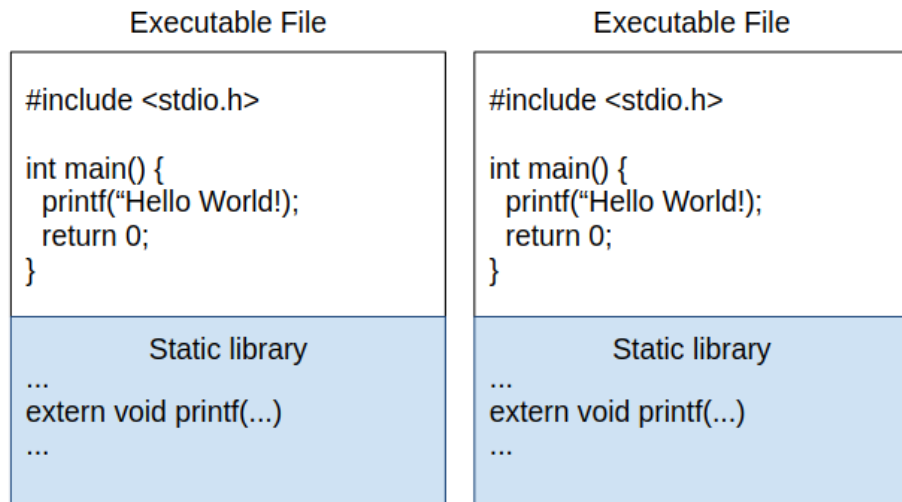


5장 Component

이철훈

정적 링킹과 동적 링킹

- 링킹(linking) : 컴파일 과정에서 여러 오브젝트 파일을 실행파일과 연결해주는 과정
- 정적 링킹 : 컴파일 과정에서 링커가 정적 라이브러리를 복사하여 실행파일에 저장하는 것
- 동적 링킹 : 런타임에서 메모리에 공유 라이브러리가 있다면 해당 코드에 접근하여 사용하고, 그렇지 않으면 메모리에 올려주는 것

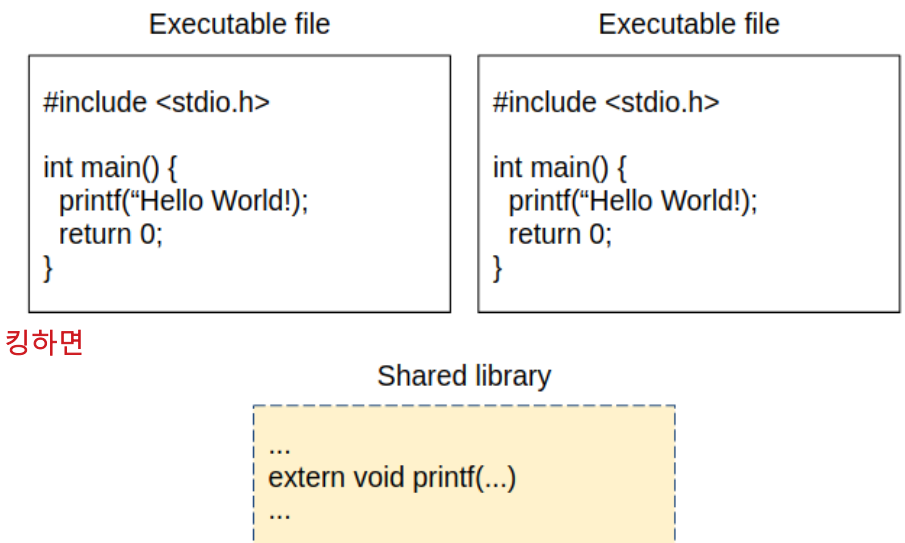


Static linking at compile time

[그림 1 정적 링킹]

공유 라이브러리화 하여 동적 링킹하면
보다 효율적으로 메모리를 사용

정적 링킹으로 컴파일된 프로그램은 복
수개가 실행되면 같은 코드가 중복되어
메모리에 적재



Dynamic linking at run time

[그림 2 동적 링킹]

동적 로딩과 ROS2 Component

- 동적 링킹 : 프로그램 시작 시, 운영 체제가 공유 라이브러리를 메모리에 찾아서 접근함
- 동적 로딩 : 런타임에서 프로그램이 공유 라이브러리 사용을 결정함
- ROS에서 동적 로딩을 지원하는 패키지 : `class_loader`, `pluginlib`
- → ROS2에서는 `class_loader` 기반의 components를 제공함, 이 컴포넌트는 `rclcpp`에서만 사용이 가능하며, `main` 함수를 통해 노드를 직접 실행하거나, `main` 함수 없이 런타임에 컨테이너로 동적 로딩되어 실행될 수 있음
- → 컴포넌트를 사용함으로써 동적 링킹의 장점을 활용할 수 있음
- 실행방법
- (1) generic container process로 service 통신을 이용하여 컴포넌트 등록
- (2) `main` 함수에 `executors`를 선언하여 컴포넌트를 등록
- (3) Launch를 이용하여 복수 개의 컴포넌트를 한번에 등록

Component 실행 예

- Generic Container Process

아래 명령어를 통해 컨테이너를 실행해 보고, 컴포넌트 리스트를 확인해보자.

```
# Terminal 1
$ ros2 run rclcpp_components component_container

# Terminal 2
$ ros2 component list
/ComponentManager
```



이제 위에서 확인한 talker 컴포넌트를 실행 중인 컨테이너에 적재해보자. 적재가 완료되면 컨테이너를 실행시켰던 터미널 창에서 퍼블리쉬 되는 정보를 확인할 수 있다.

```
# Terminal 2
$ ros2 component load /ComponentManager composition composition::Talker
Loaded component 1 into '/ComponentManager' container node as '/talker'

# Terminal 1
$ ros2 run rclcpp_components component_container
[INFO]: Load Library: /home/oroca/robot_ws/install/composition/lib/libtalker_component.so
[INFO]: Found class: rclcpp_components::NodeFactoryTemplate<composition::Talker>
[INFO]: Instantiate class: rclcpp_components::NodeFactoryTemplate<composition::Talker>
[INFO]: Publishing: 'Hello World: 1'
[INFO]: Publishing: 'Hello World: 2'
[INFO]: Publishing: 'Hello World: 3'
[INFO]: Publishing: 'Hello World: 4'
[INFO]: Publishing: 'Hello World: 5'
[INFO]: Publishing: 'Hello World: 6'
```



