ROS Programming

ROS 프로그래밍

05 ROS 프로그래밍 5

목차 ROS 프로그래밍

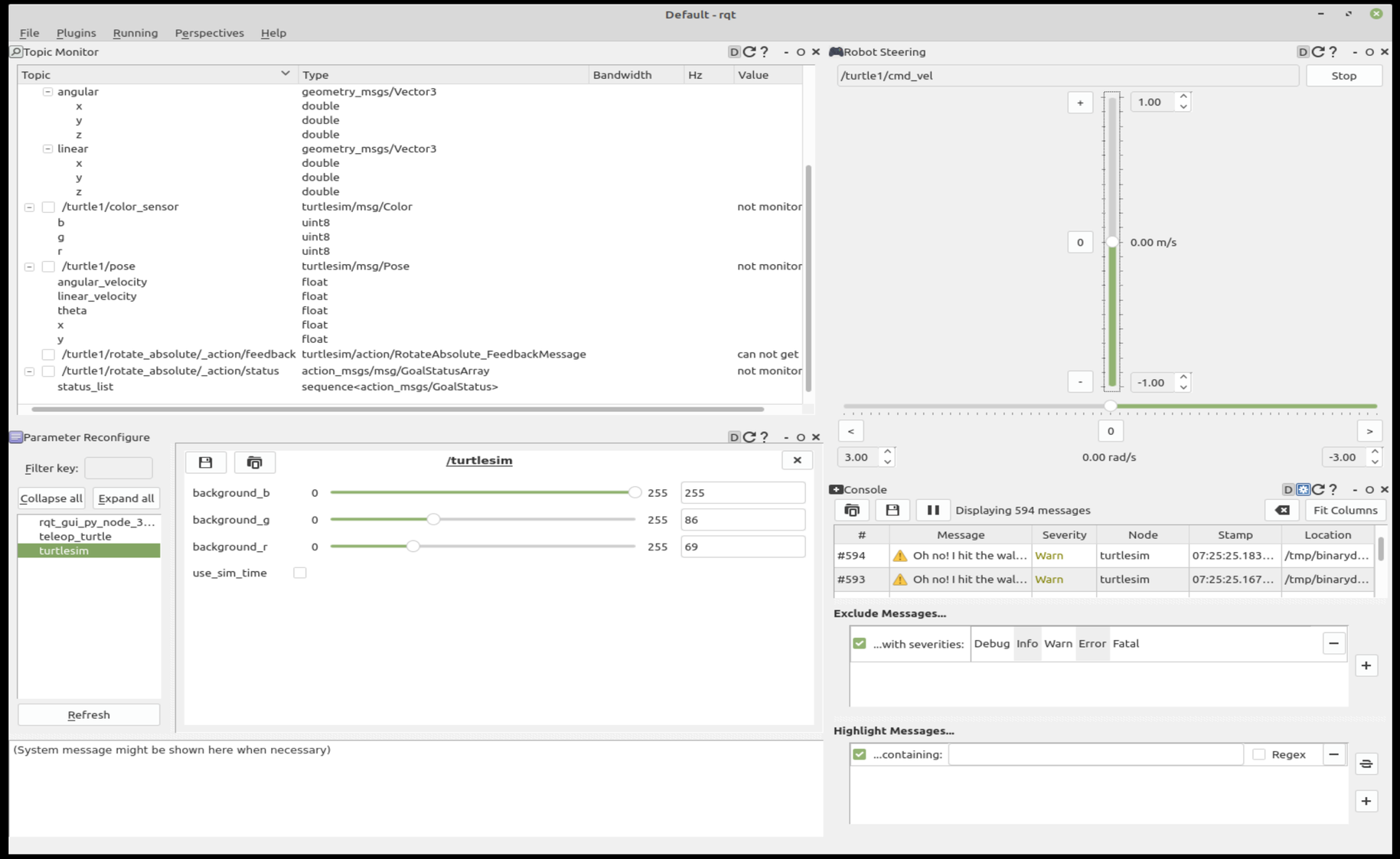
1. RQt plugin
2. Life cycle
3. Security
4. Real-time

1. RQt plugin

RQt 플러그인 (RQt plugin)

- ROS용 GUI 툴 개발은 RQt 플러그인이 아닌 일반적인 GUI 형태로 처음부터 자신 만의 GUI를 구현할 수 있음
- RQt 스타일 즉, RQt 플러그인 형태로 개발하게 되면 GUI에 대한 표준화된 공통 절차인 GUI의 시작과 종료 처리나 GUI툴을 사용하면서 각종 옵션을 종료시점에서 저장하고 시작 시 다시 불러오는 상태 복원 기능, 단일 창에 여러 위젯을 불러와 락킹하는 고정 기능 등을 RQt 플러그인에서 제공하는 API로 비교적 쉽게 구현이 가능
- 시스템 아키텍처의 관점에서 보더라도 다중 플랫폼 및 다중 언어(C++, Python) 지원 등의 특징을 살릴 수 있고 ROS 커뮤니티에서 공개된 다양한 GUI 기능들을 쉽게 이식하여 사용할 수 있다는 장점

