https://doi.org/10.33778/kcsa.2022.22.1.099

파티션 기반 보안 실시간 운영체제의 구현 및 성능 분석★

서 경 덕*, 이 우 진**, 채 병 민**, 김 훈 규***, 이 상 훈***

요 약

네트워크 중심전(NCW)으로 전장환경이 변하면서 무기체계는 IT 기술을 융합한 신개념의 무기체계로 진화하고 있고, 이러한 핵심 기능은 대부분 임베디드 소프트웨어로 구현됨에 따라 임베디드 소프트웨어는 무기체계 성능을 좌우하는 핵심 요소로 중요성이 증가하고 있다. IoT 기술이 발전하고 임베디드 소프트웨어의 활용 범위가 확대됨에 따라 점차 고도화되고 다양해지는 사이버 위협은 임베디드 소프트웨어를 운용하는 무기체계로 확대되고 있고, 무기체계는 단일체계로부터 네트워크에 의한 연동까지 다양한 형태로 운영되기 때문에 어플리케이션 수준에서의 보안보다는 시스템 수준인 운영체제 커널 수준에서의 강력한 사이버 보안이 필요한 실정이다. 본 논문에서는 무기체계 임베디드 소프트웨어를 사이버 공격으로부터 보호하기 위하여 운영체제 수준에서 임베디드 소프트웨어를 보호하는 무기체계용 보안 실시간 운영체제의 설계 및 구현과 그 성능 측정 결과에 대하여 설명하였다.

Implementation and Performance Analysis of Partition-based Secure Real-Time Operating System★

Kyungdeok Seo*, Woojin Lee**, Byeongmin Chae**, Hoonkyu Kim***, Sanghoon Lee***

ABSTRACT

With current battlefield environment relying heavily on Network Centric Warfare(NCW), existing weaponary systems are evolving into a new concept that converges IT technology. Majority of the weaponary systems are implemented with numerous embedded softwares which makes such softwares a key factor influencing the performance of such systems. Furthermore, due to the advancements in both IoT technoogies and embedded softwares cyber threats are targeting various embedded systems as their scope of application expands in the real world. Weaponary systems have been developed in various forms from single systems to interlocking networks. hence, system level cyber security is more favorable compared to application level cyber security. In this paper, a secure real-time operating system has been designed, implemented and measured to protect embedded softwares used in weaponary systems from unknown cyber threats at the operating system level.

Key words : Separation Kernel, Partitioning, ARINC 653, SKPP, Safety Critical System, Real-Time OS, RTOS

접수일(2022년 2월 25일), 수정일(2022년 3월 23일), 게재확정일(2022년 3월 29일)

★ 본 논문은 국방과학연구소에서 수행한 "무기체계 고신뢰 내장형 실시간 보안 OS 기술" 과제의 지원으로 작성되었음.

* 한화시스템 미래정보통신연구소(주저자)
** 한화시스템 미래정보통신연구소(공동저자)
*** 국방과학연구소 사이버/네트워크기술센터(공동저자)

1. 서 론

세계 최고 수준의 항공우주기술을 집대성했다는 평가를 받는 미국 록히드마틴사의 F-35 전투기 기능 중 90% 이상이 SW로 구동되는 등 현대 무기체계는 점차 다기능, 고성능이 요구됨에 따라 무기체계 내의 SW 비중이 높아지고 있고, 이들은 대부분 임베디드 SW(embedded SW)로 구현되고 있다. 군용 차량, 함정, 항공기와 같은 무기체계 내에 탑재되는 임베디드 SW들은 시스템 자체의 높은 신뢰성 및 실시간성을 보장하기 위하여 실시간 운영체제(RTOS, Real-Time Operating System) 기반으로 개발되어 운용되고 있다.

본래 RTOS는 단순하고 반복적인 작업을 수행하는 단일 응용 기반의 실시간 제어시스템을 위하여 설계된 소프트웨어 플랫폼이지만 최근 무기체계에 탑재되는 마이크로 프로세서 성능의 비약적인 발전으로 여러 응용을 구동하기에 충분한 성능을 제공하면서, 다수의 응용 구동 시에도 실시간성과 안전성이 훼손되지 않는 RTOS 개발에 대한 요구 수요가 증가하고 있다.

한편, 사물인터넷(IoT)과 가상물리시스템(CPS) 등의 기술이 발전하고 임베디드 SW의 활용 범위가 확대됨에 따라 점차 고도화되고 다양해지는 사이버 위협은 임베디드 SW를 운용하는 무기체계로 확대되고 있다. 무기체계 임베디드 SW에 대한 보안 약점 및 취약점을 공격하는 사이버 위협은 무기체계 자체 오동작으로 인한 대형 물리적 사고를 유발할 수 있을 뿐만 아니라 더 나아가서는 전쟁의 승패를 좌우할 수 있을 정도로 심각한 위협이 될 수도 있다. 무기체계는 단일체계로부터 네트워크에 의한 연동까지 다양한 형태로 운영되어 어플리케이션 수준에서의 보안 기술로는 사이버 방호에 한계가 있으므로, 시스템 수준인 운영체제 커널 수준에서의 강력한 보안 기능 제공이 필요한 실정이다.

