[07 – ROS2 PKG Service / Action]

한국폴리텍대학교 성남캠퍼스

0 Homework Review

0 Homework Review

❖ Missions

1. 사용자가 주고 받을 메시지 만들기
    1. 랜덤주행 시작, 정지
    2. 랜덤주행 영역지정 좌표(4개)
2. 2개의 노드(node1, node2)를 생성하고 node1은 turtlesim_node로 랜덤주행 Topic을 발행하며 node2로 부터 사용자가 정의한 메시지를 수신하여 랜덤주행 정보에 적용
3. Node2는 사용자가 값을 설정하고 사용자가 정의한 메시지를 이용하여 node1으로 Topic 발행

1 Service Definition

Service definition

❖ Service 정의하기

1. Service data type definition

2. Make a Service Server

3. Make a Service Client

4. Multi-Spawn turtles Service

1 Service definition

❖ Service Data Type definition

```
daesung@DSThinkPad:~/ros2_work/src/first_msg$ tree
.
├── CMakeLists.txt
├── include
│   └── first_msg
├── msg
│   └── First.msg
├── package.xml
├── src
└── srv
    └── MultiSpawn.srv

5 directories, 4 files
daesung@DSThinkPad:~/ros2_work/src/first_msg$
```

- Service Request / Response Data Type을 정의하기 위해 위와 같이 srv 폴더를 생성하고 ~~~.srv 파일 생성

```
daesung@DSThinkPad:~/ros2_work/src/hmwk01_msg/srv$ cat HmWk01.srv
int64 num
---
float64[] x
float64[] y
float64[] theta
daesung@DSThinkPad:~/ros2_work/src/hmwk01_msg/srv$
```

1 Service definition

❖ Service Data Type definition

```
daesung@DSThinkPad:~/ros2_work/src/first_msg$ tree
.
├── CMakeLists.txt
├── include
│   └── first_msg
├── msg
│   └── First.msg
├── package.xml
├── src
└── srv
    └── MultiSpawn.srv

5 directories, 4 files
daesung@DSThinkPad:~/ros2_work/src/first_msg$
```

- package.xml 파일은 변경사항 없음
- CMakeLists.txt 파일은 다음과 같이 새로 생성된 Service Data 파일의 경로 추가

```
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/First.msg"
  "srv/MultiSpawn.srv"
)
```

1 Service definition

❖ Service Data Type definition

- Workspace 위치로 이동하여 colcon build 실행

```
daesung@DSThinkPad:~/ros2_work$ colbds first_msg
Starting >>> first_msg
Finished <<< first_msg [0.47s]

Summary: 1 package finished [0.75s]
daesung@DSThinkPad:~/ros2_work$
```

alias로 ~/.bashrc에 추가

```
alias colbd="colcon build"
alias colbds="colcon build --packages-select"
```

- ros2 interface show 명령어를 통해 Service data type이 등록되었는지 확인

```
daesung@DSThinkPad:~/ros2_work$ ros2 interface show first_msg/srv/MultiSpawn
int64 num
---
float64[] x
float64[] y
float64[] theta
daesung@DSThinkPad:~/ros2_work$
```

1 Service definition

❖ Make a service server

- Service Server는 앞서 사용했던 pkg폴더에 Python파일을 추가로 생성

```
daesung@DSThinkPad:~/ros2_work/src/first_pkg$ tree
.
├── first_pkg
│   ├── __init__.py
│   ├── srv_server.py
│   ├── topic_publisher.py
│   ├── topic_subscriber.py
│   └── turtle_cmd_pose.py
├── package.xml
├── resource
│   └── first_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py

3 directories, 12 files
daesung@DSThinkPad:~/ros2_work/src/first_pkg$
```

Service definition

❖ Make a service server

- srv_server.py

```
import rclpy as rp
from rclpy.node import Node
from first_msg.srv import MultiSpawn

class Multi_spawn(Node):
    def __init__(self):
        super().__init__('Multi_spawn_node')
        self.server = self.create_service(MultiSpawn, 'Multi_spawn_node', self.callback_srv_server)

    def callback_srv_server(self, req, res):
        print('Req : ', req)
        res.x = [1., 2., 3.]
        res.y = [10., 20.]
        res.theta = [100., 200., 300.]
        return res

def main(args=None):
    rp.init(args=args)
    mlt_spn = Multi_spawn()
    rp.spin(mlt_spn)
    rp.shutdown()

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

1 Service definition

❖ Make a service server

- 다음과 같이 setpup.py에 작성한 파일의 메인 함수 추가

```
daesung@DSThinkPad:~/ros2_work/src/first_pkg$ tree
.
├── first_pkg
│   ├── __init__.py
│   ├── srv_server.py
│   ├── topic_publisher.py
│   ├── topic_subscriber.py
│   └── turtle_cmd_pose.py
├── package.xml
├── resource
│   └── first_pkg
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py