RQt 플러그인 (RQt plugin)



RQt 플러그인 관련 ROS2 패키지

- **rqt 패키지**: RQt의 메타패키지로 RQt에서 사용되는 공용 rqt_gui, rqt_gui_cpp, rqt_gui_py, rqt_py_common 패키지가 포함
- **rqt_gui 패키지**: 여러 'rqt' 위젯을 단일 창에 도킹 할 수 있는 위젯 패키지
- **rqt_gui_cpp 패키지**: C++ 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API를 제공하는 패키지
- **rqt_gui_py 패키지**: Python 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API를 제공하는 패키지

RQt 플러그인 관련 ROS2 패키지

- **rqt_py_common 패키지**: Python으로 작성된 RQt 플러그인에서 공용으로 사용되는 기능을 모듈을 제공하는 패키지로 rqt_common_plugins 에서 사용
  - https://github.com/ros-visualization/rqt/tree/crystal-devel
- **rqt_common_plugins 패키지**: rqt_action, rqt_bag 등 20여개의 RQt 플러그인을 포함하고 있는 메타패키지
  - https://github.com/ros-visualization/rqt_common_plugins/tree/ros2
- **qt_gui_core 패키지**: qt_gui, qt_gui_cpp, qt_gui_py_common, qt_gui_app, qt_dotgraph 등을 담은 메타패키지
  - https://github.com/ros-visualization/qt_gui_core/tree/foxy-devel

python_qt_binding 패키지

- **python_qt_binding 패키지**: QtCore, QtGui, QtWidgets 등을 사용할 때 Python 언어 기반의 Qt API를 제공하는 바인딩 패키지

  - https://github.com/ros-visualization/python_qt_binding/tree/crystal-devel

- Python 언어로 Qt API를 사용하려면 본래의 Qt C++ API가 아닌 Python 언어로 바인딩된 Qt Python API를 사용해야하는데 대표적인 Qt Python API으로는 **PyQt, PySide**가 있음
- Qt Python API를 직접 사용해도 되지만 python_qt_binding 패키지를 사용하면 이 두 가지 바인딩 API의 구분없이 사용할 수 있고 원한다면 쉽게 바꾸어 사용할 수 있음

python_qt_binding 패키지

- Qt C++ API를 Qt Python API로 바인딩할 때 사용되는 관련된 툴은 각 공급자가 제공하는데 PyQt의 경우 이를 'SIP'라고 하며 PySide의 경우 'Shiboken'
- pyQt는 GPL 라이선스를 사용하고 있고 PySide는 LGPL 라이선스를 사용
- python_qt_binding 패키지를 사용함에 따라 이 라이선스로부터도 자유롭게 된다. 현재 ROS2 Foxy에서 사용되는 버전은 PyQt는 **PyQt5**이며 PySide는 **PySide2**를 사용

python_qt_binding 패키지

- RQt 플러그인 패키지들의 사용 순서대로 나열하자면 제작하고자 하는 RQt 플러그인의 최상위 클래스에서 rqt_gui_py.plugin 모듈의 Plugin 클래스를 Python plugins 의 인터페이스로 상속하고 qt_gui.plugin 모듈의 Plugin 클래스를 상속.
- 그 뒤 python_qt_binding.QtCore 모듈의 QObject 클래스를 상속함으로써 Qt 기반의 ROS GUI 기능을 프로그래밍할 수 있게 됨