아래 명령어를 통해 listener 컴포넌트도 실행시켜 보자. 컨테이너를 실행시켰던 터미널에서 listener 컴포넌트가 등록된 것을 확인할 수 있고, talker 와 listener 컴포넌트의 로그도 함께 확인할 수 있다.

```
# Terminal 2
$ ros2 component load /ComponentManager composition composition::Listener
Loaded component 2 into '/ComponentManager' container node as '/listener'

# Terminal 1
[INFO]: Load Library: /home/oroca/robot_ws/install/composition/lib/liblistener_component.so
[INFO]: Found class: rclcpp_components::NodeFactoryTemplate<composition::Listener>
[INFO]: Instantiate class: rclcpp_components::NodeFactoryTemplate<composition::Listener>
[INFO]: Publishing: 'Hello World: 159'
[INFO]: I heard: [Hello World: 159]
[INFO]: Publishing: 'Hello World: 160'
[INFO]: I heard: [Hello World: 160]
[INFO]: Publishing: 'Hello World: 161'
[INFO]: I heard: [Hello World: 161]
[INFO]: Publishing: 'Hello World: 162'
[INFO]: I heard: [Hello World: 162]
[INFO]: Publishing: 'Hello World: 163'
[INFO]: I heard: [Hello World: 163]
[INFO]: Publishing: 'Hello World: 164'
```

6장 RQT Plugin

RQt 및 ROS2패키지

- RQt

: 플러그인 형태로 다양한 도구와 인터페이스를 구현할 수 있는 ROS의 GUI 툴박스

: ROS + Qt의 합성어, C++, Python 모두 사용이 가능

: ROS의 토픽, 서비스, 액션 프로그래밍 사용 가능함

- RQt 플러그인 관련 ROS2 패키지

(1) rqt 패키지 : RQt의 메타패키지로 rqt_gui, rqt_gui_cpp, rqt_gui_py, rqt_py_common 패키지가 포함되어 있음

(2) rqt_gui 패키지 : 'rqt' 위젯을 단일 창에 도킹할 수 있는 위젯 패키지

(3) rqt_gui_cpp, rqt_gui_py 패키지 : 각각 C++, Python 클라이언트 라이브러리를 사용하여 제작할 수 있는 Rqt GUI 플러그인 API를 제공하는 패키지

(4) rqt_py_common 패키지 : Python으로 작성된 RQt 플러그인에서 공용으로 사용되는 기능을 모듈로 제공하는 패키지

RQt 개발 환경

- RQt 개발환경은 Qt 5.12.x 이상, Qt용 IDE는 Qt Creator 4.5.x 이상
- 설치 (Debian Packages 기반 ros-foxy-desktop'으로 ROS 설치하면 같이 설치됨)

```
$ sudo apt install qtcreator
```

