The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.12, pp.669~675
eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.12.669

A Study on Secure Node Memory Allocation in ROS Composition

Jiwon Seo†

ABSTRACT

The Robot Operating System (ROS) has become a critical platform for developing advanced robotic systems across various fields, including smart homes, autonomous driving, and industrial automation. By modularizing complex robotic systems, ROS enables nodes to cooperate effectively while adopting a structure where each node operates as an independent process, ensuring fault isolation. However, the use of inter-process communication (IPC) introduces performance degradation and resource inefficiencies, particularly in scenarios involving frequent message exchanges or large-scale data processing. To address these challenges, ROS2 introduced the Composition feature, allowing multiple nodes to execute within a single process. While this approach improves communication efficiency by leveraging shared memory, it also raises potential security risks due to the shared address space. Specifically, loose memory access controls between nodes increase the likelihood of security vulnerabilities, which existing mechanisms such as Secure ROS (SROS) struggle to mitigate effectively. This paper proposes a novel method to enhance security and improve memory management in ROS Composition environments. By independently managing the memory segments of each node, the proposed mechanism prevents data interference and ensures secure memory access. Experimental results demonstrate that the method maintains stable performance with a low overhead of 3% to 10%, effectively enhancing the security of ROS Composition environments.

Keywords : ROS (Robot Operating System), Node Communication, Composition, SROS (Secure ROS)

ROS Composition에서의 안전한 노드 메모리 할당에 관한 연구

서 지 원†

요 약

로봇 운영체제(ROS, Robot Operating System)는 스마트 홈, 자율 주행, 산업 자동화 등 다양한 분야에서 첨단 로봇 시스템 개발의 핵심 플랫폼으로 활용되고 있다. ROS는 각 노드를 독립적인 프로세스로 실행하는 구조를 채택해 결함 격리(fault isolation)를 제공한다. 그러나 프로세스 간 통신(IPC)을 사용하는 방식은 빈번한 메시지 교환이나 대규모 데이터 처리 시 성능 저하와 리소스 비효율을 초래할 수 있다. 이 문제를 개선하기 위해 ROS2에서는 여러 노드를 하나의 프로세스에서 실행하는 Composition 기능을 도입하였다. Composition은 노드 간 공유 메모리를 활용해 통신 효율을 높이는 반면, 동일한 주소 공간을 공유함으로써 보안 취약점이 발생할 가능성이 있다. 특히, 노드 간 메모리 접근 규제가 느슨해져 보안 위협이 증가할 수 있으며, 기존 SROS(Secure ROS)와 같은 보안 메커니즘은 이를 효과적으로 해결하기 어렵다. 본 논문에서는 ROS Composition 환경에서의 보안을 강화하고 효율적인 메모리 관리 방안을 제안한다. 각 노드의 메모리 세그먼트를 독립적으로 관리하여 데이터 간섭을 방지하고, 안전한 메모리 접근을 보장하는 메커니즘을 설계하였다. 실험 결과, 제안된 방법은 3%에서 최대 10%의 낮은 오버헤드로 안정적인 성능을 유지하며, Composition 환경의 보안성을 효과적으로 향상시킴을 확인하였다.

키워드 : ROS (Robot Operating System), 노드 통신, Composition, SROS (Secure ROS)

1. 서 론

로봇 운영체제(ROS, Robot Operating System)는 스마트 홈, 자율 주행, 산업 자동화 등 다양한 분야에서 활용되는 첨단 로봇 시스템 개발의 핵심 기반이다. ROS는 오픈 소스 소프트웨어 프레임워크로, 복잡한 로봇 시스템을 모듈화하여 다양한 노드들이 협력하며 작동할 수 있게 한다[1-3]. ROS는 기본적으로 각 노드를 독립적인 프로세스로 실행하며, 노드 간 통신은 프로세스 간 통신(IPC)을 통해 이루어진다. 이러한 구조는 각 노드에 별도의 메모리 공간을 할당하여 결함 격리(fault isolation)를 가능하게 함으로써, 하나의 노드에서 발생한 오류가 다른 노드로 전파되지 않도록 한다. 그러나 독립적인 프로세스 구조는 IPC 및 소켓 통신을 사용하면서 통신 성능 저하와 리소스 비효율성을 초래할 수 있다. 특히, 빈번한 메시지 교환이나 대용량 데이터를 처리하는 경우 병목현상이 발생할

† 정 회 원 : 단국대학교 사이버보안학과 조교수
Manuscript Received : October 8, 2024
Accepted : December 5, 2024
* Corresponding Author : Jiwon Seo(jwseo@dankook.ac.kr)

가능성이 있다.

이러한 문제를 해결하기 위해, ROS1에서는 여러 노드를 하나의 프로세스에서 실행할 수 있는 nodelet 개념이 도입되었으며, ROS2에서는 이를 확장하여 Composition 기능을 통해 여러 노드를 하나의 실행 파일에서 실행할 수 있도록 하였다. 이 방식은 노드 간 공유 메모리를 활용하여 데이터를 빠르고 효율적으로 교환함으로써 성능 최적화를 가능하게 한다. 그러나 동일한 프로세스에서 여러 노드가 실행됨에 따라, 노드 간 동일한 주소 공간을 공유하게 되어 보안 취약점이 발생할 가능성이 높아진다. 이러한 상황에서는 성능과 효율성을 유지하면서도 노드 간 데이터 접근을 안전하게 제어할 수 있는 메모리 관리 및 보안 메커니즘이 필요하다.