3 directories, 12 files
daesung@DSThinkPad:~/ros2_
```

```
entry_points={
        'console_scripts': [
            'topic_subscriber = first_pkg.topic_subscriber:main',
            'topic_publisher = first_pkg.topic_publisher:main',
            'turtle_cmd_pose = first_pkg.turtle_cmd_pose:main'
            'srv_server = first_pkg.srv_server:main'
        ],
    },
```

1 Service definition

❖ Make a service server

- Package 빌드 후 서비스가 나타나는지 확인

```
daesung@DSThinkPad:~/ros2_work$ colbds first_pkg
Starting >>> first_pkg
Finished <<< first_pkg [0.63s]

Summary: 1 package finished [1.04s]
daesung@DSThinkPad:~/ros2_work$
```

- 서비스 실행 전 다음과 같이 패키지의 노드 확인

```
daesung@DSThinkPad:~/ros2_work$ ros2 run first_pkg
--prefix
srv_server
topic_publisher
topic_subscriber
turtle_cmd_pose
turtle_cmd_pose\ =\ first_pkg.turtle_cmd_pose:mainsrv_server
daesung@DSThinkPad:~/ros2_work$ ros2 run first_pkg
```

1 Service definition

❖ Make a service server

- Service server 실행 후 Service list에 나타나는지 확인

```
daesung@DSThinkPad:~$ ros2 service list -t
/Multi_spawn_node [first_msg/srv/MultiSpawn]
/Multi_spawn_node/describe_parameters [rcl_interfaces/srv/DescribeParameters]
/Multi_spawn_node/get_parameter_types [rcl_interfaces/srv/GetParameterTypes]
/Multi_spawn_node/get_parameters [rcl_interfaces/srv/GetParameters]
/Multi_spawn_node/list_parameters [rcl_interfaces/srv/ListParameters]
/Multi_spawn_node/set_parameters [rcl_interfaces/srv/SetParameters]
/Multi_spawn_node/set_parameters_atomically [rcl_interfaces/srv/SetParametersAtomically]
/turtle1/teleport_absolute [turtlesim/srv/TeleportAbsolute]
daesung@DSThinkPad:~$
```

1 Service definition

❖ Make a service server

- 다음과 같은 service call 호출

```
ros2 service call /Multi_spawn_node first_msg/srv/MultiSpawn "{num: 1}"
```

Service client

```
daesung@DSThinkPad:~$ ros2 service call /Multi_spawn_node first_msg/srv/MultiSpawn "{num: 1}"
waiting for service to become available...
requester: making request: first_msg.srv.MultiSpawn_Request(num=1)

response:
first_msg.srv.MultiSpawn_Response(x=[1.0, 2.0, 3.0], y=[10.0, 20.0], theta=[100.0, 200.0, 300.0])
```

srv_server

```
daesung@DSThinkPad:~/ros2_work$ ros2 run first_pkg srv_server
Req :  first_msg.srv.MultiSpawn_Request(num=1)
```

Service definition

- Make a service client
  - srv_server.py를 다음과 같이 수정

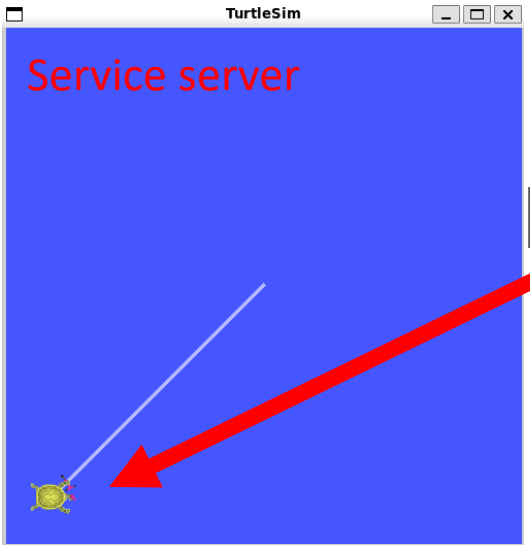
```python
import rclpy as rp
from rclpy.node import Node
from first_msg.srv import MultiSpawn
from turtlesim.srv import TeleportAbsolute

class Multi_spawn(Node):
    def __init__(self):
        super().__init__('Multi_spawn_node')
        self.server = self.create_service(MultiSpawn, 'Multi_spawn_node', self.callback_srv_server)
        self.teleport = self.create_client(TeleportAbsolute, '/turtle1/teleport_absolute')
        self.req_teleabs = TeleportAbsolute.Request()

    def callback_srv_server(self, req, res):
        self.req_teleabs.x = 1.
        self.req_teleabs.y = 1.
        self.req_teleabs.theta = 0.
        self.teleport.call_async(self.req_teleabs)
        return res

def main(args=None):
    rp.init(args=args)
    mlt_spn = Multi_spawn()
    rp.spin(mlt_spn)
    rp.shutdown()