  1) rqt_gui_py.plugin 모듈 Plugin 클래스

  2) qt_gui.plugin 모듈의 Plugin 클래스

  3) python_qt_binding.QtCore 모듈의 QObject 클래스

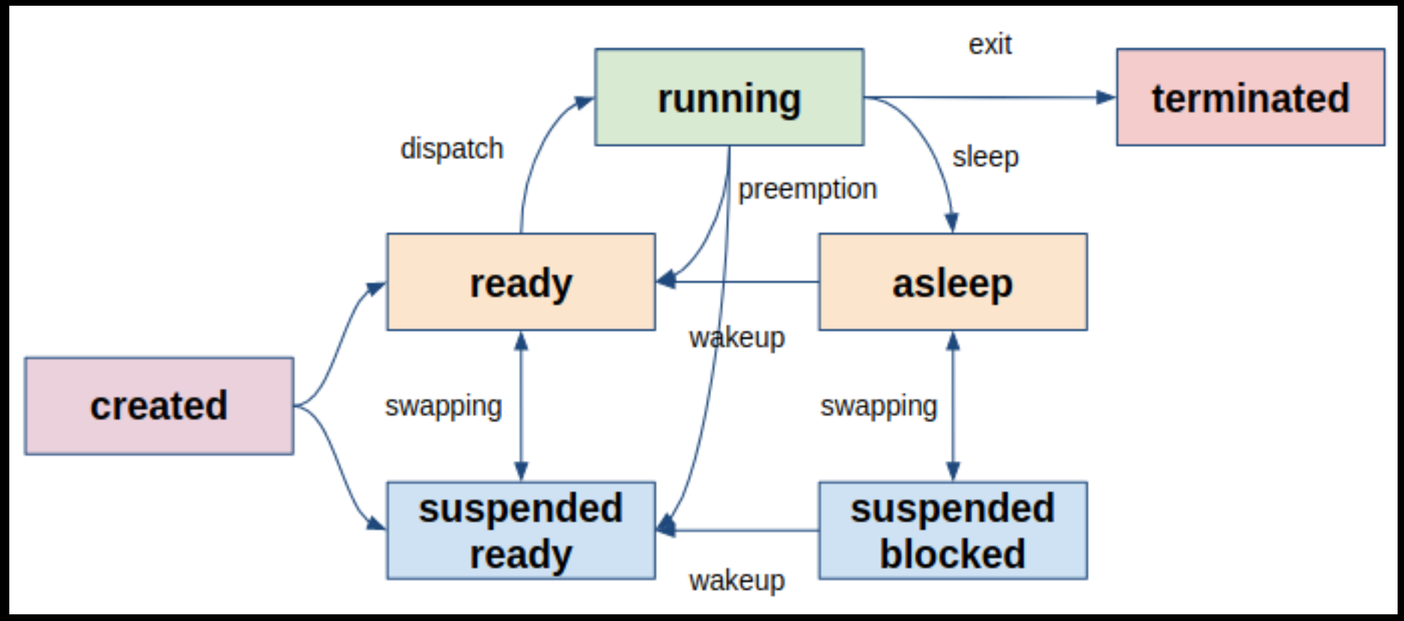
RQt 플러그인 작성 순서

- RQt 플러그인 형태로 개발하는 것이기에 RQt 플러그인의 기본 기능 관련과 GUI 관련 패키지는 의존성 패키지로 포함
- 한가지 특이한 설정으로는 이 패키지는 Python 언어로 작성하지만 RQt 플러그인의 일부로 작성될 예정이기에 빌드 형태를 'ament_cmake'으로 해줘야한다는 것

2. Lifecycle

노드 관리

- 운영체제는 복수개의 프로세스를 효율적으로 관리하기 위해 프로세스의 상태를 정의하고 상태의 전환을 조율한다. 프로세스의 상태는 프로세서, 메모리와 같은 자원의 할당 여부에 따라 정의되고 처리 순서, 교착 상태, 메모리 할당 등에 의해 그 상태가 전환될 수 있음