따라서, ROS에서는 보안 기능을 제공하는 새로운 패키지로 SROS(Secure Robot Operating System)[4]가 도입되었다. SROS는 ROS 환경에서 보안을 강화하기 위한 확장 시스템으로, TLS(Transport Layer Security) 및 인증, 암호화 등을 적용하여 노드 간 통신을 보호한다. SROS는 각 노드가 독립된 인증서를 가지고 있으며, 노드 간 메시지가 송수신될 때 이를 암호화하여 보안성을 높인다. 또한, 각 노드가 수행하는 작업에 대한 권한을 설정하고 관리하여, 불필요한 데이터 접근을 방지하는 역할을 한다. 그러나 SROS의 보안 메커니즘을 ROS의 Composition에 바로 도입하는 것은 기술적인 어려움이 있다. Composition 에서는 여러 노드가 하나의 프로세스 내에서 실행되며, 이들은 동일한 메모리 공간을 공유하게 된다. SROS는 기본적으로 노드 간 통신을 보안하는 데 중점을 두고 있지만, Composition에서는 노드들이 프로세스 내부에서 같은 주소 공간을 공유하고 있기 때문에 노드 간 메모리 접근에 대한 통제가 어렵다. 즉, SROS는 서로 독립된 프로세스 간의 보안을 강화하는 데 효과적이지만, 같은 프로세스 내에서 실행되는 노드들에 대해서는 각 노드의 메모리 접근을 제한하는 데 한계가 있다.

본 논문에서는 ROS Composition에서의 보안성을 강화하고 효율적인 메모리 할당 방안을 제시한다. 이를 위해, 동일한 주소 공간에서 동작하는 각 노드의 메모리 세그먼트를 독립적으로 관리하는 방법을 제안한다. 제안된 메커니즘은 노드별로 메모리를 독립적으로 할당하고, 이를 다른 노드와 분리된 상태로 유지함으로써 노드 간 메모리 간섭을 방지하고 안전한 메모리 관리 환경을 제공한다. 실험 결과, 제안된 메커니즘을 Composition 환경에 적용했을 때 발생하는 오버헤드는 3%에서 최대 10%로 안정적인 성능을 유지하여 안전한 메모리 할당을 강화하는 데 기여한다.

2. Robot Operating System (ROS)

로봇 운영체제(ROS, Robot Operating System)는 모듈화된 구조를 통해 개발자들이 하드웨어와 소프트웨어의 다양한 구성 요소를 독립적으로 개발하고 배포할 수 있게 한다. 특히

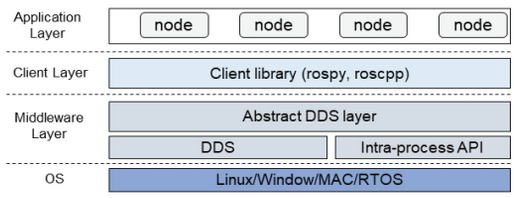

Fig. 1. ROS Architecture

ROS는 복잡한 로봇 시스템을 분산형 컴퓨팅 환경에서 실행할 수 있도록 설계되어, 여러 장치나 컴퓨터에 분산된 노드들이 협력하여 작업을 수행할 수 있다.

Fig. 1에서 볼 수 있듯이, ROS는 클라이언트 레이어(Client Layer), 미들웨어 레이어(Middleware Layer), 그리고 애플리케이션 레이어(Application Layer)의 세 가지 주요 레이어로 구성된다. 클라이언트 레이어는 개발자가 ROS와 상호작용하며 애플리케이션을 개발할 수 있는 다양한 도구와 라이브러리를 제공한다. 주요 기능으로는 노드 작성, 통신 설정, 데이터 처리 등이 있으며, Python과 C++를 지원하는 API인 rospy와 roscpp를 통해 메시지 송수신, 서비스 호출, 액션 실행 등을 구현할 수 있다.

미들웨어 레이어는 ROS의 통신 및 메시지 전달을 담당하며, 퍼블리셔-서브스크라이버(Pub/Sub) 모델과 서비스 모델을 통해 노드 간 데이터 교환을 관리한다. ROS2에서는 데이터 분배 서비스(DDS)를 기반으로 설계되어 분산 시스템에서 실시간 통신을 지원하며, 네트워크 상태에 따라 성능을 조정해 실시간성과 유연성을 제공한다.

애플리케이션 레이어는 센서 데이터 처리, 로봇의 동작 제어, 네비게이션, 로봇 팔 조작 등 실제 로봇 응용 프로그램이 실행되는 부분이다. 이 레이어에서 각 노드는 독립적으로 작동하며 상호 통신을 통해 로봇 시스템의 유연성과 확장성을 보장한다.