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

1 Service definition

❖ Make a service client

- Package 빌드 후 실행 확인

```
def callback_srv_server(self, req, res):
    self.req_teleabs.x = 1.
    self.req_teleabs.y = 1.
    self.req_teleabs.theta = 0.
    self.teleport.call_async(self.req_teleabs)
```




```
daesung@DSThinkPad:~/ros2_work$ ros2 run first_pkg srv_server
```

Service client

Service server

```
daesung@DSThinkPad:~$ ros2 service call /Multi_spawn_node first_msg/srv/MultiSpawn "{num: 1}"
requester: making request: first_msg.srv.MultiSpawn_Request(num=1)

response:
first_msg.srv.MultiSpawn_Response(x=[], y=[], theta=[])

daesung@DSThinkPad:~$
```

Service client

ROS

Service definition

❖ Mission

- 원하는 위치에 원하는 만큼 거북이를 spawn할 수 있는 코드를 작성 하시오.

Service definition

- Hint

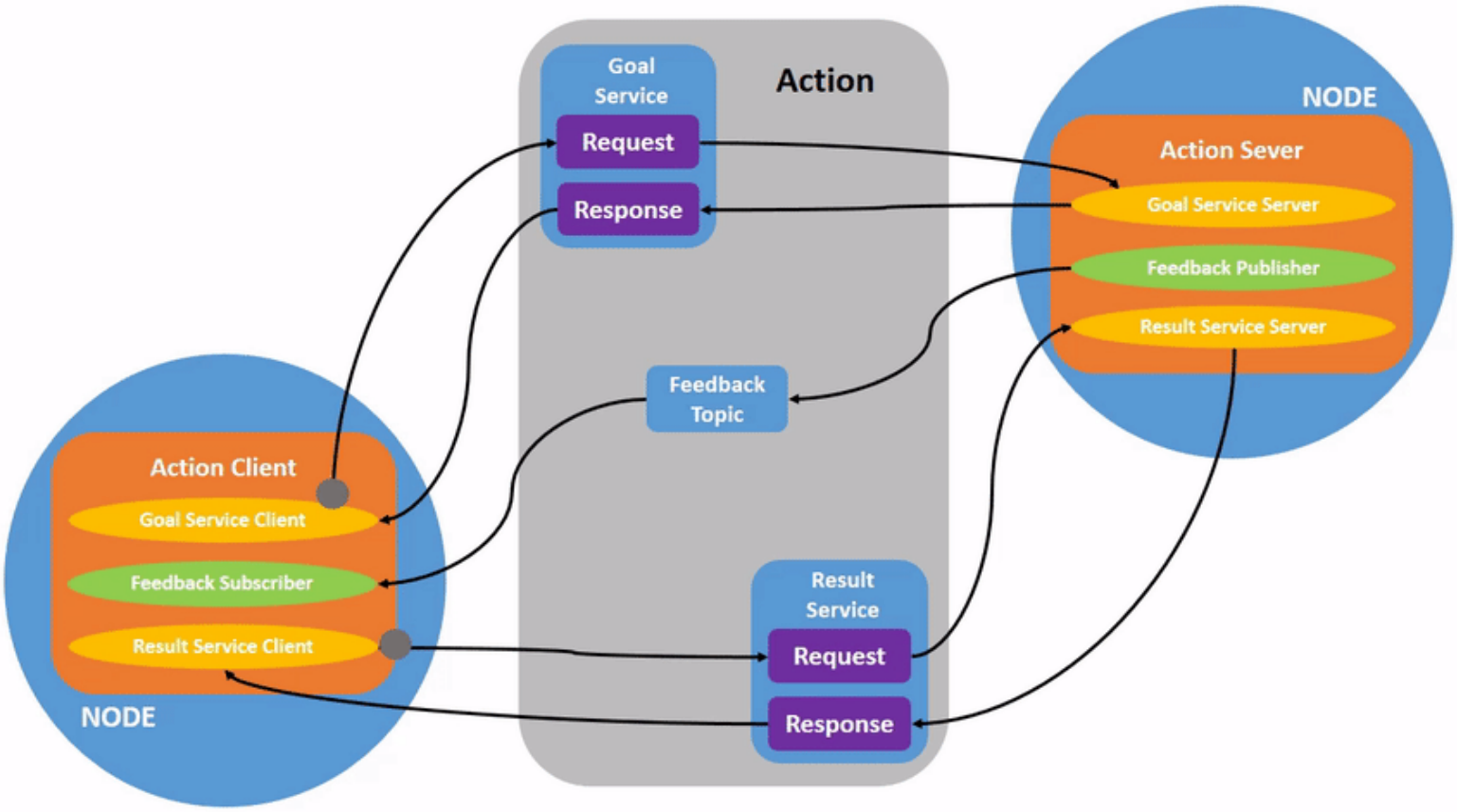
```
import rclpy as rp
from rclpy.node import Node
from first_msg.srv import MultiSpawn
from turtlesim.srv import TeleportAbsolute

class Multi_spawn(Node):
    def __init__(self):
        super().__init__('Multi_spawn_node')
        self.server = self.create_service(MultiSpawn, 'Multi_spawn_node', self.callback_srv_server)