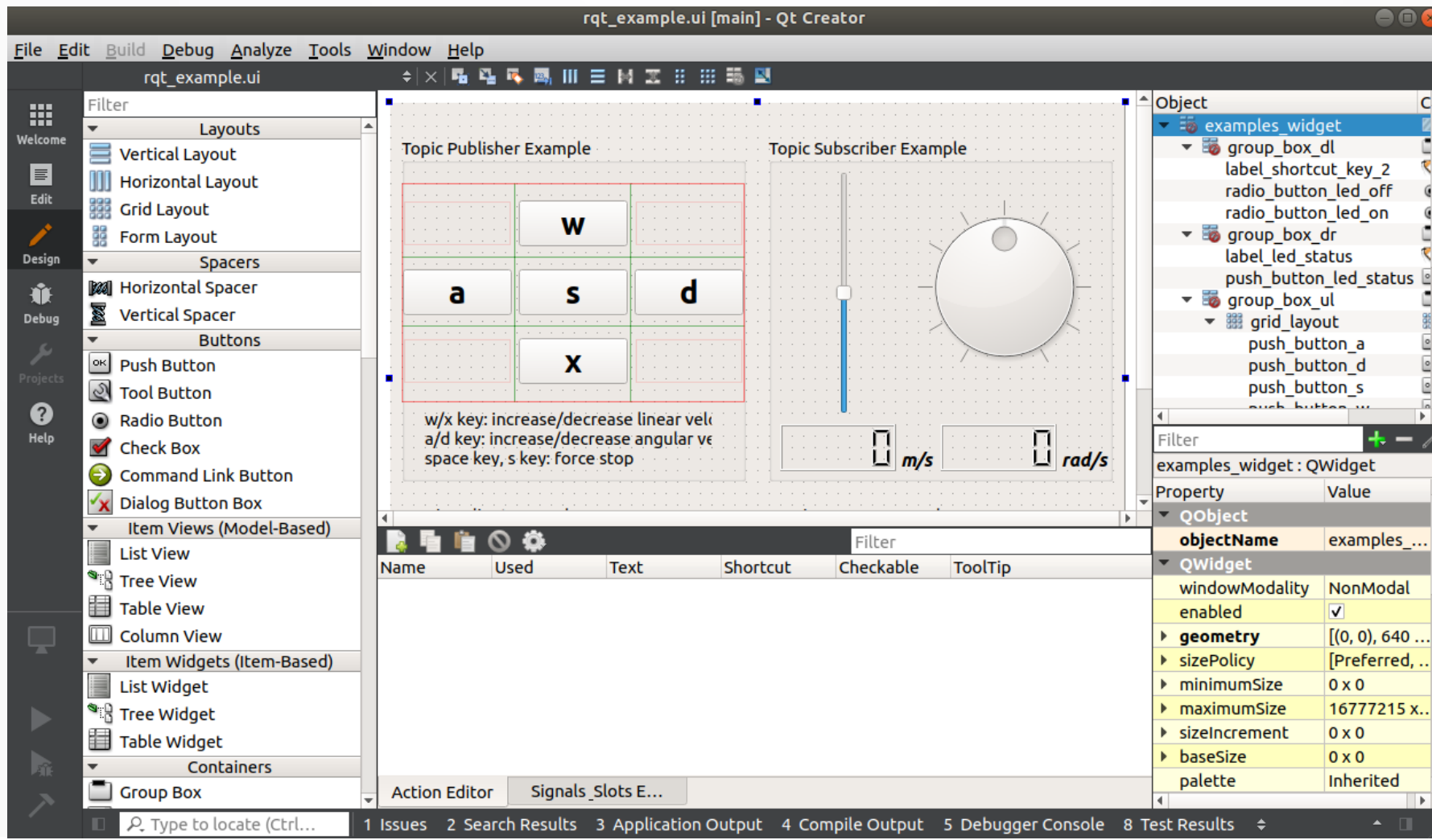
RQt 플러그인 작성 순서

- RQt 패키지 생성은 일반적인 패키지 생성과 동일하지만, RQt 플러그인의 기본 기능 관련과 GUI 관련 패키지를 의존성 패키지로 포함시킴

```
lch@lch-TB250-BTC:~/robot_ws/src$ ros2 pkg create my_first_rqt_plugin_pkg --build-type ament_cmake
--dependencies rclpy rqt_gui rqt_gui_py python_qt_binding
going to create a new package
package name: my_first_rqt_plugin_pkg
destination directory: /home/lch/robot_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['lch <lch@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['rclpy', 'rqt_gui', 'rqt_gui_py', 'python_qt_binding']
creating folder ./my_first_rqt_plugin_pkg
creating ./my_first_rqt_plugin_pkg/package.xml
creating source and include folder
creating folder ./my_first_rqt_plugin_pkg/src
creating folder ./my_first_rqt_plugin_pkg/include/my_first_rqt_plugin_pkg
creating ./my_first_rqt_plugin_pkg/CMakeLists.txt
lch@lch-TB250-BTC:~/robot_ws/src$
```


RQt 예제의 구성

```
lch@lch-TB250-BTC:~/robot_ws$ qtcreeator ~/robot_ws/src/ros2-seminar-examples/rqt_example/resource/rqt_example.ui
"The command \"/home/lch/robot_ws\" could not be started."
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
```



RQt 예제의 구성

- Rqt 플러그인 패키지 생성

rqt_example

<https://github.com/robotpilot/ros2-seminar-examples/>

```
$ cd  
$ git clone https://github.com/robotpilot/ros2-seminar-examples.git  
$ cd  
$ cbp rqt_example
```

- 패키지 설정 파일 수정

: ROS2 패키지와 다른 점은 <export> 태그에

Rqt 플러그인 파일(plugin.xml)을 추가함

rqt_example/package.xml

<https://github.com/robotpilot/ros2-seminar-examples>

(일부 생략)

<export>

<build_type>ament_cmake</build_type>

<rqt_gui plugin="\${prefix}/plugin.xml"/>

</export>

(일부 생략)

RQt 예제의 구성

- 빌드 설정 파일 수정

rqt_example/CMakeLists.txt

<https://github.com/robotpilot/ros2-seminar-examples>

```
install(FILES
  plugin.xml
  DESTINATION share/${PROJECT_NAME}
)

install(DIRECTORY
  resource
  launch
  DESTINATION share/${PROJECT_NAME}
)

install(PROGRAMS
  scripts/rqt_example
  DESTINATION lib/${PROJECT_NAME}
)
```

- 스크립트 폴더 및 파일 생성

: 이 스크립트 파일이 RQt의 진입코드라고도 볼 수 있으며 rqt_gui의 main모듈의 Main 클래스를 이용하여 RQt의 플러그인 기능을 사용할 수 있게 되고, 우리가 작성한 메인 코드인 rqt_example의 examples 모듈의 Examples 클래스를 호출하게 됨

rqt_example/scripts/rqt_example

<https://github.com/robotpilot/ros2-seminar-examples>

```
import sys

from rqt_gui.main import Main

from rqt_example.examples import Examples

plugin = 'rqt_example.examples.Examples'
main = Main(filename=plugin)
sys.exit(main.main(standalone=plugin))
```

RQt 예제의 구성

- 리소스 폴더 및 UI 파일 생성

```
rqt_example/resource/rqt_example.ui  
  
https://github.com/robotpilot/ros2-seminar-examples
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<ui version="4.0">  
  <class>examples_widget</class>  
  <widget class="QWidget" name="examples_widget">  
    <property name="geometry">  
      <rect>  
        <x>0</x>  
        <y>0</y>  
        <width>640</width>  
        <height>480</height>  
      </rect>  
    </property>  
    <property name="windowTitle">  
      <string>RQT Example</string>  
    </property>  
    <widget class="QGroupBox" name="group_box_ur">  
      <property name="geometry">  
        <rect>  
          <x>330</x>  
          <y>30</y>
```

- 소스 폴더 및 파일 생성

```
rqt_example/src/rqt_example/__init__.py  
rqt_example/src/rqt_example/examples.py  
rqt_example/src/rqt_example/examples_widget.py  
  
https://github.com/robotpilot/ros2-seminar-examples
```

- 런치 폴더 및 런치 파일 생성
- : turtlesim 패키지의 turtlesim_node 노드와 연동하는 테스트를 위해 작성한 파일

```
rqt_example/launch/turtlesim.launch.py  
  
https://github.com/robotpilot/ros2-seminar-examples
```