2.1 ROS의 In-Process 통신: Nodelet과 Composition

ROS의 초기 구조에서는 각 노드가 독립된 프로세스로 실행되며, 노드 간 통신은 주로 IPC(Inter-Process Communication)나 소켓을 통해 이루어졌다. 이러한 구조는 결함 격리(fault isolation)를 가능하게 하지만, 빈번한 데이터 교환이 필요한 응용 프로그램에서는 통신 오버헤드로 인해 성능 저하와 자원 낭비 문제가 발생했다. 이를 해결하기 위해 ROS1에서는 Nodelet 개념이 도입되었으며, ROS2에서는 이를 확장한 Composition이 등장하였다. 두 개념 모두 노드 간 통신 효율성을 개선하는 데 중점을 두었지만, 구조와 활용 목적에서 차이가 있다.

1) Nodelet

Nodelet은 ROS1에서 도입된 개념으로, 여러 노드를 하나의 프로세스 내에서 실행할 수 있도록 설계되었다. 기존의 독립 프로세스 실행 방식에서는 노드 간 통신에 IPC 또는 소켓

통신이 사용되어야 했지만, Nodelet은 동일한 프로세스 내에서 노드가 메모리를 공유함으로써 통신 효율을 크게 향상시킨다. 이를 통해 데이터 전송 시 발생하는 복제와 변환 과정을 줄이고, 통신 오버헤드를 최소화할 수 있다. 이러한 특징은 대량의 데이터 교환이 필요한 이미지 처리나 센서 데이터 처리 응용 프로그램에서 특히 유용하다. Nodelet의 주요 장점은 intra-process 통신을 통해 노드 간 통신 속도를 대폭 개선할 수 있다는 점이다.

2) Composition

Composition은 ROS2에서 Nodelet의 한계를 보완하고 확장한 개념이다. ROS2의 Composition은 ROS Client Library (RCL)에 포함되어 있으며, 여러 노드를 하나의 실행 파일에서 실행하도록 지원한다. Nodelet과 달리, Composition은 ROS2의 DDS(Data Distribution Service) 기반 아키텍처를 활용하여 노드 간 통신을 더욱 효율적으로 관리한다. 이 방식은 자원이 제한된 환경이나 실시간 응답성이 중요한 시스템에서 성능 최적화에 유리하며, ROS2의 분산 시스템 특성을 유지하면서도 높은 통신 성능을 제공한다.

2.2 SROS(Secure Robot Operating System)

SROS(Secure Robot Operating System)는 ROS의 보안 문제를 해결하기 위해 도입된 확장 시스템으로, ROS2의 DDS (Data Distribution Service) 기반 통신 구조와 밀접하게 연관되어 있다. ROS2는 DDS를 통해 분산 시스템에서 실시간 통신을 지원하며, SROS는 이 DDS Security 확장을 활용하여 통신의 기밀성, 무결성, 그리고 인증을 제공한다. DDS는 다양한 QoS(Quality of Service) 설정으로 데이터 전송을 최적화하는 동시에, DDS Security 확장을 통해 데이터 암호화와 접근 제어 기능을 지원한다. 이를 기반으로 SROS는 노드 간 메시지를 암호화하고, 각 노드에 고유 인증서를 발급하여 네트워크 상에서의 보안을 강화한다. 이러한 방식은 중간자 공격과 같은 보안 위협을 차단하며, 통신의 안전성을 보장한다.

특히, SROS는 DDS에서 제공하는 TLS(Transport Layer Security) 프로토콜을 활용하여 노드 간 데이터를 안전하게 암호화한다. 각 노드는 고유한 인증서를 가지고 있으며, 인증서를 통해 상호 신뢰를 보장받은 노드 간에만 통신이 이루어진다. 이를 통해 무단 노드가 시스템에 접근하거나 데이터를 탈취하는 것을 방지할 수 있다. 또한, SROS는 세분화된 액세스 제어를 통해 각 노드가 접근할 수 있는 데이터와 리소스를 관리한다. 이를 통해 불필요한 데이터 접근을 방지하고, 특정 노드가 공격을 받거나 손상되더라도 그 영향이 전체 시스템으로 확산되지 않도록 한다. 이처럼 SROS는 통신 보안과 액세스 제어를 통해 ROS2 시스템의 전반적인 보안성을 크게 향상시킨다.

그러나 SROS는 주로 프로세스 간 통신을 보호하는 데 초점을 맞추고 있으며, 같은 프로세스 내에서 실행되는 노드 간의 메모리 공유 문제를 해결하기에는 한계가 있다. ROS의 Composition 구조에서는 여러 노드가 동일한 프로세스 내에서 실행되며, 동일한 주소 공간을 공유하기 때문에 노드 간 데이터 접근에 대한 보안 통제가 상대적으로 어렵다. 이로 인해 하나의 노드에서 발생한 취약점이 다른 노드로 확산될 위험이 존재한다. 따라서, Composition 환경에서의 노드 메모리 접근에 대한 세밀한 권한 제어가 필수적이다.