        self.teleport = self.create_client(TeleportAbsolute, '/turtle1/teleport_absolute')
        self.req_teleabs = TeleportAbsolute.Request()

    def callback_srv_server(self, req, res):
        self.num = req.num
        print(self.num)
        res.x.append(3.)
        res.y.append(4.)
        res.theta.append(0.)
        self.req_teleabs.x = 1.
        self.req_teleabs.y = 1.
        self.req_teleabs.theta = 0.
        self.teleport.call_async(self.req_teleabs)
        return res

def main(args=None):
    rp.init(args=args)
    mlt_spn = Multi_spawn()
    rp.spin(mlt_spn)
    rp.shutdown()

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

2 Action Definition

2 Action definition

❖ Action review

- 기본적으로 3쌍의 서버와 클라이언트로 구성되어 있음

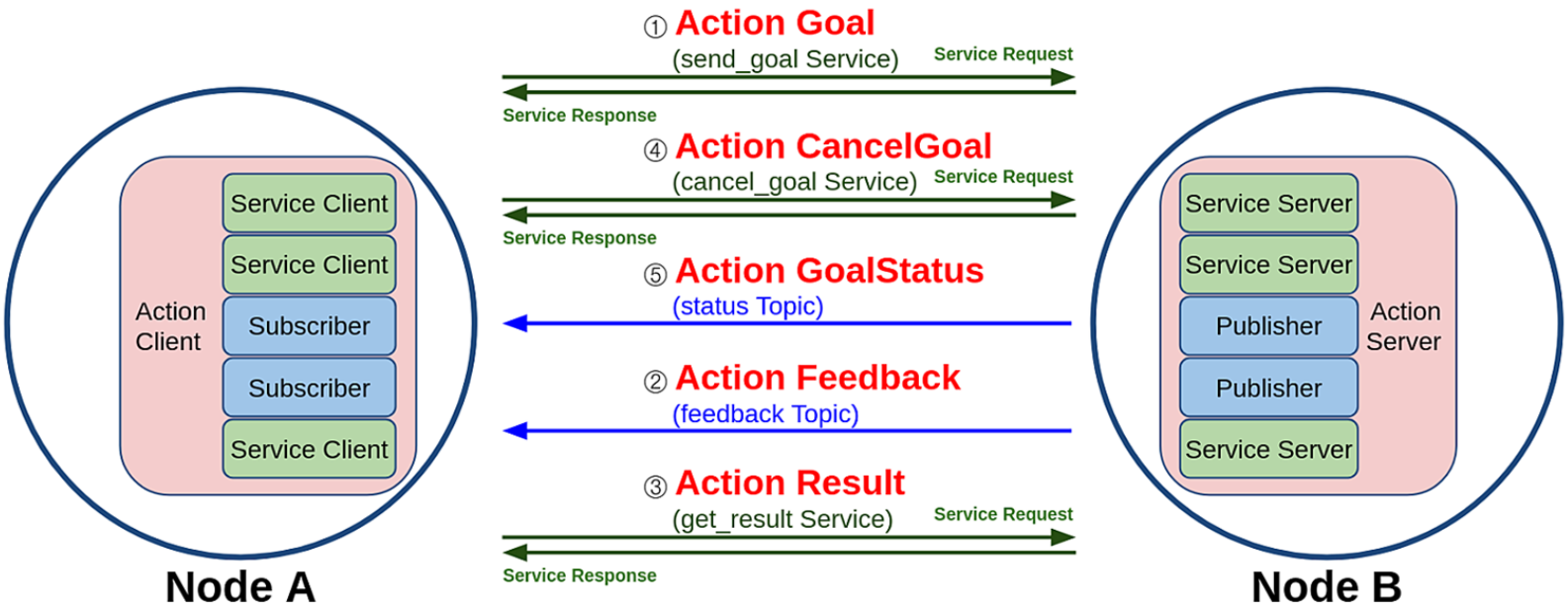


ROS

2 Action definition

❖ Action review

- Action Client = Service Client 3개 + Topic Subscriber 2개
- Action Server = Service Server 3개 + Topic Publisher 2개
- 액션 목표/피드백/결과(goal/feedback/result) 데이터는 msg 및 srv 인터페이스의 변형으로 action 인터페이스라 함



ROS

2 Action definition

❖ Action data type 정의하기

- Action data를 저장하기 위해 first_msg(토픽과 서비스 데이터 타입 선언한 package)에 action 폴더 만들기
- action 폴더에 ~~~.action 파일 생성