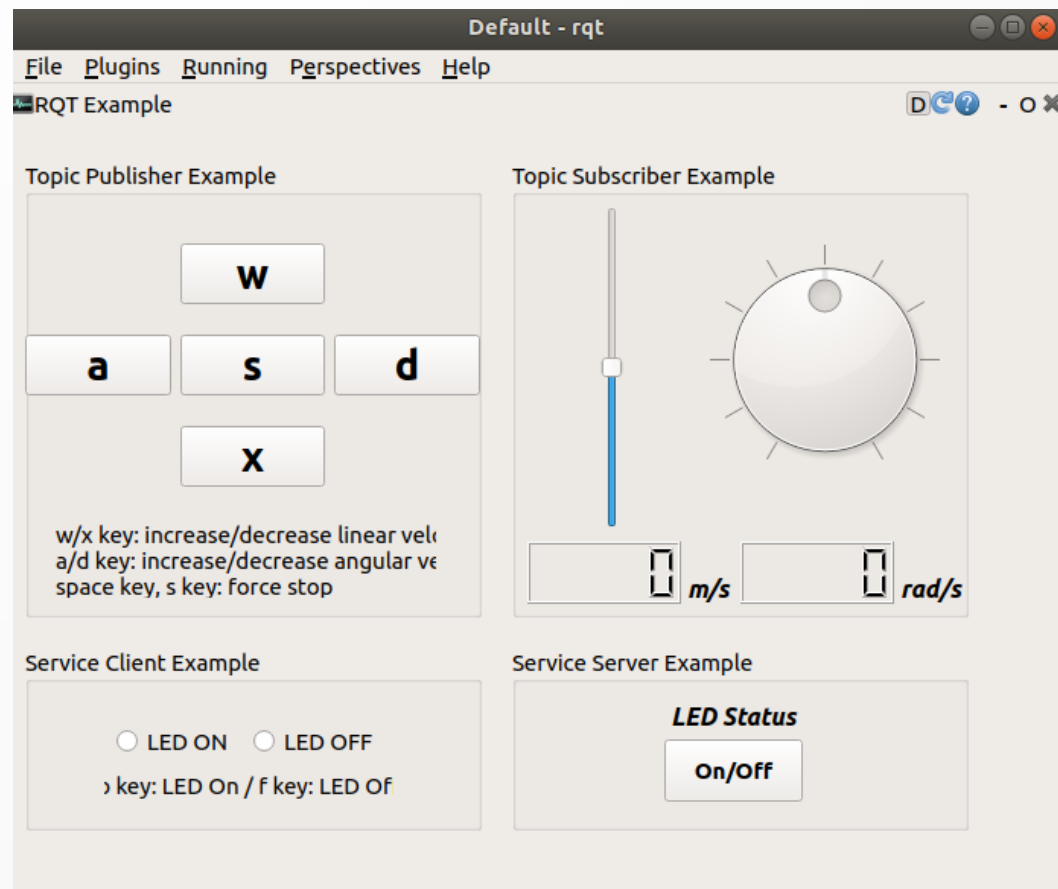
```
from launch import LaunchDescription  
from launch.actions import LogInfo  
from launch_ros.actions import Node  
  
def generate_launch_description():  
    return LaunchDescription([  
        LogInfo(msg=['Execute the rqt_example with turtlesim node.']),
```

RQt 예제의 구성

- RQt 플러그인 예제 실행

```
$ ros2 run rqt_example rqt_example
```

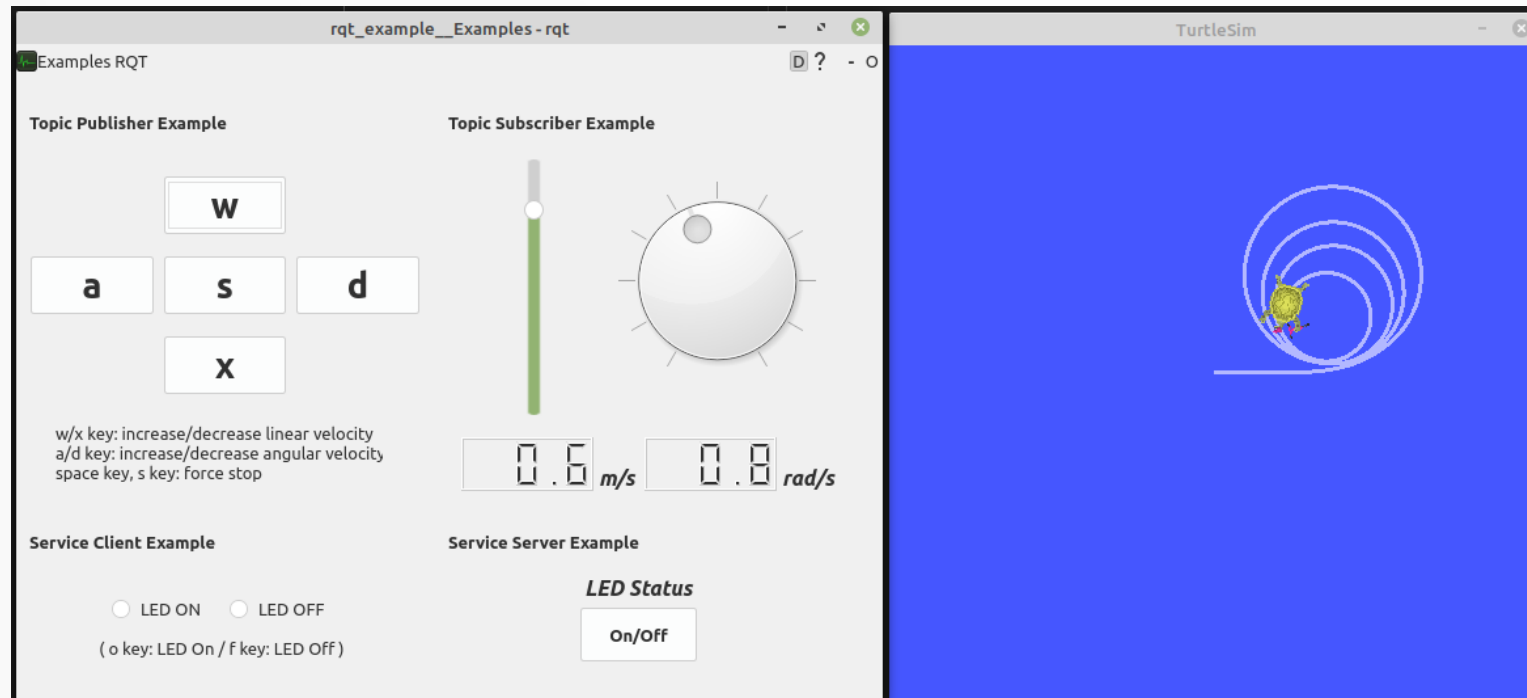
참고로 위 예제나 새롭게 작성한 신규 RQt 플러그인을 실행할 때 'qt_gui_main() found no plugin matching "xxxx"' 등의 에러로 인해 실행이 되지 않는 경우가 있다면 'rqt --force-discover' 명령어를 이용하여 플러그인 찾기를 강제로 수행해주거나 'rm ~/.config/ros.org/rqt_gui.ini' 명령어로 설정 파일을 삭제하며 된다.