2.3 ROS 보안을 위한 관련 연구 조사

최근 ROS 보안 연구는 주로 통신 보안과 암호화 기술에 초점을 맞추어왔다. 대표적인 예로 SROS[4]와 SROS2[5]는 TLS 기반 암호화를 통해 데이터의 기밀성과 무결성을 보장하며, 데이터 스니핑 및 중간자 공격(man-in-the-middle) 방어를 목표로 한다. 그러나 이러한 솔루션들은 공격자가 노드를 제어하거나 암호화되지 않은 메모리 상의 노드 데이터에 접근하는 것을 방지하지 못한다. [6]는 Intel SGX를 활용하여 암호화되지 않은 데이터(예: 메시지, 노드 ID)를 안전한 환경(Secure World)에 저장하고, 보안 컨텍스트로 전환하여 마스터 노드의 역할을 수행하도록 설계되었다. 그러나 ROS1을 기반으로 설계되었기 때문에 ROS2에 적용하기 위해서는 전체 설계 요구사항을 수정해야 하는 한계가 있다. 다른 연구[7]에서는 ROS2 프로젝트에서 SROS2 권한 파일을 효율적이고 자동으로 생성하는 방법을 제안했으며, 또 다른 연구[8]는 ROS2의 DDS 보안을 형식적으로 검증했다. 그러나 이들 연구는 SROS2의 기존 기능을 활용하는 데 중점을 두었으며, SROS2의 구조적 취약점을 악용해 공격자가 이를 우회할 가능성을 충분히 고려하지 못했다. 이에 반해, [9]는 ROS2 구현 자체의 보안에 대한 첫 번째 연구를 제시하며, SROS2로는 대응할 수 없는 새로운 취약점이 존재함을 강조하였다. 그러나 이 연구 역시 ROS Composition 환경에서의 메모리 보호 문제는 다루지 않았다.

본 논문은 기존 연구들이 주로 통신 보안과 암호화 기술에 중점을 둔 것과 달리, ROS2의 Composition 환경에서 보안성을 강화하는 새로운 방향성을 제시한다. 이를 통해 본 논문은 암호화되지 않은 메모리 데이터까지 안전하게 보호하며, ROS2에서 완전히 호환되는 실용적인 보안 솔루션을 제공한다.

3. ROS Composition에서의 안전한 메모리 할당

ROS의 Composition에서는 여러 노드가 하나의 프로세스 내에서 실행되며, 각 노드가 동일한 메모리 공간을 공유한다. 이러한 구조는 성능 최적화라는 장점을 제공하지만, 노드가 독립적으로 메모리를 관리하지 못하면서 동적 메모리가 무질서하게 배열되는 문제가 발생할 수 있다. 이는 노드 간 메모리 침범 및 보안 취약점을 유발하며, 메모리 관리의 어려움을 초래한다. 이를 해결하기 위해 본 논문에서는 노드별로 독립된 메모리 세그먼트를 관리하는 방안을 제안한다. 제안된 방법은

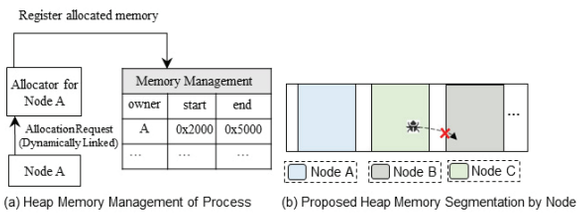

Fig. 2. Secure heap memory allocation

Fig. 2에서와 같이 노드별로 메모리를 할당하고, 이를 분리된 상태로 유지하는 메커니즘을 사용한다. 각 노드는 필요한 메모리를 자신의 세그먼트 내에서 할당받아, 노드 간 메모리 간섭을 방지하고 안전한 메모리 관리를 실현한다.

3.1 노드 메모리 세그먼트 기법

일반적인 메모리 할당자는 큰 메모리 영역을 미리 확보한 뒤, 요청된 크기에 맞는 메모리 청크를 나누어 제공하여 효율적인 메모리 관리를 가능하게 한다. 그러나 한 프로세스 내에서 여러 노드가 실행될 경우, 메모리가 스택, 힙, 글로벌 영역 등에 혼재되어 저장되며, 노드별로 독립적인 할당이 이루어지지 않는다. 이로 인해 노드 간 메모리 접근이 규제 없이 발생할 가능성이 높아져 보안 취약점이 생길 수 있다.

이를 해결하기 위해, 본 논문에서는 각 노드가 독립적인 메모리 할당 영역을 사용할 수 있는 방식을 제안한다. Composition에서는 실행할 노드 수를 정적 또는 동적으로 설정할 수 있는데, 본 논문에서는 정적 설정을 가정하여 하나의 주소 공간을 노드 수에 따라 여러 세그먼트로 분할하고, 각 세그먼트를 독립적인 메모리 영역으로 관리하도록 설계하였다. 이 방식은 각 세그먼트에 해당 노드의 데이터만 저장되도록 하여, 다른 노드의 메모리 객체와의 혼재를 방지함으로써 보안성을 확보하고 노드 간 데이터 간섭을 차단한다.

노드가 할당된 세그먼트 내의 메모리를 모두 사용하는 상황을 대비하여, 본 논문에서는 추가 메모리 요청을 처리하기 위한 기법을 도입하였다. 각 세그먼트의 크기를 사전에 충분히 설정하여 메모리 초과 상황을 최소화하였으며, 세그먼트를 초과하는 요청이 발생할 경우, 프로세스 내에 예약된 확장 가능한 메모리 풀(memory pool)을 통해 추가 메모리를 제공하도록 설계하였다. 이 메모리 풀은 초과 요청을 처리하기 위해 특정 노드에만 접근 권한이 부여된 별도의 영역으로 관리된다. 또한, 향후 연구에서는 메모리 사용량을 실시간으로 모니터링하고, 세그먼트 내 미사용 공간을 재활용하기 위한 메모리 압축 기법을 적용하여 메모리 자원의 효율성을 더욱 향상시킬 예정이다.