```
daesung@DSThinkPad:~/ros2_work/src/first_msg$ tree
.
├── action
│   └── ActionData.action
├── CMakeLists.txt
├── include
│   └── first_msg
├── msg
│   └── First.msg
├── package.xml
├── src
└── srv
    ├── MultiSpawn.srv
    └── TurtleCmd.srv

6 directories, 6 files
daesung@DSThinkPad:~/ros2_work/src/first_msg$
```

2 Action definition

❖ Action data type 정의하기

- Action data를 저장하기 위해 first_msg(토픽과 서비스 데이터 타입 선언한 package)에 action 폴더 만들기
- action 폴더에 ~~~.action 파일 생성

```
daesung@DSThinkPad:~/ros2_work/src/first_msg$ tree
.
├── action
│   └── ActionData.action
├── CMakeLists.txt
├── include
│   └── first_msg
├── msg
│   └── First.msg
├── package.xml
├── src
└── srv
    ├── MultiSpawn.srv
    └── TurtleCmd.srv

6 directories, 6 files
daesung@DSThinkPad:~/ros2_work/src/first_msg$
```

action > ActionData.action

```
1   # Request
2   float32 linear_x
3   float32 angular_z
4   float32 dist
5
6   ---
7   # Result
8   float32 pos_x
9   float32 pos_y
10  float32 pos_theta
11  float32 result_data
12
13  ---
14  # Feedback
15  float32 reemained_dist
```

2 Action definition

❖ Action data 빌드

- Action data를 빌드하기 위해 다음과 같이 CMakeLists.txt와 package.xml 파일 수정

- CMakelists.txt

```
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/First.msg"
  "srv/MultiSpawn.srv"
  "srv/TurtleCmd.srv"
  "action/ActionData.action"
)
```

- Package.xml

```
  <build_depend>rosidl_default_generators</build_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <depend>action_msgs</depend>
```

2 Action definition

❖ Action data 빌드 후 확인

- Action data가 포함된 package를 빌드한 후 다음과 같은 명령을 통해 ROS2에 등록되었는지 확인

```
ros2 interface show first_msg/action/ActionData
```

```
daesung@DSThinkPad:~/ros2_work$ ros2 interface show first_msg/action/ActionData
# Request
float32 linear_x
float32 angular_z
float32 dist

---
# Result
float32 pos_x
float32 pos_y
float32 pos_theta
float32 result_data

---
# Feedback
float32 remained_dist
daesung@DSThinkPad:~/ros2_work$
```

2 Action definition

❖ 간단한 Action Server 테스트

- 지금까지 작성한 프로그램이 포함되어 있는 패키지에 Action Server를 테스트하기 위한 소스코드 추가

```python
import rclpy as rp
from rclpy.action import ActionServer
from rclpy.node import Node
from first_msg.action import ActionData

class ActionServer(Node):
    def __init__(self):
        super().__init__('Action_Server_node')
        self.action_server = ActionServer(
            self,
            ActionData,
            'first_action',
            self.execute_callback
        )

    def execute_callback(self, goal_handle):
        goal_handle.succeed()
        result = ActionData.Result()
        return result
```

```python
def main(args=None):
    rp.init(args=args)
    act_srv = ActionServer()
    rp.spin(act_srv)
    rp.shutdown()

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

2 Action definition

❖ 간단한 Action Server 테스트

- 작성한 코드가 포함된 패키지의 setup.py에 코드를 등록한 후 빌드
- Action Sever 실행 후 테스트 메시지를 보내서 확인

```
daesung@DSThinkPad:~/ros2_work$ ros2 run first_pkg action_server
```

```
ros2 action send_goal /first_action first_msg/action/ActionData "{linear_x: 0,
angular_z: 0, dist: 0}"
```

```
daesung@DSThinkPad:~$ ros2 action send_goal /first_action first_msg/action/ActionData "{linear_x: 0, ang
ular_z: 0, dist: 0}"
Waiting for an action server to become available...
Sending goal:
     linear_x: 0.0
angular_z: 0.0
dist: 0.0

Goal accepted with ID: a03fa2214a0a4075b0e1141ccf6d48d5

Result:
    pos_x: 0.0
pos_y: 0.0
pos_theta: 0.0
result_data: 0.0

Goal finished with status: SUCCEEDED
daesung@DSThinkPad:~$
```

2 Action definition

❖ Action Server에 피드백 추가

- Action Goal에 도달하기 까지의 중간과정을 토픽으로 발행
- 가상으로 피드백 발송하는 코드를 execute_callback 함수에 추가

```
    def execute_callback(self, goal_handle):
        feedback_msg = ActionData.Feedback()
        for i in range(0, 10):
            feedback_msg.remained_dist = float(i)
            goal_handle.publish_feedback(feedback_msg)
            time.sleep(0.5)

        goal_handle.succeed()
        result = ActionData.Result()
        return result
```

