейнера для настройки.

2.2 Настройка окружения

Для того, чтобы можно было работать с ROS, необходимо настроить окружение. Как минимум переменные окружения:

- ROS_VERSION=2
- ROS_PYTHON_VERSION=3
- ROS_DISTRO=galactic
- ROS_DOMAIN_ID=0

Также я создала директорию dev_ws, в котором будет разработан пример использования механизма publisher/subscriber.

2.3 Подключение к контейнеру с графическим интерфейсом

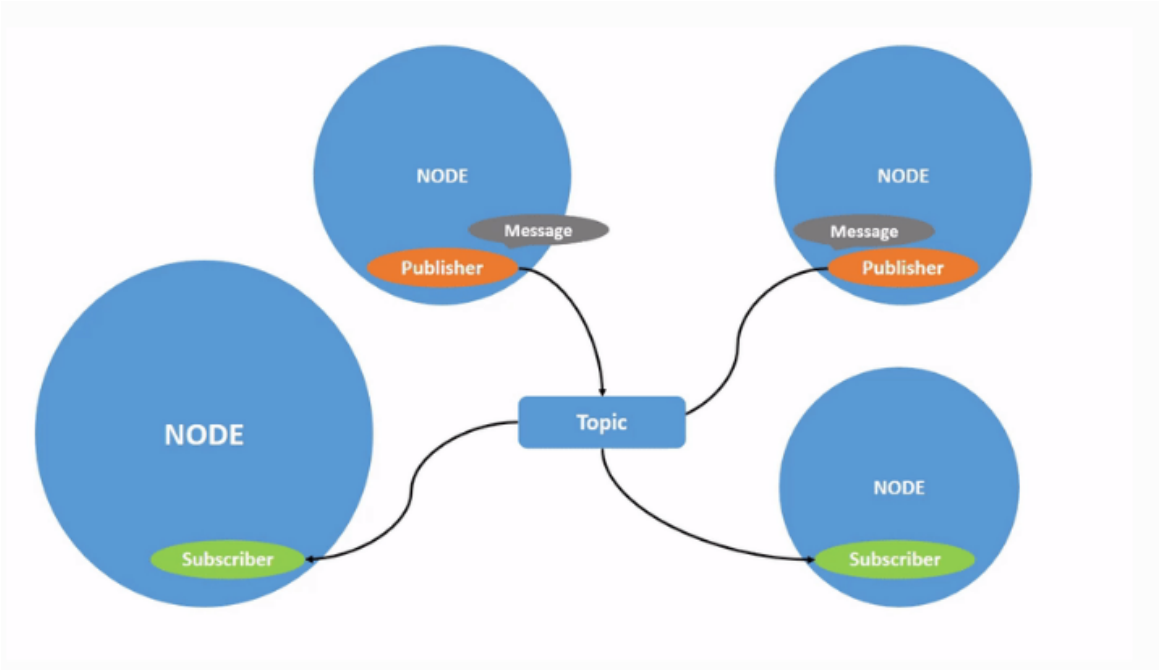
Обычно я подключаюсь к docker-контейнеру через консоль по ssh, поэтому нужно было поднять ssh-сервер внутри контейнера с помощью утилиты openssh-server и разрешить

доступ к графическому интерфейсу (дополнительно, в качестве ключа при подключении указать -X).

3 Механизм publisher/subscriber

3.1 Принцип работы

Отправители сообщений, именуемые издателями (англ. publishers), напрямую не привязаны программным кодом отправки сообщений к подписчикам (англ. subscribers). Вместо этого сообщения делятся на классы и не содержат сведений о своих подписчиках, если таковые есть. Аналогичным образом подписчики имеют дело с одним или несколькими классами сообщений, абстрагируясь от конкретных издателей.



3.2 Преимущества и недостатки

К преимуществам механизма можно отнести:

- Издатели не зависят от конкретных классов подписчиков и наоборот.
- Можно подписывать и отписывать получателей на лету.
- Реализует принцип открытости/закрытости.

И явным недостатком этого механизма является то, что подписчики оповещаются в случайном порядке.