3.2 노드 메모리 보호 기법

ROS Composition 환경에서 여러 노드가 하나의 프로세스 내에서 실행될 때, 각 노드의 메모리 세그먼트는 독립적으로 관리되어야 하며, 다른 노드의 메모리 영역에 접근하지 못하도록 보호하는 것이 중요하다. 이를 위해 본 논문에서는 mprotect() 시스템 호출을 활용한 메모리 보호 기법을 제안하였다.

mprotect()는 리눅스 및 유닉스 기반 시스템에서 특정 메모리 영역의 접근 권한(읽기, 쓰기, 실행)을 동적으로 설정할 수 있는 시스템 호출이다. 본 논문에서는 이를 활용하여, 노드가 활성 상태일 때만 해당 노드의 메모리 세그먼트에 접근 권한을 부여하고, 비활성 상태에서는 접근 권한을 제한하는 방식을 구현하였다. 이러한 메모리 보호는 각 노드의 독립성을 보장하고, 하나의 노드에서 발생한 오류나 침해가 다른 노드에 영향을 미치는 것을 방지한다.

메모리 보호 메커니즘은 다음과 같다. 각 노드는 4.1 노드 메모리 세그먼트 기법으로 고유의 메모리 세그먼트를 가지고 있으며, 메모리 세그먼트는 프로세스의 주소 공간 내에서 각 노드별로 분리된다. 이때 mprotect()를 사용하여, 노드가 활성화되어 있을 때는 해당 노드의 메모리 세그먼트에 읽기, 쓰기, 실행 권한을 부여하고, 비활성화 상태에서는 권한을 제거하여 메모리 접근을 차단한다. 구체적으로, 다음과 같은 단계로 메모리 보호가 이루어진다.

노드 실행 전: 각 노드의 메모리 세그먼트는 기본적으로 읽기 전용(read-only)으로 설정된다. 이를 통해 다른 노드나 외부 프로세스가 해당 메모리 영역에 임의로 접근하는 것을 차단한다.

노드 활성화 시: 노드가 활성화되면, mprotect()를 사용하여 해당 노드의 메모리 세그먼트에 읽기(read), 쓰기(write), 실행(execute) 권한을 부여한다. 이를 통해 노드가 자신의 메모리 세그먼트에서 자유롭게 작업을 수행할 수 있다.

노드 비활성화 시: 노드가 작업을 완료하거나 비활성화되면, mprotect()를 다시 호출하여 해당 노드의 메모리 세그먼트를 읽기 전용 또는 완전히 차단된 상태로 변경한다. 이를 통해 다른 노드가 해당 메모리 영역에 침범하거나 데이터를 손상시키는 것을 방지한다.

이 방식은 각 메모리 페이지에 대해 동적으로 접근 권한을 설정하여 메모리 격리를 유지하며, 성능에 미치는 영향을 최소화한다. 또한, 실행 중인 노드만 자신의 메모리 세그먼트에 접근할 수 있도록 제한함으로써, 악성 코드나 취약점이 있는 노드가 다른 노드의 메모리 공간에 접근할 가능성을 줄인다. 이를 통해 노드 간 메모리 침범 위험을 최소화하고, 시스템의 안정성과 보안성을 크게 향상시킬 수 있다.

3.3 노드 실행 흐름 추적 기법

노드 메모리 보호 메커니즘에서 제안된 mprotect()를 통한 메모리 접근 권한 제어는, 특정 노드가 활성화되었을 때만 해당 노드의 메모리 세그먼트에 접근할 수 있도록 허용하고, 비활성화된 노드에 대해서는 접근을 차단하는 방식으로 이루어진다. 이러한 제어를 정확히 수행하기 위해, 각 노드의 활성화 및 비활성화 시점을 실시간으로 파악하는 것이 필수적이다.

이를 위해 본 논문에서는 현재 실행 중인 노드 정보를 추적하여, mprotect()를 통해 메모리 접근 권한을 동적으로 설정할 수 있도록 한다. 실행 중인 노드에만 메모리 세그먼트 접근 권한을 부여하며, 다른 노드의 메모리에 대한 접근을 차단함으로써 보안을 강화한다.

구체적으로, 각 스레드에서 실행 중인 노드의 상태를 지속적으로 모니터링하며, 노드 간 컨텍스트 전환 시점에 관련 정보를 갱신하고 저장한다. 예를 들어, 특정 노드로 컨텍스트가 전환되면 해당 노드의 메모리 세그먼트에 대해 읽기/쓰기 권한을 부여하기 위해 mprotect()가 호출된다. 반대로, 노드가 비활성화되거나 다른 노드로 전환되면, 이전 노드의 실행을 중단하고 해당 메모리 접근 권한을 제거하며, 새롭게 활성화된 노드에 대해서만 접근 권한을 부여한다. 이 실행 흐름 추적 기법은 노드 간 컨텍스트 전환에 따라 실시간으로 메모리 접근 권한을 동적으로 관리하고 노드 간 비정상적인 메모리 접근을 방지한다.