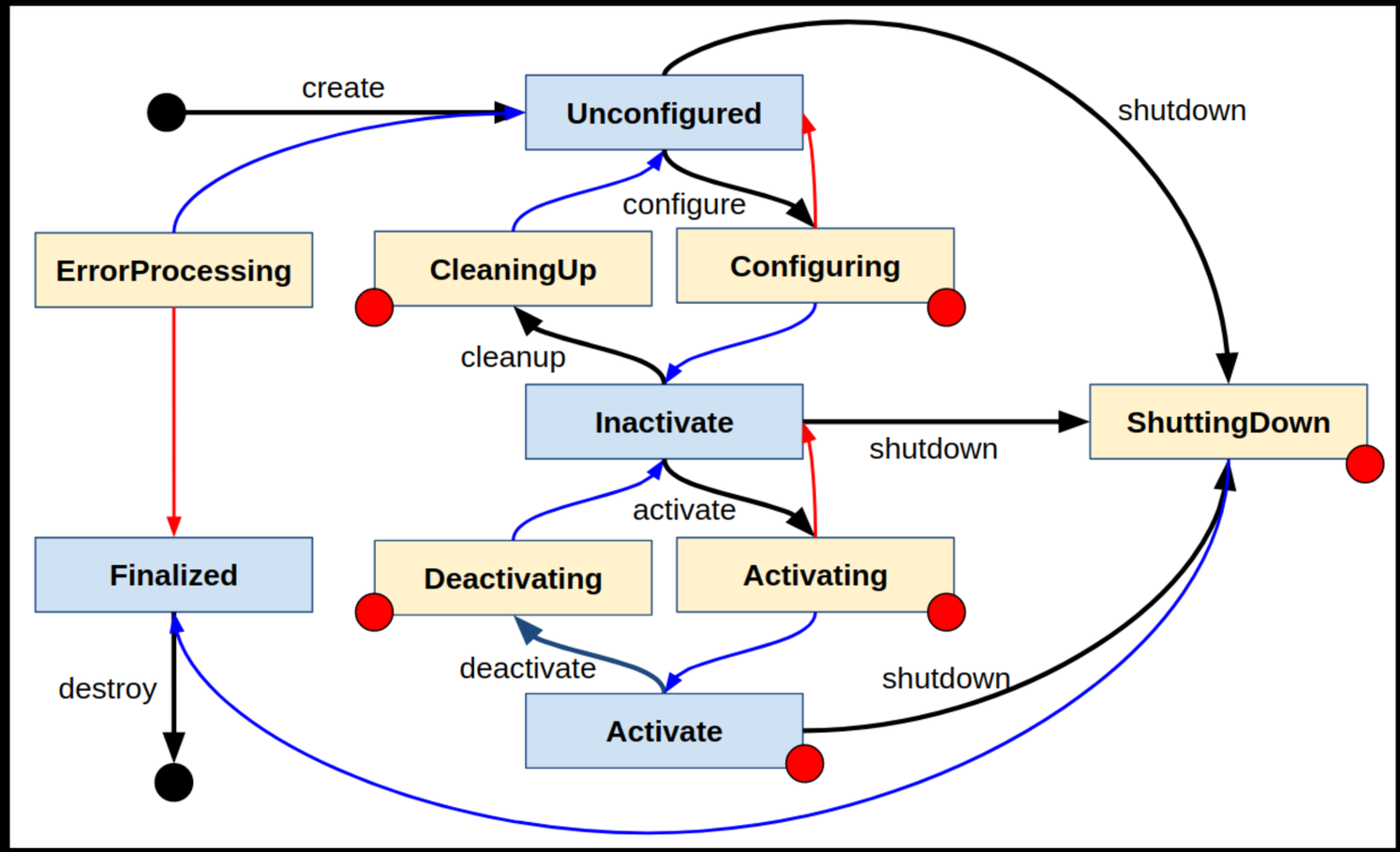
노드 관리

- 카메라 센서를 통해 받은 이미지 정보를 발간하는 노드를 예를 들어 보자. 먼저 노드를 동작시키기 전에 카메라와의 통신을 위한 포트가 제대로 잡혔는지 확인해볼 수 있을 것이고 포트가 제대로 잡히지 않았다면 그 동작을 제한할 수 있을 것
- 만약 노드가 동작되는 도중에 에러가 발생 하였다면 잠시 그 동작을 멈추고 에러를 해결한 다음 재시작할 수 있을 것이고, 주변 환경의 변화로 인해 에러를 해결할 수 없다면 해당 노드는 종료시키고 준비된 다른 노드를 동작시킬 수도 있을 것
- 물론 이와 같은 동작은 개발자가 만든 코드를 통해 처리할 수도 있음. 하지만 Lifecycle 을 통해 통일된 인터페이스를 사용함으로서 ROS 에 등록된 복수개의 노드를 동일한 방법으로 제어할 수 있다는 장점을 가질 수 있음

Lifecycle

- Lifecycle 은 노드를 관리하기 위한 인터페이스로 주요 상태 (Primary states) 와 전환 상태 (Transition states) 그리고 전환 (Transition) 를 제공
- 파란색 박스는 주요 상태, 노란색 박스는 전환 상태를 이야기 한다. 검정색 화살표는 전환을 표기하였고, 파란색 화살표는 전환 상태의 결과가 성공일때 주요 상태의 변화를 나타내고, 빨간색 화살표는 전환 상태의 결과가 실패일때 주요 상태의 변화를 나타낸다. 빨간색 작은 원은 에러가 확인될 수 있는 상태

Lifecycle


create
Unconfigured
shutdown
configure
ErrorProcessing
CleaningUp
Configuring
cleanup
Inactivate
ShuttingDown
shutdown
activate
Finalized
Deactivating
Activating
deactivate
shutdown
destroy
Activate

Lifecycle 주요 상태

- 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

Lifecycle 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

Lifecycle

3. Security

SROS2(Secure Robot Operating System 2)

