Libclang 을 활용한 ROS2 의 스레드 안전성 분석

신채원 [1], 강정환 [2], 권동현 [3]
[1] 부산대학교 정보컴퓨터공학부 학부생
[2] 부산대학교 정보융합공학과 박사과정
[3] 부산대학교 정보컴퓨터공학부 교수 (교신저자)

sinchaewon@pusan.ac.kr, jeonghwan@pusan.ac.kr, kwondh@pusan.ac.kr

Analyzing thread-safety on ROS2 with Libclang

Chae-Won Shin[1], Jeong-Hwan Kang[2], Dong-Hyun Kwon[3]
[1]School of Computer Science and Engineering, Pusan National University
[2]Dept. of Information Convergence Engineering, Pusan National University
[3]School of Computer Science and Engineering, Pusan National University

요 약

ROS2 코드에서, multi-threaded executor 은 여러 메시지 또는 이벤트를 병렬로 처리할 수 있도록 복수의 스레드를 만든다. Multi-threaded executor 을 사용하면 프로그램 성능이 가속화되고 메모리를 경제적으로 사용할 수 있지만, 하나의 시스템 자원을 여러 개의 프로세스가 동시에 참조하거나 수정하기 때문에 deadlock 등의 여러 문제가 발생한다. 이 논문에서는 libclang 인터페이스를 이용해 ROS2 코드를 구문 분석하고 스레드 안전성을 평가하는 방법과 그 한계점, 향후 연구 방향에 대해 논한다.

1. 서론

ROS2 에서, multi-threaded executor 은 여러 메시지 또는 이벤트를 병렬로 처리할 수 있도록 복수의 스레드를 만든다[1]. Multi-threaded executor 을 사용하면 프로세서의 리소스를 얻기 위해 스레드 스위치를 사용하지 않고, 여러 스레드를 독립적으로 실행할 수 있기 때문에 프로그램 성능이 가속화된다. 또한 복수의 스레드가 하나의 메모리 공간을 공유하여 경제적이다[2]. 그러나 스레드 안전성이 보장되지 않은 상태로 multi-threaded executor 을 사용하면 공유된 데이터에 관련한 문제가 발생할 수 있다. 이에 multi-threaded executor 을 사용할 때 스레드 안전성이 보장되는지 확인할 수 있는 방안이 필요하다.

본 논문에서는 ROS2 의 multi-threaded executor 을 사용하지 않는 노드 중 multi-threaded executor 을 사용할 수 있는 노드를 식별하기 위한 방법을 탐색한다.

2. Background

2-1. Libclang

Libclang 은 구문 분석을 통해 C/C++ 코드에 대한 추상 구문 트리(이하 AST)를 생성하고, AST 에 대한 정보를 검색하기 위한 API 함수를 제공하는 C-인터페이스이다[3]. Libclang 의 주요 컨테이너에는 Index, Translation Unit, Cursor 이 있다. C/C++ 코드의 AST 를 이용하기 위해 먼저 Index 를 생성하고, Index 에 대상 파일의 Translation Unit 을 로드한 후 그 TU 의 Cursor 을 생성하고 반복자로 삼아 AST 를 탐색한다.

Libclang 을 통해 AST 노드의 이름, 위치, 범위, 참조 정보, 타입 등을 알 수 있다.

2-2. ROS2

ROS2 는 메타 운영체제를 기반으로 로봇 응용 프로그램에 필수인 로봇과 센서를 하드웨어 추상화 개념으로 제어하는 지원 시스템이자 툴이다. ROS2 의 노드는 최소 단위의 실행 가능한 entity 를 가리킨다. ROS2 에는 single-threaded executor, multi-threaded executor, static single-threaded executor 의 세 가지 executor 이 존재한다. 개발자는 add_node() 매서드를 통해 노드를 executor 에 추가하고, spin() 매서드를 통해 executor 에 추가된 노드를 실행한다.

3. Concept

Multi-threaded executor 을 사용하기 위해서는 해당 노드들이 스레드 안전성을 가지는지 확인해야 한다. 스레드 안전성을 보장하기 위해서는 각 스레드의 원자성을 보장해야 하는데[4], 원자성이 보장되지 않으면 한 스레드가 작업 중인 데이터 공간을 다른 스레

드가 수정하여 본래 의도했던 결과가 나오지 않을 수도 있기 때문이다.

즉, 스레드 안전성을 보장하기 위해, 같은 multi-threaded executor 에 추가된 메서드들이 같은 데이터 공간에 접근하지 않도록 해야 한다.

4. Implementation

ROS2 코드에서 스레드 안전성이 보장받지 못하는 경우에는 두 가지가 있다.