- Action Server를 실행시키고 send_goal에 다음과 같은 피드백 확인 옵션을 사용하여 동작 확인

```
ros2 action send_goal --feedback /first_action first_msg/action/ActionData
"{linear_x: 0, angular_z: 0, dist: 0}"
```

2 Action definition

❖ Mission

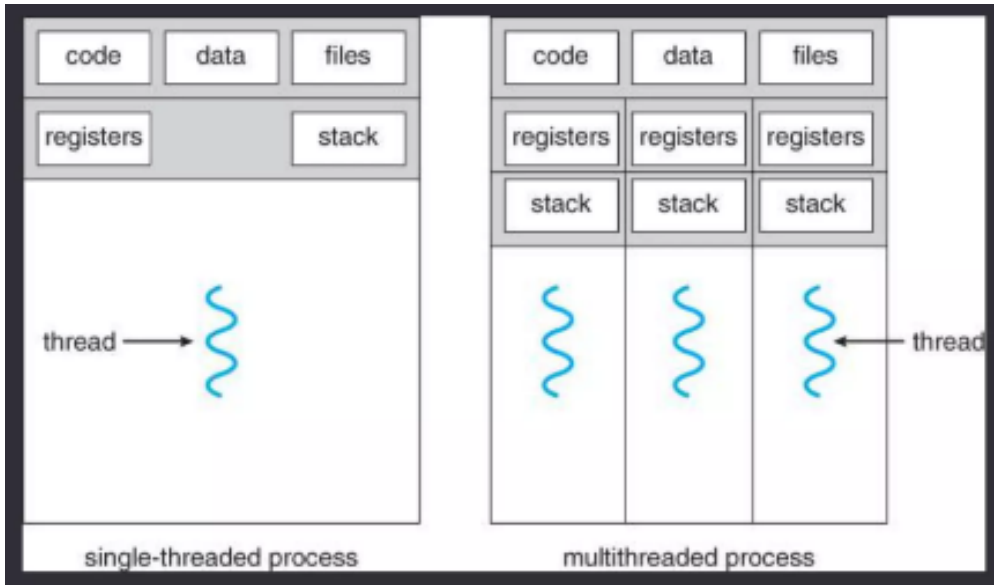
- turtlesim_node를 실행시킨 후 거북이는 (x: 1, y: 1, theta:0)으로 위치
- 사용자가 지정한 영역 (x, y, x_l, y_l) → (x, y) (x, y+y_l), (x+x_l, y) (x+x_l, y+y_l) 으로 이동 하도록 동작
  - 조건1) 거북이는 랜덤주행
  - 조건2) action 방식을 이용하여 사용자가 도착영역을 goal로 지정하면 주행시작
  - 조건3) 도착영역에 도달할 때 까지 현재 좌표를 피드백으로 전송
  - 조건4) 도착영역에 도달하면 거북이 주행 종료

3 Multi-Thread

3 Multi-Thread

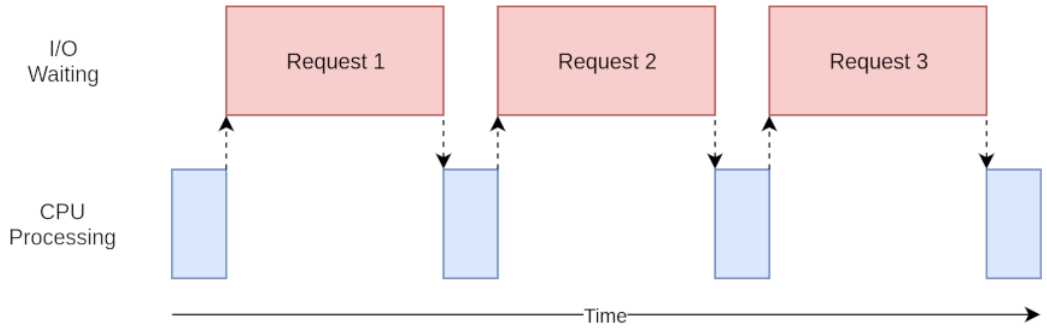
❖ Multi Thread 정의 및 필요성

- 지금까지 Topic 발행/구독, Service 서버/클라이언트, Action 구현 등 하나의 프로그램에서 여러가지 기능을 구현할 경우, 여러 기능들의 실행으로 메시지 누락 등의 문제가 발생하여 명령어를 제대로 수신하지 못하고 실행하지 못하는 경우가 발생
- 상기 문제 해결을 위해 Multi Thread를 사용