RQt 예제의 구성

- 다른 노드와 Rqt 플러그인 연동 예제 실행

```
lch@lch-TB250-BTC:~/robot_ws$ ros2 launch rqt_example turtlesim.launch.py
Failed to load entry point 'env': cannot import name 'add_subparsers_on_demand'
[INFO] [launch]: All log files can be found below /home/lch/.ros/log/2021-11-06-16-56-28-493823-lch-TB250-BTC-16999
[INFO] [launch]: Default logging verbosity is set to INFO
[ERROR] [launch]: Caught exception in launch (see debug for traceback): __init__
() missing 1 required keyword-only argument: 'node_executable'
lch@lch-TB250-BTC:~/robot_ws$
lch@lch-TB250-BTC:~/robot_ws$
```

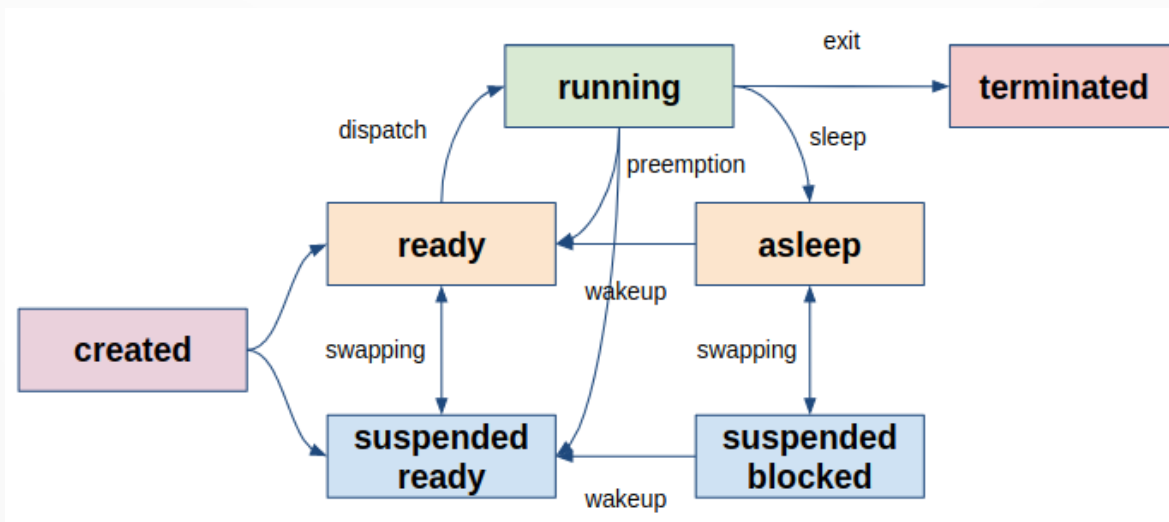


7장 Lifecycle

Lifecycle

- 노드 관리

- 운영체제는 복수개의 프로세스를 효율적으로 관리하기 위해 프로세스의 상태를 정의하고, 상태의 전환을 조율함
- ROS2에서는 Node의 상태를 관리할 수 있는 인터페이스인 Lifecycle을 제공함
- Lifecycle을 통해 상태를 확인하고, 런타임에서 노드를 재실행하거나 교체할 수 있음
- 각 노드의 여러 동작은 개발자가 만든 코드를 통해 처리할 수도 있지만, Lifecycle을 통해 통일된 인터페이스를 사용함으로써 ROS에 등록된 복수개의 노드를 동일한 방법으로 제어할 수 있음



Process states

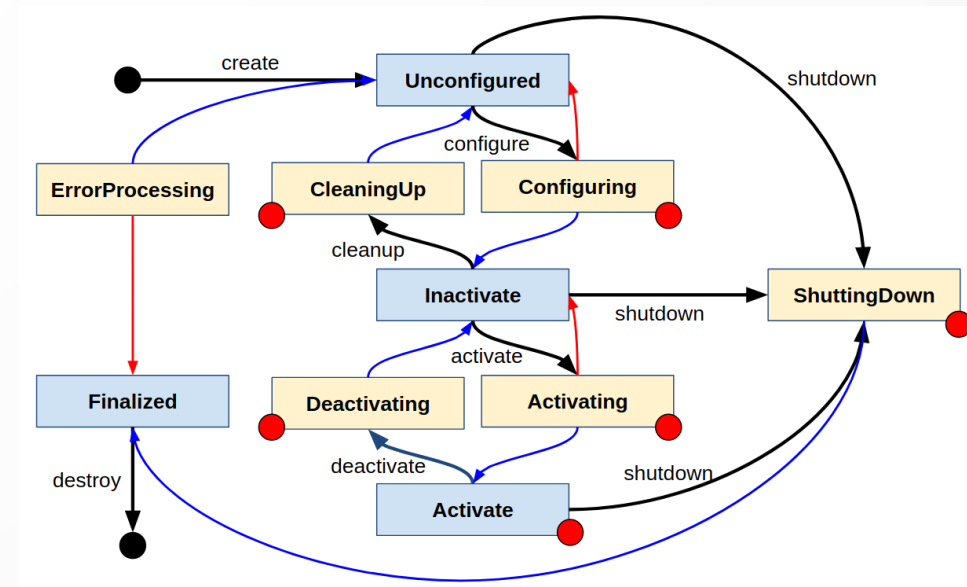
ROS2 Lifecycle

- Lifecycle 주요 상태 4가지

- Unconfigured : 노드가 생성된 직후의 상태, 에러 발생 이후 다시 조정될 수 있는 상태
- Inactivate : 노드가 동작을 수행하지 않는 상태. 파라미터 등록, 토픽 발간과 구독 추가 삭제 등을 (재)구성 할 수 있는 상태
- Activate : 노드가 동작을 수행하는 상태.
- Finalized : 노드가 메모리에서 해제되기 직전 상태. 노드가 파괴되기 전 디버깅이나 내부 검사를 진행할 수 있는 상태