- SROS의 차기 버전으로 SROS의 컨셉과 목적은 동일하게 계승하고 있지만 ROS2에서 새롭게 도입된 DDS의 확장 기능인 DDS-Security을 ROS 2에서 사용할 수 있도록 새롭게 개발
- 우리가 사용하는 RCL(ROS client library)에 도메인 참가자(domain participant)를 위한 보안 파일 지원, 실행 옵션, 보안 기능 On/Off 등을 포함시켜 놓았기에 사용자는 SROS2에서 제공하는 유틸리티를 'ros2 security' 명령어로 보안 관련 도구(tools)와 지침(policies)을 사용하면됨
- DDS-Security는 DDS 확장 기능의 사양으로 DDS의 기본 사양은 아닌 추가 옵션

SROS2(Secure Robot Operating System 2)

- 기본적으로는 인증(Authentication), 액세스 제어(Access control), 암호화(Cryptography), 로깅(Logging), 데이터 태깅(Data tagging)과 같이 아래 5가지 기능을 담고있다. 이 5가지 기능 중 SROS2에서는 DDS-Security에 대응하고 있는 벤더들의 공통 기능인 1) 인증, 2) 액세스 제어, 3) 암호화 이렇게 3가지 기능을 플러그인 형태로 제공
  - Authentication : DDS 도메인 참가자(domain participant) 확인
  - Access control : 참가자가 수행 할 수있는 DDS 관련 작업에 대한 제한
  - Cryptography : 암호화, 서명, 해싱 처리
  - Logging : DDS 보안 관련 이벤트를 감시하는 기능
  - Data tagging : 데이터 샘플에 태그를 추가하는 기능

인증(Authentication)

- 인증 플러그인에서는 DDS의 도메인 참가자(domain participant) 확인 작업에 PKI(Public Key Infrastructure)를 사용
- 인증은 각 도메인 참가자당 x.509 인증서에 해당되는 공개키(public key) 및 개인키(private key)가 요구되며 각 x.509 인증서는 특정 인증 기관인 CA(Certificate Authority)에 의해 서명되어 있어야 함

* X.509는 공개키 인증서(public key certificates)의 형식을 정의하는 표준으로 X.509 인증서는 웹 검색을 위한 보안 프로토콜인 HTTPS에 사용되는 TLS/SSL을 포함하여 많은 인터넷 프로토콜에서 사용

액세스 제어(Access control)

- 액세스 제어 플러그인에서는 지정된 도메인 참여자의 DDS 관련 기능에 대한 제한을 정의하고 적용.
- 이것은 시스템 관리자가 프로그램 별로 기능을 제한할 수 있게 해주는 리눅스 커널 보안 모듈인 AppArmor(Application Armor)이나 프로세스들의 자원의 사용을 제한하고 격리시키는 리눅스 커널 기능인 cgroups(control groups)은 아님
- 이는 특정 DDS 도메인에 특정 도메인 참가자가 참여하는 것을 제한하거나 참가자가 특정 DDS 토픽을 읽거나 쓸 수 있도록 허용하는 것. 이러한 제어는 XML 문서 형태로 **액세스 제어 방침(policy)**을 지정할 수 있게 되어 있음

암호화(Cryptography)

- 암호화 플로그인에서는 **암호화(encryption), 복호화(decryption), 서명(signing), 해싱(hashing)** 등의 모든 암호화 관련 작업을 담당하며 앞서 설명한 인증과 액세스 제어 플러그인은 암호화 플러그인을 사용

4. Real-time

Real-time의 정의

- Hard real-time system 은 매우 엄격한 데드라인을 가지고 있어서, 만약 한번이라도 데드라인 이내에 입력에 대한 정확한 출력을 받아볼 수 없다면 매우 큰 위험을 초래할 수 있는 시스템(예를 들면, 항공 센서, 자율 조종 시스템, 우주선, 핵 융합 발전소 등)
- Firm real-time system 은 입력에 대한 출력이 정해진 데드라인 이내에 보장되어야 하지만, 만약 그렇지 않더라도 그 동작에 큰 문제가 없는 시스템(화상 회의, 금융 예보 시스템, 자동 조립 라인 등)
- Soft real-time system 은 데드라인이 명확하지 않고, 데드라인 이후에 출력을 받아보더라도 그 동작에 큰 문제가 없는 시스템(예를 들면, 웹 브라우징, 티켓 예매 등)