3.3 Реализация

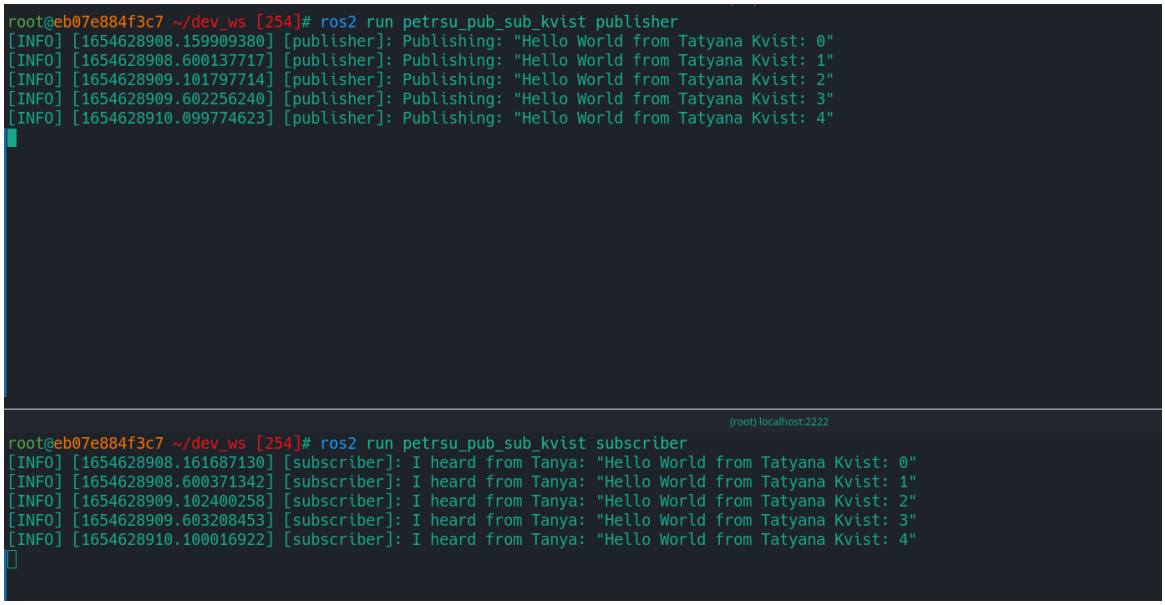
Я создала пакет petrsu_pub_sub_kvist. При создании пакета я указала в файле package.xml создателя (себя), лицензию, под которой будет распространяться пакет, зависимости как для тестов, так и для основной программы.

В этом пакете реализовала пример механизма publisher/subscriber на Python 3, то есть два основных файла: publisher.py и subscriber.py. В каждом мне понадобилась клиентская библиотека ROS (rclpy), класс Node для наследования и библиотека со стандартными сообщениями, из которой я взяла сообщения типа String.

Для того, чтобы программа понимала откуда нужно начинать работу в файле setup.py я установила точки входа (от английского entrypoints) в формате:

"название_пакета.основной_файл_узла:функция_начала"

И в конце я добавила пути для запуска для ros2 в setup.cfg.



Код программ можно посмотреть в приложениях.

3.4 Отличия в реализации на ROS 1

Одно из основных отличий, которое можно сразу отметить: разные версии языка Python. В ROS 1 используется python2, а в ROS 2 — python3.