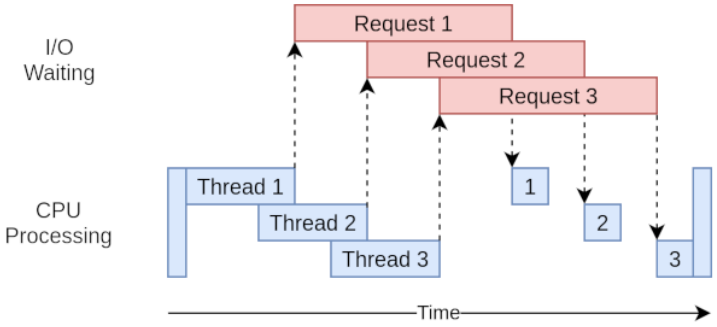


3 Multi-Thread

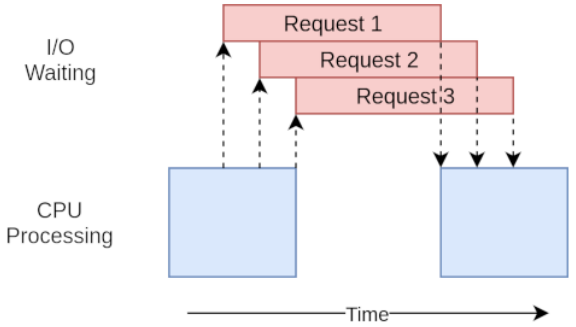
- Multi Thread 정의 및 필요성



(a) Single-threaded process



(b) Multi-threaded process
with GIL acquired by current thread



(c) Single-threaded process
with asyncio

3 Multi-Thread

❖ Multi Thread Sample Code

- 하나의 노드에 동시에 여러 상황이 발생하는 경우에는 Multi Thread를 사용하는 것을 추천

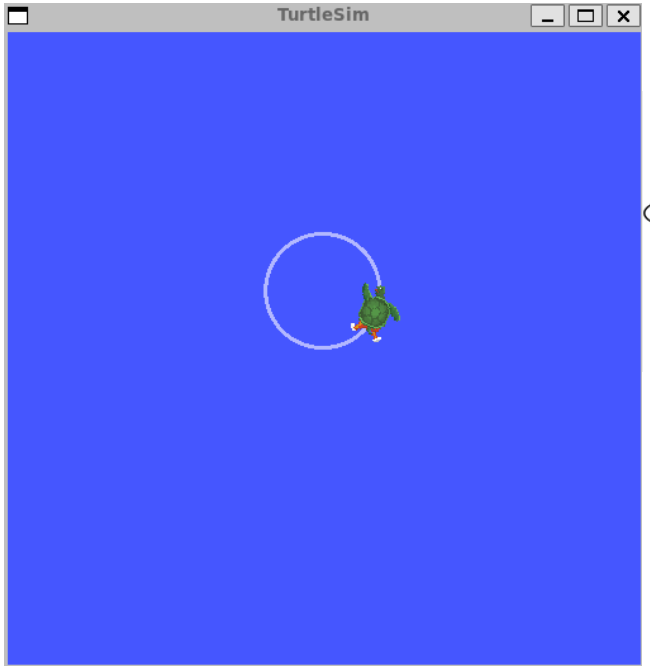
```python
import rclpy as rp
from rclpy.executors import MultiThreadedExecutor
from rclpy.node import Node
from first_pkg.topic_subscriber import TurtlesimSubscriber
from first_pkg.topic_publisher import TurtlesimPublisher

def main(args=None):
    rp.init(args=args)

    sub = TurtlesimSubscriber()
    pub = TurtlesimPublisher()

    exe = MultiThreadedExecutor()

    exe.add_node(sub)
    exe.add_node(pub)
    try:
        exe.spin()
    finally:
        exe.shutdown()
        sub.destroy_node()
        pub.destroy_node()
        rp.shutdown()

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

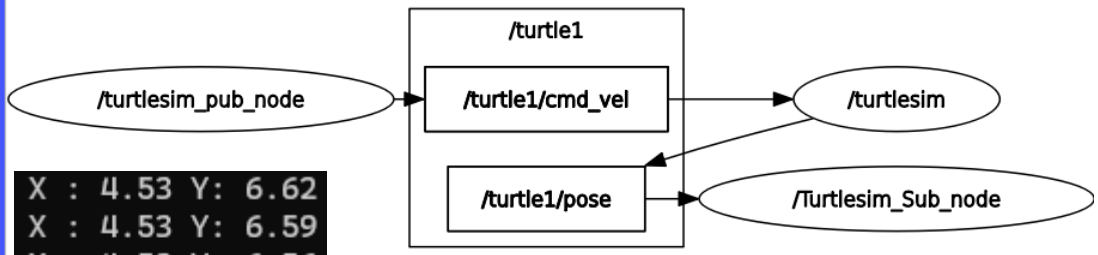
3 Multi-Thread

❖ Multi Thread Sample Code

- 예제 코드를 실행하면 publishing과 Subscription이 동시에 실행되는 노드임을 확인할 수 있음