4. 실 험

본 장에서는 제안한 안전한 메모리 할당 메커니즘의 효율성과 실효성을 평가하기 위한 실험 결과를 제시한다. 실험은 제안된 메커니즘이 기존 메모리 할당 방식과 비교하여 성능 및 메모리 사용량 측면에서 얼마나 향상되었는지를 분석하는 것을 목표로 한다. 실험은 여러 노드 구성 및 메모리 할당 시나리오에서 진행되었으며, 각 실험 결과를 통해 제안된 메커니즘이 기존 메모리 할당 방식과 비교하여 얼마나 성능 최적화가 이루어졌는지를 확인한다.

4.1 실험 환경 세팅

본 연구는 제안된 메커니즘의 기능적 정확성과 성능을 평가하기 위해 다음과 같은 환경에서 실험을 수행하였다. 하드웨어는 Intel i9-10900K CPU, 128GB 메모리로 구성되었으며, 운영체제는 Ubuntu 18.04.6 LTS를 사용하였다. 실험에 사용된 ROS2는 Dashing Diademata 버전을 기반으로 클라이언트 라이브러리를 수정하였으며, 메모리 할당자는 glibc 2.27 버전을 수정하여 적용하였다. ROS2의 Dashing 버전은 Composition 기능이 충분히 제공되지 않는 초기 단계였기 때문에, 실험을 위해 ROS2 패키지에서 제공하는 예제[10]와 데모[11]를 포팅하여 Composition 환경을 구축하였다. 이를 통해 제안된 메커니즘을 테스트할 수 있는 실험 환경을 구성하였다.

성능 평가는 두 가지 기준점을 기반으로 제안된 안전한 ROS Composition 메커니즘인 SEC-COMP와 비교하여 진행되었다. 첫 번째 기준점인 PROCESS는 프로세스 격리를 통해 하나의 프로세스에서 단일 노드만 실행하는 안전한 구조를 의미하며, 두 번째 기준점인 IN-PROCESS는 하나의 프로세스 내에서 다수의 노드를 실행하지만, 보안이 적용되지 않은 일반적인 ROS Composition 환경을 의미한다.

본 실험의 목표는 노드 메모리 보호 메커니즘이 적용된 SEC-COMP가 기존 PROCESS 및 IN-PROCESS와 비교하여 성능과 보안 측면에서 제공하는 개선된 이점을 입증하는 것이다. 이를 위해 psutil 라이브러리를 사용하여, CPU 및 메모리 사용률을 측정하고 분석하였다. 실행 중인 프로세스의 초당 CPU 및 메모리 사용률을 기록한 후, 일정 시간 동안의 데이터를 바탕으로 기하 평균(geometric mean)을 계산하여 성능을 평가하였다.

4.2 성능 비교: PROCESS, IN-PROCESS, SEC-COMP

1) CPU 성능 비교

CPU 성능 비교 결과는 Fig. 3에 요약되어 있으며, 각 메시지 전송 주기에 따른 평균 응답 시간을 기반으로 평가되었다. 실험은 하나의 프로세스 내에서 메시지를 송신하는 노드 1개와 이를 수신하는 노드 2개로 구성된 Composition 환경에서 진행되었다. 송신 노드는 10ms에서 90ms까지 다양한 통신 주기로 16바이트 크기의 메시지를 전송하며, 통신 주기에 따라 CPU 성능 변화를 관찰하였다.

IN-PROCESS는 여러 노드가 하나의 프로세스 내에서 실행되며, 프로세스 내부에서 메시지를 직접 주고받는 방식으로 동작하여, 프로세스 간 통신을 사용하는 PROCESS보다 더 우수한 CPU 성능을 보였다. SEC-COMP는 노드 메모리 보호 메커니즘이 추가된 구성임에도 불구하고, PROCESS와 비교해 더 나은 성능을 나타냈다. 특히, SEC-COMP는 mprotect()를 활용한 메모리 접근 제어를 통해 안전한 Composition 환경을 제공하면서도, IN-PROCESS 대비 3%에서 최대 10% 수준의 낮은 오버헤드만을 발생시키며 안정적인 성능을 유지하였다.

Fig. 3을 보면 통신 주기가 길어질수록 CPU 사용률이 감소하는 경향을 확인할 수 있다. 이는 통신 주기가 길어지면 메시지 발행(publish) 및 수신(subscribe) 작업이 덜 빈번하게 이루어져, 노드 간 데이터 교환에 필요한 연산량이 줄어들기 때문이다. 특히, 10ms 주기에서는 30ms 또는 50ms 주기와 비교했을 때 SEC-COMP의 오버헤드가 상대적으로 더 높아지는 것을 관찰할 수 있었다. 이는 노드의 콜백 함수가 매우 빈번하게 실행되면서, 메모리 보호 작업(mprotect 호출)이 자주 발생하기 때문이다. 그러나 전반적으로 SEC-COMP는 IN-

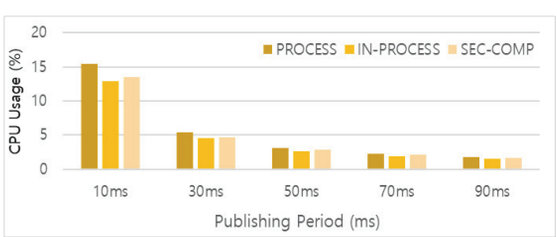

Fig. 3. CPU usage comparison across PROCESS, In-PROCESS, and SEC-COMP environments