Real-time programming

- 일반적인 실시간 프로그래밍은 아래와 같은 패턴을 가짐

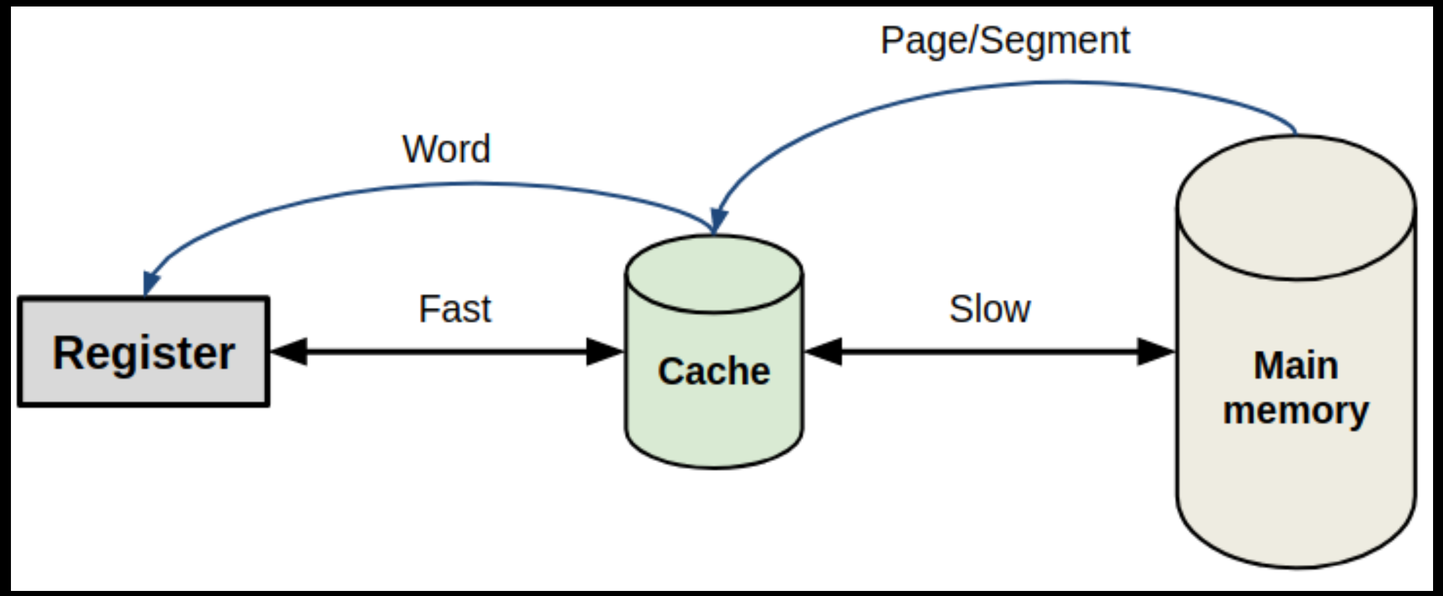
```
init() // non real-time safe
{
  // Allocate memory
  memory.new();

  // Start threads
  threads.start();
}

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

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

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

Real-time의 정의

- 첫번째 파트는 실시간을 보장하지 않는 부분으로 필요한 메모리를 할당, 스레드를 시작
- 두번째 파트는 실시간을 보장하는 부분으로 실시간성이 필요한 연산에 대한 코드가 포함
- 세번째 파트는 실시간을 보장하지 않는 부분으로 할당된 메모리를 반납

Page faults

- 프로그램이 실행될 때 CPU는 연산에 필요한 자원의 위치를 파악하고 해당 자원을 메인 메모리로부터 복사하여 캐시를 통해 레지스터로 할당.
- 레지스터와 캐시는 하드웨어가 관리하는 메모리로 그 속도가 매우 빠르지만 그 크기가 작음. 하지만 메인 메모리는 소프트웨어가 관리하는 자원으로 속도가 느리지만 크기가 큼



Page faults

- 운영체제는 자원의 지역성을 이용하여 메인 메모리로 부터 프로세서가 다음에 접근할 것 같은 자원들을 캐시로 복사하여 캐시히트(Cache hit)를 더 자주 일어나게 만들려고 함
- 하지만 만약 프로세서가 필요로 하는 자원이 캐시에 없게 되면 결국 메인 메모리에 접근하여 다른 자원을 복사해야만 하는데 이를 페이지 폴트(Page faults)라고 함

Page faults

1) mlockall function : 가상 메모리 주소를 RAM에 미리 할당하는 mlockall 함수 사용

2) Dynamic memory pool : 가상 메모리 크기를 고정된 사이즈로 할당하고, 프로세스 런타임 시 해당 메모리를 반납하지 않도록 함

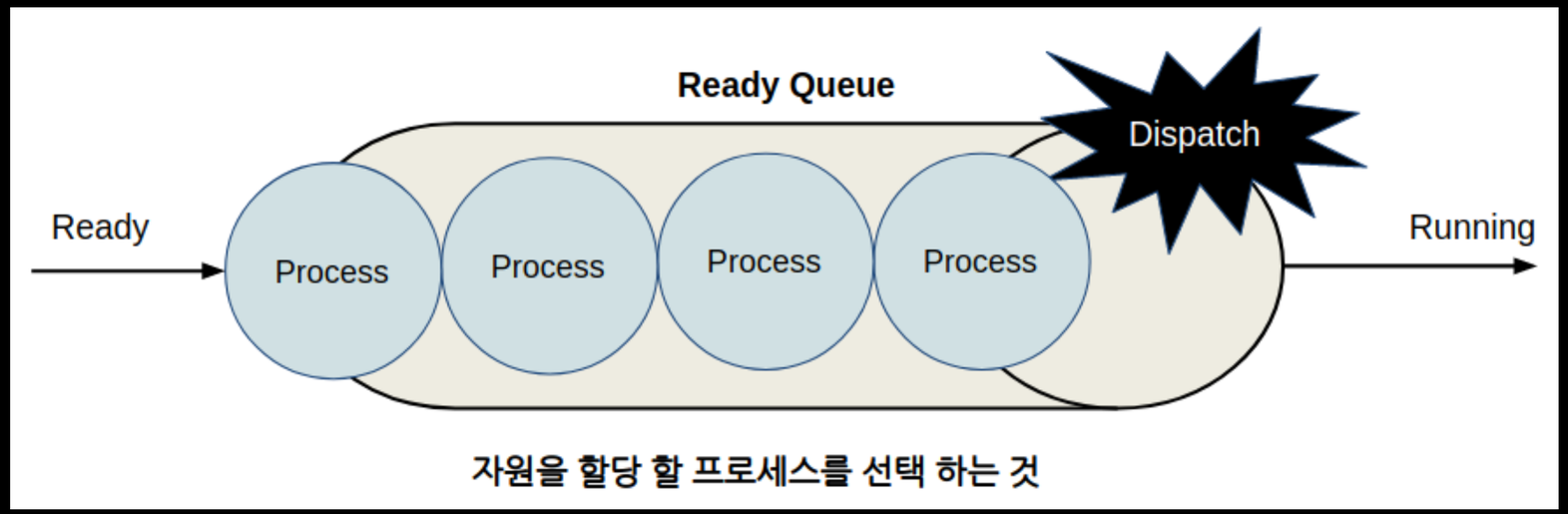
3) Custom real-time safe memory allocators : 기존의 메모리 할당자는 실시간성을 가지기에 어렵기에 사용자가 개발한 메모리 할당자를 사용하도록 함

4) Global variables and (static) arrays : 프로세스 시작시에 전역 변수나 정적 배열을 통해 미리 메모리 공간을 할당하도록 함