```
ros2 run first_pkg multi_thread_test
```






```
X : 4.53 Y: 6.62
X : 4.53 Y: 6.59
X : 4.53 Y: 6.56
X : 4.53 Y: 6.53
X : 4.53 Y: 6.50
X : 4.53 Y: 6.47
X : 4.53 Y: 6.43
X : 4.54 Y: 6.40
X : 4.54 Y: 6.37
```

3 Multi-Thread

❖ 지정한 거리만큼 이동하는 Action Server

1. 사용자가 속도와 이동거리를 지정하여 Action Server에 명령 전달
2. Action Server는 이동하는 거리를 계산해서 사용자가 지정한 거리만큼 이동한 후 멈추기
3. Action Server는 이동하는 동안 진행 상황을 feedback

Multi-Thread

- 지정한 거리만큼 이동하는 Action Server – class Sub_Action

```
import rclpy as rp
from rclpy.action import ActionServer
from rclpy.executors import MultiThreadedExecutor
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from first_msg.action import ActionData
from first_pkg.topic_subscriber import TurtlesimSubscriber

import math
import time

class Sub_Action(TurtlesimSubscriber):
    def __init__(self, ac_server):
        super().__init__()
        self.ac_server = ac_server

    def callback(self, msg):
        self.ac_server.cur_pose = msg
```

Multi-Thread

- 지정한 거리만큼 이동하는 Action Server – Class DistServer-1

```
class DistServer(Node):
    def __init__(self):
        super().__init__('dist_action_server')
        self.t_dist = 0
        self.first_time = True
        self.cur_pose = Pose()
        self.pre_pose = Pose()
        self.pub = self.create_publisher(Twist, '/turtle1/cmd_vel', 10)
        self.act_srv = ActionServer(self, ActionData, 'dist_act_srv', self.exe_callback)

    def cal_diff_pose(self):
        if self.first_time:
            self.pre_pose.x = self.cur_pose.x
            self.pre_pose.y = self.cur_pose.y
            self.first_time = False

        diff_dist = math.sqrt((self.cur_pose.x - self.pre_pose.x)**2+\
                              (self.cur_pose.y - self.pre_pose.y)**2)
        self.pre_pose = self.cur_pose
        return diff_dist
```

3 Multi-Thread

- 지정한 거리만큼 이동하는 Action Server – Class DistServer-2

```
def exe_callback(self, goal_handle):
    feedback_msg = ActionData.Feedback()

    msg = Twist()
    msg.linear.x = goal_handle.request.linear_x
    msg.angular.z = goal_handle.request.angular_z

    while True:
        self.t_dist += self.cal_diff_pose()
        feedback_msg.remained_dist = goal_handle.request.dist - self.t_dist
        goal_handle.publish_feedback(feedback_msg)
        self.pub.publish(msg)
        time.sleep(0.01)

        if feedback_msg.remained_dist < 0.01 :
            break

    goal_handle.succeed()
    rlt = ActionData.Result()
```

3 Multi-Thread

- 지정한 거리만큼 이동하는 Action Server – Class DistServer-3 / main

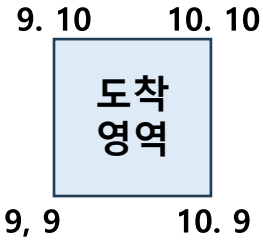
```
        rlt.pos_x = self.cur_pose.x
        rlt.pos_y = self.cur_pose.y
        rlt.pos_theta = self.cur_pose.theta
        rlt.result_data = self.t_dist

        self.t_dist = 0
        self.first_time = True

        return rlt
```

```
def main(args=None):
    rp.init(args=args)
    exe = MultiThreadedExecutor()

    ac = DistServer()
    sub = Sub_Action(ac_server=ac)

    exe.add_node(sub)
    exe.add_node(ac)

    try:
        exe.spin()
    finally:
        exe.shutdown()
        sub.destroy_node()
        ac.destroy_node()
        rp.shutdown()

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

3 Multi-Thread

❖ Mission - 다음 조건을 Multi-Thread로 작성 하시오.

1. Multi-thread 사용
2. action을 이용하여 설계
3. turtlesim_node 내에 거북이가 지나갈 수 없는 장애물 영역을 3개 랜덤하게 설정 (직사각형으로 제한, 거북이가 지나갈 수 있는 경로가 있으면 크기 무관)
4. 좌측 하단 영역(x: 1, y: 1) 에서 출발하여 우측상단 영역에 도착할 수 있도록 자율 주행 프로그램 작성



5. 가능한 최단 시간에 도달할 수 있도록 소프트웨어 설계


Q & A