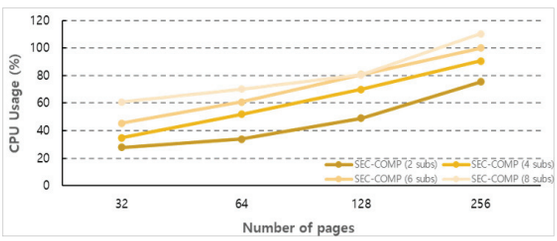

Fig. 4. Impact of node count and message size on context switching and mprotect overhead at 1ms intervals

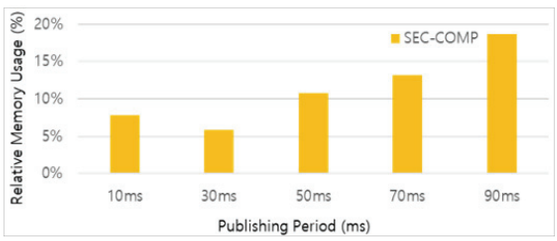

Fig. 5. Memory usage compared to IN-PROCESS

PROCESS와 유사한 CPU 성능을 유지하면서도, 메모리 보호 기능을 추가적으로 제공하여 효율성을 효과적으로 입증하였다.

Fig. 4는 메시지를 송신하는 노드 1개와 이를 수신하는 노드의 수를 2개에서 8개까지 증가시키는 환경에서, context switching이 빈번하게 발생하는 상황을 실험한 결과를 나타낸다. 실험은 32바이트에서 256바이트 크기의 메시지를 1ms 통신 주기로 전송하며, 노드 수 증가가 context switching과 시스템 성능에 미치는 영향을 측정하였다. 실험 결과, 노드 수와 메시지 크기가 증가함에 따라 context switching 빈도가 높아지고, 이에 따른 시스템 오버헤드도 점진적으로 증가하는 경향을 보였다. 특히, 메시지 크기가 커질수록 각 노드가 처리해야 할 데이터량이 많아지며, 메모리 접근과 보호 작업에 필요한 mprotect 호출이 증가하여 오버헤드가 크게 상승하였다. 이러한 결과는 노드 수와 메시지 처리량이 많은 환경에서 메모리 관리와 context switching을 최적화할 필요성을 강조한다. 향후 연구에서는 이와 같은 한계를 개선하기 위한 효율적인 메모리 관리 및 context switching 최적화 방안을 탐구하고자 한다.

2) 메모리 성능 비교

앞선 CPU 성능 비교와 동일하게 실험 환경은 하나의 프로세스 내에서 메시지를 송신하는 노드 1개와 이를 수신하는 노드 2개로 구성하였다. 메모리 측정을 위해 송신 노드의 통신 주기를 변경하며, 16바이트 크기의 메시지를 다양한 시간 간격으로 전송하였다.

PROCESS는 프로세스 간 격리를 통해 각 노드가 독립적인 메모리 공간을 사용하지만, 메모리 사용량 측면에서 가장 높은 값을 나타내었다. 예를 들어, 메시지 전송 주기가 10ms일 때 PROCESS의 메모리 사용률은 IN-PROCESS 및 SEC-COMP에 비해 약 2.47 배 정도 더 많은 메모리를 사용한다. IN-PROCESS는 프로세스 내에서 여러 노드가 메모리 자원을 공유함으로써 PROCESS보다 훨씬 효율적인 메모리 사용을 보여준다.

Fig. 5는 IN-PROCESS와 비교했을 때 SEC-COMP의 상대적인 메모리 오버헤드를 보여준다. SEC-COMP는 안전한 메모리 할당 기능을 제공하면서도, 메모리 사용량 측면에서 IN-PROCESS와 유사한 성능을 보였다. 그러나 CPU 성능과 달리, 메모리 오버헤드는 통신 주기가 길어짐에 따라 증가하는 양상을 보였다. 이는 통신 주기가 길어질수록 발행된 메시지가 소비되기까지 더 오래 대기열에 머물며, 메모리에 저장되는 메시지의 양이 증가하기 때문이다. 전반적으로, SEC-COMP는 PROCESS에 비해 훨씬 적은 메모리를 소비하면서도 IN-PROCESS와 유사한 메모리 효율성을 유지하였다. 이를 통해, SEC-COMP가 추가적인 메모리 할당 기능을 제공하면서도 효율적인 메모리 성능을 유지할 수 있음을 입증하였다.