5) Cache friendliness for pointer and vtable accesses : 가상 메모리를 사용함에 따른 오버헤드를 줄이기 위한 포인터를 사용하도록 함

Process scheduling

- 운영체제는 여러 프로세스에 대한 효율적인 자원 할당을 통해 멀티 프로그래밍을 지원
- 프로세스 스케쥴링은 메모리를 할당받은 준비 상태의 프로세스가 레디큐에 있을 때 어떤 순서로 이를 프로세서에 할당(Dispatch)하여 동작 상태로 변경할 것인가를 이야기



Real-time의 정의

- 선점 스케줄링(Preemptive scheduling)은 현재 실행중인 프로세스를 인터럽트 하여 다른 프로세스에 자원을 할당
- 비선점 스케줄링(Non-preemptive scheduling)은 프로세스가 자원을 할당 받았을 때 다른 프로세스가 해당 자원을 빼앗을 수 없음
- 우선순위는 정적 우선순위(Static priority)와 동적 우선순위(Dynamic priority)
- 만약 실시간성을 가져야 하는 프로세스가 자원을 빼앗길 수 있거나 우선순위가 낮게 설정된다면 설정된 데드라인 안쪽으로 출력을 만들어내지 못할 것. 따라서 RT_PREEMPT와 같은 커널 패치를 통해 개발자가 프로세스 스케줄링에 임의로 개입할 수 있도록 하고, 프로세스의 우선순위를 높게 설정하여 실시간성을 가질 수 있도록 해야함

Performance benchmarking

- 실시간성을 가지는지 그렇지 않은지를 판단하기 위해서는 페이지 폴트와 스케쥴링 지터 (Scheduling jitter)를 확인하고 이를 개선해나가야 할 것
- 프로그램에 printf와 같은 로그함수를 사용하는 것은 오히려 실시간성을 깨뜨리는 요인

1) cyclictest : 실시간 환경의 스케쥴링 지터를 측정하기 위한 리눅스 명령어. 스레드의 갯수와 우선순위, 스케쥴러 타입을 설정하여 레이턴시와 지터가 얼마나 되는지 확인 가능

2) rttest : ROS 2 에서 지원하는 라이브러리로 사용자 코드에서 발생하는 스케쥴링 지터를 저장하여 알려줌

3) getrusage : 페이지 폴트, 메모리 스왑, I/O 블락 등 실시간성을 나타내는 지표에 대한 통계를 반환

실시간 프로그래밍 데모

- 해당 데모는 ROS 2 Foxy 버전을 소스빌드하여 설치한 후 진행해야만 하고, 실시간성을 위한 커널은 설치하지 않음. 해당 데모가 실시간성을 가지기 위해서는 최소 8Gb RAM 저장 공간을 필요로 하며 이상적으로는 rttest 결과 페이지 폴트가 단 한번도 없어야 하고, 평균 지연이 30,000ns 이하 여야만 함