- 6개의 전환 상태

- Configuring : 노드를 구성하기 위해 필요한 설정 수행
- CleaningUp : 노드가 처음 생성되었을 때 상태와 동일하게 만드는 과정 수행
- Activating : 노드가 동작을 수행하기 전 마지막 준비 과정 수행
- Deactivating : 노드가 동작을 수행하기 전으로 돌아가는 과정 수행
- ShuttingDown : 노드가 파괴되기 전 필요한 과정 수행
- ErrorProcessing : 사용자 코드가 동작되는 상태에서 발생하는 에러를 해결하기 위한 과정 수행



ROS2 Lifecycle

- Create
- Configure
- Cleanup
- Activate
- Deactivate
- Shutdown
- Destroy

데모 코드

- ROS2 Foxy 버전, rclcpp을 통해서만 사용 가능
 - lc_talker : Lifecycle이 적용된 노드
 - lc_listener : lc_talker에서 발간하는 정보와 주요 상태를 구독하는 노드
 - lc_client : lc_talker의 현재 주요 상태를 확인하고, 다른 주요 상태로 전환시켜주는 노드
 - - <https://index.ros.org/p/lifecycle/>
 - <https://github.com/ros2/demos/tree/master/lifecycle>
 - lc_talker와 lc_listener를 단독으로 실행시켰을 때는 터미널 창에 아무런 로그를 확인할 수 없고, lc_client 노드를 실행시키면 그때 부터 lc_talker의 상태가 configure → inactive → activate → inactivate → activate → inactivate → finalized 순서로 전환되어 아래와 같은 로그 확인 가능함
 - lc_client 노드는 서비스 통신을 통해 lc_talker의 상태를 전환하고, 그 전환 결과를 보여줌
 -

데모 코드

lc_talker

```
[INFO]: on_configure() is called.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: Lifecycle publisher is currently inactive. Messages are not published.
[WARN]: Trying to publish message on the topic '/lifecycle_chatter', but the publisher is currently inactive.
[INFO]: on_activate() is called.
[INFO]: Lifecycle publisher is active. Publishing: [Lifecycle HelloWorld #10]
[INFO]: Lifecycle publisher is active. Publishing: [Lifecycle HelloWorld #11]
[INFO]: Lifecycle publisher is active. Publishing: [Lifecycle HelloWorld #12]
[INFO]: Lifecycle publisher is active. Publishing: [Lifecycle HelloWorld #13]
```

lc_listener

```
[INFO]: notify callback: Transition from state unconfigured to configuring
[INFO]: notify callback: Transition from state configuring to inactive
[INFO]: notify callback: Transition from state inactive to activating
[INFO]: notify callback: Transition from state activating to active
[INFO]: data_callback: Lifecycle HelloWorld #10
[INFO]: data_callback: Lifecycle HelloWorld #11
[INFO]: data_callback: Lifecycle HelloWorld #12
[INFO]: data_callback: Lifecycle HelloWorld #13
[INFO]: data_callback: Lifecycle HelloWorld #14
[INFO]: data_callback: Lifecycle HelloWorld #15
[INFO]: data_callback: Lifecycle HelloWorld #16
[INFO]: data_callback: Lifecycle HelloWorld #17
[INFO]: data_callback: Lifecycle HelloWorld #18
[INFO]: notify callback: Transition from state active to deactivating
[INFO]: notify callback: Transition from state deactivating to inactive
[INFO]: notify callback: Transition from state inactive to activating
[INFO]: data_callback: Lifecycle HelloWorld #29
[INFO]: notify callback: Transition from state activating to active
[INFO]: data_callback: Lifecycle HelloWorld #30
[INFO]: data_callback: Lifecycle HelloWorld #31
[INFO]: data_callback: Lifecycle HelloWorld #32
```

lc_client

```
[INFO]: Transition 1 successfully triggered.
[INFO]: Node lc_talker has current state inactive.
[INFO]: Transition 3 successfully triggered.
[INFO]: Node lc_talker has current state active.
[INFO]: Transition 4 successfully triggered.
[INFO]: Node lc_talker has current state inactive.
[INFO]: Transition 3 successfully triggered.
[INFO]: Node lc_talker has current state active.
[INFO]: Transition 4 successfully triggered.
[INFO]: Node lc_talker has current state inactive.
[INFO]: Transition 2 successfully triggered.
[INFO]: Node lc_talker has current state unconfigured.
[INFO]: Transition 5 successfully triggered.
[INFO]: Node lc_talker has current state finalized.
```

- Lifecycle 인터페이스는 navigation2[8], moveit2[9] 와 같이 여러 노드로 이루어진 패키지에서 실제 사용 중에 있음
- 초기 로봇 개발 단계에서는 노드의 갯수가 적어서 그 관리가 어렵지 않지만 점점 기능이 추가되면서 노드가 많아지면 어느 순간 시스템 오류로 인해 로봇이 제대로 동작하지 않는 경우가 많아짐.
- 이때 해당 인터페이스를 사용하면 복수개의 노드를 동일하게 관리할 수 있어서 시스템 오류를 예방하거나 디버깅하기 쉬워짐