5. 향후 발전 방향

5.1 동적 메모리 보호 메커니즘

본 논문에서는 ROS Composition 환경에서 안전한 메모리 할당을 위해 컴파일 타임에 고정된 노드 수를 기반으로 한 정적 메모리 할당 및 보호 메커니즘을 설계하고 구현하였다. 그러나, 정적 메모리 할당 방식은 동적인 요구 사항이 많은 실제 사이버-물리 시스템(CPS) 환경에서 유연성이 부족하다는 한계가 있다. CPS의 동적인 특성을 고려하면, 실행 중에 노드가 추가되거나 제거되는 상황이 빈번히 발생하며, 시스템은 이러한 변화를 실시간으로 처리할 수 있어야 한다. 현재의 정적 메모리 할당 방식은 컴파일 타임에 노드 수가 고정된 환경에서만 작동하므로, 실행 중 새로운 노드가 추가되거나 기존 노드가 제거되는 상황에 유연하게 대응하지 못한다. 따라서, 향후 연구에서는 실행 중 노드의 추가와 제거를 지원하는 동적 메모리 할당 메커니즘의 도입이 필요하다. 이를 위해, 노드가 추가될 때 메모리 세그먼트를 동적으로 생성하고, 이를 안전하게 관리하는 메모리 보호 기법을 개발하여, ROS Composition 환경에서 요구되는 실시간성과 유연성을 향상시키고자 한다.

5.2 하드웨어 기반 메모리 보호

소프트웨어 기반 메모리 보호는 유연성을 제공하지만, 시스템 호출(mprotect) 과정에서 오버헤드가 발생해 실시간성을 요구하는 시스템에서 성능 저하를 초래할 수 있다. 이를 보완하기 위해 ARM MPU(Memory Protection Unit)와 MTE (Memory Tagging Extension) 같은 하드웨어 기반 보호 기법이 주목받고 있다. MPU는 메모리 영역에 대한 접근 권한을 하드웨어적으로 제어하며, 읽기, 쓰기, 실행 권한을 세밀하게

설정할 수 있다. 이를 통해 노드 활성화 시 해당 메모리 세그먼트에 접근 권한을 부여하고, 비활성화 시 자동으로 차단할 수 있다. MTE는 메모리 블록에 고유 태그를 할당하여, 잘못된 태그를 감지하면 메모리 침범을 하드웨어적으로 차단한다. 이러한 하드웨어 기법은 소프트웨어적 오류나 악의적인 공격으로부터 시스템을 보호하면서도 성능 저하를 최소화한다. 향후 연구에서는 ARM MPU와 MTE를 ROS Composition 환경에 적용하여, 동적 노드 추가 및 제거가 빈번한 시스템에서도 실시간성과 보안성을 동시에 만족하는 효율적인 메모리 보호 기법을 모색할 예정이다. 이러한 연구는 사이버-물리 시스템에서 안전성과 성능을 강화하는 데 기여할 것이다.

6. 결 론

로봇 운영체제(ROS)는 다양한 분야에서 첨단 로봇 시스템 개발의 핵심 플랫폼으로 자리 잡았지만, 기존 프로세스 간 통신(IPC) 방식은 성능 저하와 리소스 비효율의 한계를 드러냈다. 이를 개선하기 위해 도입된 ROS2의 Composition 기능은 통신 효율을 높였으나, 동일한 주소 공간 공유로 인한 보안 취약점이 문제로 남아 있다. 본 논문에서는 ROS Composition 환경에서 보안을 강화하고 효율적인 메모리 관리 방안을 제안하였다. 노드별 독립적 메모리 세그먼트를 통해 데이터 간섭을 방지하며, 낮은 오버헤드로 안정적인 성능과 보안성을 동시에 확보하였다. 향후 연구를 통해 더욱 복잡한 로봇 시스템에서도 제안된 메커니즘의 적용 가능성을 확대할 예정이다.

References

[1] M. Quigley et al., “Ros: an open-source robot operating system,” in ICRA Workshop on Open Source Software, Vol.3, No.3.2, p.5, Kobe, Japan, 2009.

[2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, Vol.7, No.66, p.eabm6074, 2022.

[3] J. McClean, C. Stull, C. Farrar, and D. Mascarenas, “A preliminary cyber-physical security assessment of the robot operating system (ros),” in *Unmanned Systems Technology XV*, Vol.8741, SPIE, pp.341-348, 2013.

[4] R. White, D. H. I. Christensen, and D. M. Quigley, “Sros: Securing ros over the wire, in the graph, and through the kernel,” *arXiv preprint arXiv:1611.07060*, 2016.

[5] O. Robotics, “Ros 2 dds-security integration,” https://design.ros2.org/articles/ros2_dds_security.html, 2022.

[6] G. Mazzeo and M. Staffa, “Tros: Protecting humanoids ros from privileged attackers,” *International Journal of Social Robotics*, Vol.12, No.3, pp.827-841, 2020.

[7] R. White, H. I. Christensen, G. Caiazza, and A. Cortesi, “Procedurally provisioned access control for robotic systems,” In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp.1-9, 2018.

[8] Y. Liu, Y. Guan, X. Li, R. Wang, and J. Zhang, “Formal analysis and verification of DDS in ROS2,” In *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pp.1-5, 2018.

[9] G. Deng, G. Xu, Y. Zhou, T. Zhang, and Y. Liu, “On the (in) security of secure ROS2,” In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp.739-753, 2022.

[10] ROS2 Examples [Internet], https://github.com/ros2/examples, 2022.

[11] ROS2 Demos [Internet], https://github.com/ros2/demos, 2022.

서 지 원

https://orcid.org/0000-0003-1848-750X
e-mail : jwseo@dankook.ac.kr
2023년 3월~2024년 2월
한국자동차연구원 선임연구원
2024년~현 재 단국대학교
사이버보안학과 조교수
관심분야 : System Security & Cyber Physical System Security