```
$ . ~/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
```

실시간 프로그래밍 데모

- output.txt에 저장된 로그를 확인해 보면 rttest 결과 페이지 폴트가 한번 있었음을 확인

```
$ 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
```

실시간 프로그래밍 데모

- 페이지 폴트 횟수를 줄이기 위해서는 RAM 에 저장할 수 있는 메모리 크기에 대한 제한을 풀어줘야만함. 관리자 권한으로 /etc/security/limits.conf 파일 가장 아랫줄에 memlock 옵션(${username} - memlock -1)을 추가해주고, 재부팅 이후 아래 명령어도 함께 터미널창에 입력

```
$ ulimit -l unlimited
```

실시간 프로그래밍 데모

- 컴퓨터를 재실행하고 데모코드를 실행하면 앞서 볼 수 있었던 에러 메시지가 출력 X
- output.txt 에 저장된 로그를 열어보면 rttest 결과 에서 페이지 폴트가 완전히 없어진 것도 확인

```
$ 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
```

실시간 프로그래밍 데모

- logger 노드를 이용해서 pendulum_demo 의 런타임 로그를 확인. 새로운 터미널 창을 열어 아래 명령어로 logger 노드를 먼저 실행시키고 다른 터미널 창을 열어서 pendulum demo를 실행. 그러면 logger 노드를 실행시킨 터미널 창에서 로그가 출력되는 것을 확인

```
# Terminal 1
$ . ~/ros2_foxy/install/local_setup.bash
$ cd ~/ros2_foxy/install/pendulum_control/bin
$ ./pendulum_logger
# ... 내용 생략 ...
Commanded motor angle: 1.570796
Actual motor angle: 1.541950
Current latency: 99387 ns
Mean latency: 105697.995992 ns
Min latency: 53414 ns
Max latency: 170257 ns
Minor pagefaults during execution: 0
Major pagefaults during execution: 0

# Terminal 2
$ . ~/ros2_foxy/install/local_setup.bash
$ cd ~/ros2_foxy/install/pendulum_control/bin
$ ./pendulum_demo > output.txt
```

실시간 프로그래밍 데모

- pendulum_demo 노드를 종료한 다음 output.txt에 저장된 실시간성에 대한 결과를 확인

```
$ cat output.txt
Initial major pagefaults: 16
Initial minor pagefaults: 2124638
rttest statistics:
  - Minor pagefaults: 0
  - Major pagefaults: 0
  Latency (time after deadline was missed):
    - Min: 53414 ns
    - Max: 170257 ns
    - Mean: 105750.270000 ns
    - Standard deviation: 33639.231572


PendulumMotor received 553 messages
PendulumController received 967 messages
```

- 페이지 폴트는 나타나지 않지만, 최소 레이턴시와 최대 레이턴시가 크게 차이나는 모습을 확인할 수 있음. 이는 커널 수정 없이 데모 코드를 실행하여서 나타나는 결과로 리눅스 스케줄러의 프로세스 스케쥴링으로 인한 딜레이가 주 원인

실시간 프로그래밍 데모

- 이를 해결하기 위해 관리자 권한으로 /etc/security/limits.conf 파일 가장 아랫줄에 우선순위 옵션 (${username} - rtprio 98)을 추가하여 특정 유저가 실행하는 프로세스의 우선순위를 수정해주고 재부팅
- 우선순위는 0-99까지 설정 가능하지만 가장 높은 우선순위는 중요한 시스템 프로세스가 가지고 있기 때문에 그보다 하나 낮은 98 로 설정

실시간 프로그래밍 데모

- 실시간성의 변화를 쉽게 확인하기 위해 그래프 그리기

```
$ . ~/ros2_foxy/install/local_setup.bash
$ cd ~/ros2_foxy/install/pendulum_control/bin
$ ./pendulum_demo -f pendulum_demo_results
$ rttest_plot pendulum_demo_results
```