8장 Security

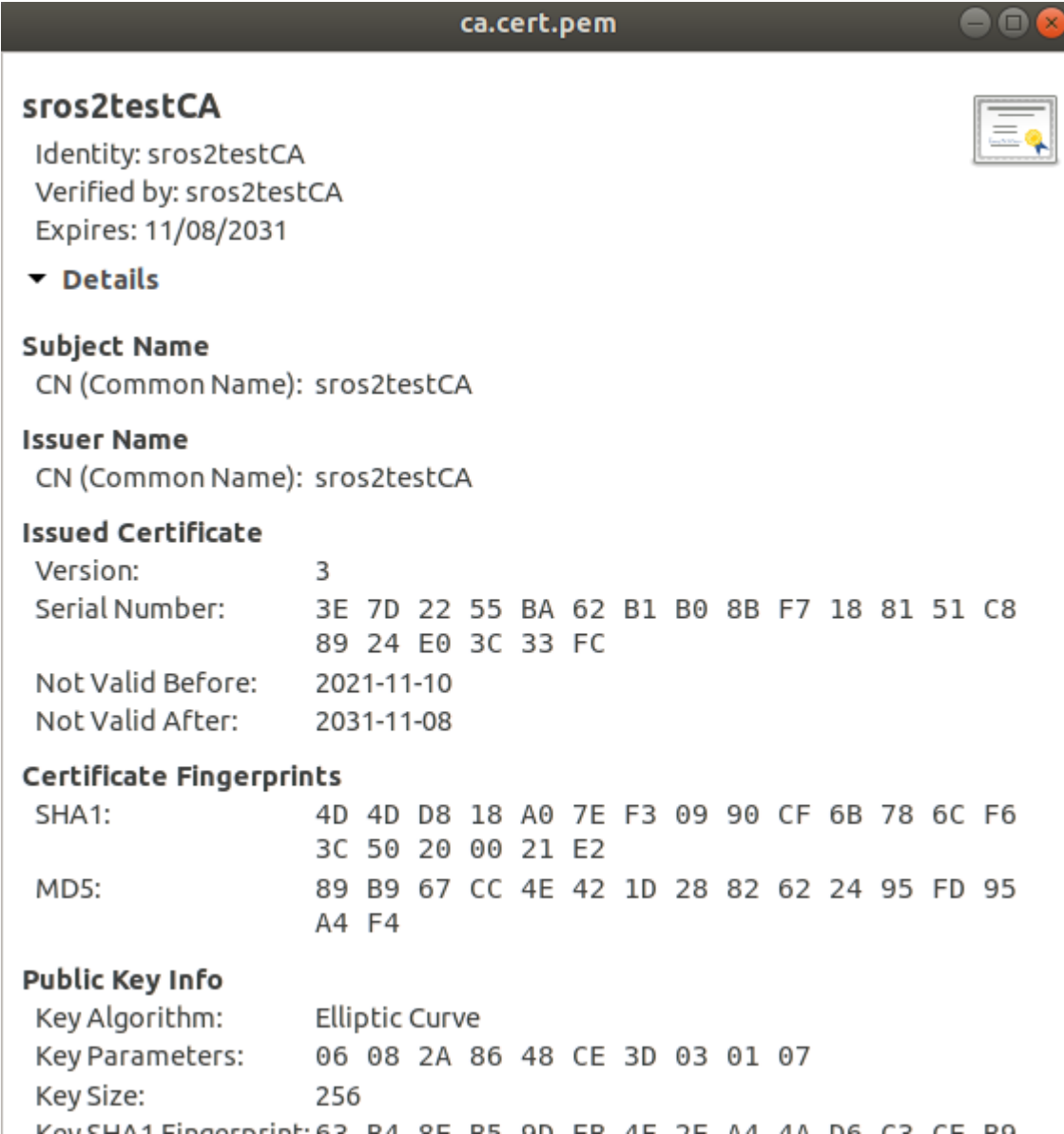
Security

- ROS1의 보안 기능 부재가 항상 문제로 거론되었음
 - ROS1의 경우, ROS master의 IP 주소와 연결 포트만 알아내면 노드 간에 주고 받는 모든 메시지를 들여다 볼 수 있고, 메시지의 조작도 가능하고, ROS master 혹은 특정 노드를 강제 종료하는 방법 등으로 시스템을 죽일 수도 있음
- ROS2에서는 디자인 설계부터 보안 기능이 고려되어 개발됨
 - ROS1의 TCP 기반의 통신은 DDS(Data Distribution Service)로 대체됨
 - SROS의 후속작인 SROS2를 개발하여 보안 관련 ROS 클라이언트 라이브러리를 지원함
 - ROS2CLI 강좌에서 설명한 'ros2 security' 명령어가 SROS2 유틸리티임

노드 인증

- ROS2 네트워크에 참가하는 노드들을 인증하여 참여 불가 판정을 하는 것으로 공개키(public key) 및 개인키(private key)가 요구됨
- 'ROS2 security' 명령어를 사용하여 보안 설정을 하게 됨.
- 암호키 및 보안 파일과 'ROS2 security' 명령어를 사용하여 보안 설정을 하게 됨.

```
$ cd ~/robot_ws
$ ros2 security create_keystore key_box
creating keystore: key_box
creating new CA key/cert pair
creating governance file: key_box/enclaves/governance.xml
creating signed governance file: key_box/enclaves/governance.p7s
all done! enjoy your keystore in key_box
cheers!
```



The screenshot shows a terminal window titled 'ca.cert.pem'. It displays the details of a certificate named 'sros2testCA'. The certificate is issued by 'sros2testCA' and expires on 11/08/2031. The subject name is 'sros2testCA' (Common Name). The issuer name is also 'sros2testCA' (Common Name). The certificate is issued on 2021-11-10 and is valid until 2031-11-08. The certificate is of version 3. The serial number is 3E 7D 22 55 BA 62 B1 B0 8B F7 18 81 51 C8 89 24 E0 3C 33 FC. The SHA1 fingerprint is 4D 4D D8 18 A0 7E F3 09 90 CF 6B 78 6C F6 3C 50 20 00 21 E2. The MD5 fingerprint is 89 B9 67 CC 4E 42 1D 28 82 62 24 95 FD 95 A4 F4. The public key algorithm is Elliptic Curve, with parameters 06 08 2A 86 48 CE 3D 03 01 07 and a key size of 256.

sros2testCA

Identity: sros2testCA
Verified by: sros2testCA
Expires: 11/08/2031

▼ **Details**

Subject Name
CN (Common Name): sros2testCA

Issuer Name
CN (Common Name): sros2testCA

Issued Certificate

Version: 3
Serial Number: 3E 7D 22 55 BA 62 B1 B0 8B F7 18 81 51 C8 89 24 E0 3C 33 FC
Not Valid Before: 2021-11-10
Not Valid After: 2031-11-08

Certificate Fingerprints

SHA1: 4D 4D D8 18 A0 7E F3 09 90 CF 6B 78 6C F6 3C 50 20 00 21 E2
MD5: 89 B9 67 CC 4E 42 1D 28 82 62 24 95 FD 95 A4 F4

Public Key Info

Key Algorithm: Elliptic Curve
Key Parameters: 06 08 2A 86 48 CE 3D 03 01 07
Key Size: 256
Key SHA1 Fingerprint: 62 B4 8E B5 0D EB 4E 2E A4 4A D6 C2 CE B0