(1) 서로 다른 노드들이 같은 데이터 공간에 접근하는 경우, 해당 데이터 공간은 노드 바깥에 선언되어 있고, 데이터 공간은 노드를 선언할 때 인자를 통해 전달되어야 할 것이다. 따라서 노드의 선언부의 인자를 확인하면 이 경우를 구분할 수 있다.

Pseudo code for (1)

```
executorList = [cursor of single-threaded executor]
for executor in executorList:
    nodeList.append({executor : [nodes added to executor]})
for node in nodeList:
    for arg of node:
        argList.append({arg : [nodes referencing arg]})
for arg in argList.keys():
    if len(argList[arg]) > 1:
        Warning!
```

```
<clang.cindex.TranslationUnit object at 0x0000029446710948>
Parsing file: singlethread_example.cpp
===============

shared_data_3  |  node_d
shared_data_1  |  node_c    node_b    node_a
shared_data_2  |  node_c    node_b
Warning with shared_data_1 => referenced in node_c node_b node_a
Warning with shared_data_2 => referenced in node_c node_b
```

(그림 1) ROS2 예제 코드에 대한 결과 화면

(2) 같은 노드 안의 서로 다른 콜백 함수들이 같은 데이터 공간에 접근하는 경우, 해당 데이터 공간은 노드 밖이나 안에 선언될 수 있지만 적어도 각각의 콜백 바깥에 선언되어 콜백을 호출할 때 인자로 전달해야 할 것이다. 따라서 콜백이 호출될 때 인자를 확인하면 이 경우를 구분할 수 있다.

Pseudo code for (2)

```
nodeList = [node in ROS2 code]
for node in nodeList:
    for func in node:
        for arg in [args referenced by func]:
            argList.append({arg : [funcs referencing arg]})
for arg in argList.keys():
    if len(argList[arg]) > 1:
        Warning!
```

```
Parsing file: member_function.cpp
===============

MinimalPublisher::count_           |   timer_callback
```

(그림 2) ros2_humble 의 예제 중 topics/minimal_publisher/member_function.cpp 에 대한 결과 화면

```
Parsing file: member_function_2.cpp
===============

MinimalPublisher::shared_number    |  timer_callback1    timer_callback2
MinimalPublisher::count_           |  timer_callback1    timer_callback2    timer_callback3
MinimalPublisher::shared_number2   |  timer_callback2
MinimalPublisher::shared_number3   |  timer_callback3

Warning with MinimalPublisher::shared_number => referenced in timer_callback1 timer_callback2
Warning with MinimalPublisher::count_ => referenced in timer_callback1 timer_callback2 timer_callback3
```

(그림 3) member_function.cpp 를 변형한 ROS2 코드에 대한 결과 화면

5. Conclusion

Libclang 은 파일 단위로 코드를 분석하기 때문에 다른 파일에 노드의 선언부와 구현부가 작성된 경우 분석이 어렵다. 같은 파일에 노드의 선언부와 구현부가 작성된 경우에도 노드의 네임스페이스가 다른 경우 cursor 의 get_children()이나 referenced() 매서드를 통해 검색할 수 없고, 토큰을 통해 수동으로 상호참조를 수행해야 한다. 또한 multi-threaded executor 을 이용할 경우 서로 다른 스레드에서 공유된 데이터 공간을 통해 값을 쉽고 빠르게 전달할 수 있다는 장점이 있는데, 이를 위해 의도적으로 데이터 공간을 공유하는 경우를 구분하지 못한다.

따라서 libclang 을 통해 서로 다른 네임스페이스나 파일의 내용까지 함께 분석하는 방법을 찾거나, 데이터 공간에 대한 단순 접근과 수정을 구분해 분석하는 방향으로 향후 연구를 수행할 것이다.

Acknowledgement

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학 ICT 연구센터육성지원사업의 연구결과로 수행되었음 (IITP-2023-2020-0-01797). 또한, 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 융합보안핵심인재양성사업의 연구 결과로 수행되었음 (IITP-2023-2022-0-01201).

참고문헌

[1] Open Robotics. 2023. Ros2 Documentation:Foxy. https://docs.ros.org/en/foxy/index.html

[2] Intel Corporation. 2023. Intel Hyper-Threading Technology Technical User's Guide. http://www.cslab.ece.ntua.gr/courses/advcomparch/2007/material/readings/Intel%20Hyper-Threading%20Technology.pdf

[3] Pontus Ljungren. Evaluation of Clang Tools for Information Extraction. 2023.

[4] Michael Pradel, Tomas R.Gross. Fully Automatic and Precise Detection of Thread Safety Violations. PLDI. Beiging China. 2012. 521-530.