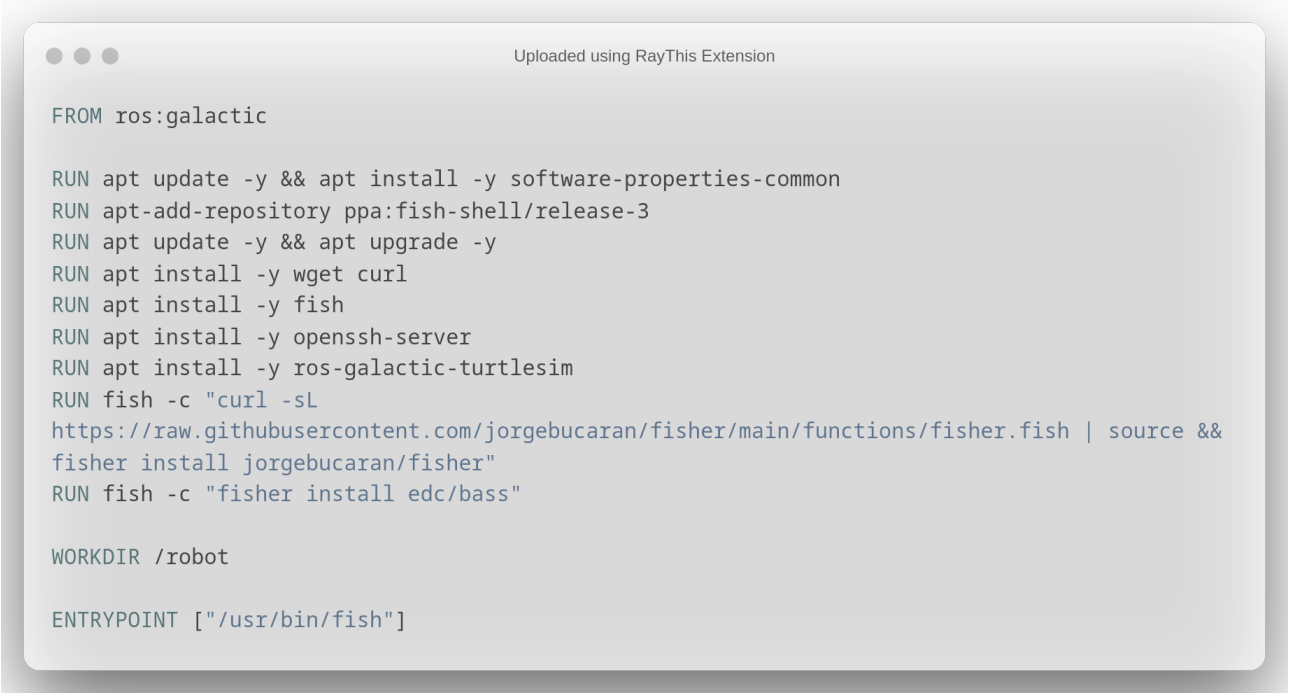
Также есть различие в реализации издателя. В ROS 1 для отправления данных используется цикл while, условие которого, что издатель в работе. И все данные отправляются внутри этого цикла. После отправки данных издатель "засыпает" на секунду. В ROS 2 в

издателе используется таймер и callback-функции. Создаётся таймер с указанным промежутком времени и функцией, которую необходимо вызвать по истечении времени.

4 Заключение

В ходе практики был изучен принцип работы ROS, docker и механизм publisher/subscriber. Развернута и настроена система для разработки на ROS. Разработан пример использования механизма publisher/subscriber. Проведён анализ и выявлены отличия реализаций на разных версиях ROS.

Приложение 1. Dockerfile



Приложение 2. Издатель


```python
import rclpy
from rclpy.node import Node # для подключения класса Node

from std_msgs.msg import String # для сообщений типа string

# Класс издателя, наследуемый от Node
class Publisher(Node):

    def __init__(self, node_name: str, topic_name: str):
        # Создание узла
        super().__init__(node_name)
        # Создание издателя для темы
        self.publisher_ = self.create_publisher(String, topic_name, 10)

        # Создание таймера с периодом и callback-функцией
        timer_period = 0.5
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0 # индексация

    def timer_callback(self):
        # Создаём новое сообщение типа строки
        msg = String()
        msg.data = f'Hello World from Tatyana Kvist: {self.i}'
        # Публикуем в тему
        self.publisher_.publish(msg)
        # Записываем в логи содержимое сообщения
        self.get_logger().info(f'Publishing: "{msg.data}"')
        self.i += 1


def main(args=None):
    rclpy.init(args=args) # инициализируем клиентскую библиотеку ROS

    node_name, topic_name = "publisher", "kvist_topic"
    publisher = Publisher(node_name, topic_name) # создаём объект типа издателя

    rclpy.spin(publisher) # запуск издателя

    publisher.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

Приложение 3. Подписчик