노드 인증

- 노드 인증 기능을 테스트할 talker 노드와 listener 노드를 위해 암호키와 인증서를 생성함

```
$ cd ~/robot_ws  
$ ros2 security create_key key_box /talker_listener/talker  
$ ros2 security create_key key_box /talker_listener/listener
```

- 'export' 명령으로 SROS2 환경 변수를 선언하여 ROS2 보안 설정을 함.
- 1) 'ROS_SECURITY_KEYSTORE': 보안 설정 파일을 보관하는 폴더를 지정하는 것으로 위에서 'create_keystore'으로 지정한 폴더의 경로를 넣어줌
- 2) 'ROS_SECURITY_ENABLE': 보안 설정의 On/Off 기능으로 true/false 형태로 설정. 디폴트 설정 값은 false로 지금까지 이 설정을 사용하지 않았다면 ROS2의 Security 기능을 사용하고 있지 않았던 것임
- 3) 'ROS_SECURITY_STRATEGY': 보안 설정 방법에 대한 것으로 Enforce로 설정하게 되면 보안 설정 파일이 없는 메시지 통신은 금지하고 Permissive의 경우 비보안 참여자도 참여 시킴. 보안 설정 파일을 찾을 수 없는 경우를 대비하여 Permissive이 아닌 Enforce 옵션을 사용.

```
$ export ROS_SECURITY_KEYSTORE=~/robot_ws/key_box  
$ export ROS_SECURITY_ENABLE=true  
$ export ROS_SECURITY_STRATEGY=Enforce
```


노드 인증

- 예제로 talker 노드와 listener 노드를 실행
 - Enclave라는 ROS arguments를 이용하여 각 노드를 위해 생성한 암호키와 인증서를 담은폴더를 지정함
 - 2개의 노드가 인증되어 접속되어 메시지를 주고 받는 것 확인 가능함

```
$ ros2 run demo_nodes_cpp talker --ros-args --enclave /talker_listener/talker
```

```
$ ros2 run demo_nodes_py listener --ros-args --enclave /talker_listener/listener
```


9장 Real-time

9장 Real-time

- Real-time의 정의
 - 정해진 시간 안에 입력에 대한 정확한 출력을 보장하는 시스템 실시간 시스템이라 함.
 - Hard real-time system : 엄격한 데드라인 (항공센서, 자율조정시스템, 우주선 등)
 - Firm real-time system : 출력이 정해진 데드라인 이내에 보장되어야 하지만, 그렇지 않더라도 큰 문제가 없는 시스템 (화상회의, 금융 예보 시스템 등)
 - Soft real-time system : 데드라인이 명확하지 않음. 데드라인 이후에 출력을 받아 보더라도 그 동작에 문제가 없는 시스템 (웹 브라우징, 티켓 예매 등)

Real-time programming

- Real time 프로그래밍의 일반적인 패턴

```
init() // non real-time safe
{
    // Allocate memory
    memory.new();

    // Start threads
    threads.start();
}

thread() // real-time safe
{
    // Loop
    while (...)
}

shutdown() // non real-time safe
{
    // End threads
    threads.end();

    // Deallocate memory
    memory.delete();
}
```

파트 1) 실시간을 보장하지 않는 부분
- 메모리 할당
- 스레드 시작

파트 2) 실시간을 보장하는 부분
- 실시간성이 필요한 연산

파트 3) 실시간을 보장하지 않는 부분
- 할당된 메모리를 반납

ROS2 Real-time

- 실시간성은 ROS2의 특정 기능을 사용한다고 되는 것은 아님
- 개발자의 컴퓨터공학적 지식과 하드웨어 개발 환경에 더 밀접하게 관련되어 있음
- ROS2는 rcl과 rmw에 별도의 추상화 라이브러리를 이용하여 사용자 정의 메모리 할당자를 설정할 수 있도록 되어있음
- 데모 코드
 - 실시간 프로그래밍 데모 pendulum_control 패키지

```
$ . ~/ros2_foxy/install/local_setup.bash
$ cd ~/ros2_foxy/install/pendulum_control/bin
$ ./pendulum_demo > output.txt
Couldn't set scheduling priority and policy: Operation not permitted
mlockall failed: Cannot allocate memory
Couldn't lock all cached virtual memory.
Pagefaults from reading pages not yet mapped into RAM will be recorded.
No results filename given, not writing results
```

Output.txt 보면 fault가 1건 있음

```
$ cat output.txt
Initial major pagefaults: 172
Initial minor pagefaults: 4115
rttest statistics:
  - Minor pagefaults: 1
  - Major pagefaults: 0
Latency (time after deadline was missed):
  - Min: 56179 ns
  - Max: 170799 ns
  - Mean: 129188.509000 ns
  - Standard deviation: 40351.343055

PendulumMotor received 498 messages
PendulumController received 946 messages
```

ROS2 Real-time

- 데모 코드

- 페이지 폴트 횟수를 줄이기 위해 RAM에 저장할 수 있는 메모리 크기에 대한 제한을 풀어줘야 함.
- 관리자 권한으로 /etc/security/limits.conf 파일 가장 아랫줄에 memlock 옵션({username} - memlock - 1)을 추가 후 재부팅 이후, 아래 명령어 입력, 이후 재부팅 재실행

```
$ ulimit -l unlimited
```

Output.txt에서 fault가 사라짐

```
$ cat output.txt
Initial major pagefaults: 0
Initial minor pagefaults: 2124268
rttest statistics:
- Minor pagefaults: 0
- Major pagefaults: 0
Latency (time after deadline was missed):
- Min: 53620 ns
- Max: 177322 ns
- Mean: 146186.291000 ns
- Standard deviation: 17348.023064

PendulumMotor received 519 messages
PendulumController received 958 messages
```

- 실시간성을 위해서는 개발자가 개발하는 시스템에서 실시간성이 필요한 부분을 명확히 캐치하고, 해당 부분의 메모리 관리와 프로세스의 스케줄링 등을 고려할 수 있어야 함.