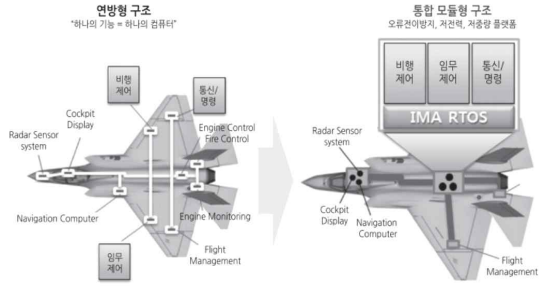
본 논문에서는 다수의 응용 간 분리 및 보호를 위한 파티셔닝 개념을 도입하여 실시간성과 함께 안전성 및 보안성을 제공하는 무기체계용 보안 RTOS를 기존 타 OS 기반이 아닌 신규 설계 및 구현한 결과에 대하여 설명하고, 구현한 보안 RTOS와 상용 RTOS의 성능을 비교 측정한 결과를 분석한다.

2. 관련 연구

2.1 ARINC 653

2.1.1 ARINC 653 개요

기존의 항공전자 시스템은 신뢰성과 인증을 이유로 하나의 컴퓨터에서 하나의 기능만을 수행하였고, RTOS 없이 펌웨어 레벨에서 제어하는 것이 일반적이었으나, 항공기의 기능이 많아짐에 따라 컴퓨터의 수가 증가하면서 SWaP(Size, Weight and Power) 문제가 심각하게 대두되었다. 이러한 문제를 해결하기 위하여 기존의 연방형 구조(Federated Architecture)에서 장비의 사이즈와 무게, 전력소모를 줄여 하나의 컴퓨터에서 다양한 신뢰성 레벨을 갖는 여러 응용을 수행할 수 있는 통합 모듈형 구조(IMA, Integrated Modular Architecture)가 제시되었으며, 현재 항공전자 분야에서는 IMA 구조를 기반으로 한 시스템 개발이 많이 진행되고 있다[1].



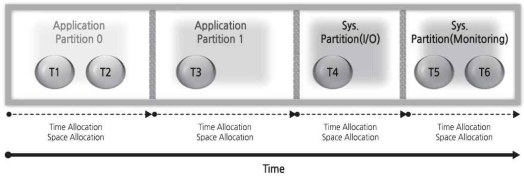
(그림 1) 연방형 구조와 통합 모듈형 구조

IMA 구조를 기반으로 여러 응용들이 하나의 컴퓨터에서 동작할 경우 하나의 응용 오류로 인하여 시스템 전체가 심각한 영향을 받을 수 있는 단점이 있다. 예를 들면, 저렴하게 개발되고 오류의 가능성이 큰 조종석 디스플레이 응용 오류로 인하

여 시스템 전체가 영향을 받아 비행제어와 같은 안전성이 중요한 응용이 수행되지 못해 항공기가 추락할 수 있다. IMA 구조를 지원하기 위하여 하나의 컴퓨터에서 여러 응용이 수행될 때 하나의 응용에서 발생한 오류로 인하여 다른 응용이 영향을 받지 않도록 근본적으로 차단하는 파티셔닝이란 개념이 등장하였고, 이를 포함하여 신뢰성을 높이기 위한 응용과 운영체제 간의 인터페이스(APEX, Application/Executive)에 대한 표준인 ARINC 653(Aeronautical Radio Incorporated 653)이 제정되었다[2].

2.1.2 ARINC 653 기능 구성

ARINC 653은 항공전자 시스템의 운영체제와 응용 간의 인터페이스를 제공하며 필수 서비스인 파트 1, 확장 서비스인 파트 2 및 적합 테스트인 파트 3 등으로 구성된다. ARINC 653의 핵심 서비스는 시/공간 파티셔닝으로 하나의 시스템 내에서 응용프로그램이 실행되는 공간과 시간을 분리시켜서 하나의 응용프로그램의 오류가 다른 응용에 영향을 미치는 것을 방지한다.



(그림 2) 시/공간적 파티셔닝 개념도

ARINC 653 파트 1에서 규정하는 필수 서비스는 (그림 3)과 같이 7개의 기능으로 구성된다.

1) 파티션 관리(Partition Management)는 파티션들이 각각 시간적 공간적으로 분리되어 서로 영향을 미치지 못한다. 파티션들은 시스템 설정에 의해 고정된 스케줄에 따라 반복적으로 실행되며 운영 중에는 스케줄이 변하지 않도록 함으로써 시간적 독립성이 보장된다. 또한 파티션 마다 미리 정해진 메모리 영역을 벗어나지 못하도록 함으로써 공간적으로 독립성을 가지도록 한다.

2) 프로세스 관리(Process Management)는 파티션 내의 프로세스를 관리한다. 프로세스는 파티션 내에서 기능을 동적으로 수행하기 위한 구성단위다. 파티션은 하나 이상의 프로세스로 구성된다. 주기적 프로세스는 일정한 주기를 가지고 반복적으로 실행되며 비주기적 프로세스는 작업을 한 번 실행하고 종료한다. 프로세스는 정지(dormant), 실행(running), 준비(ready), 대기(waiting) 네 가지의 상태를 가지며 프로세스의 상태를 전이시키는 APEX들을 제공한다.

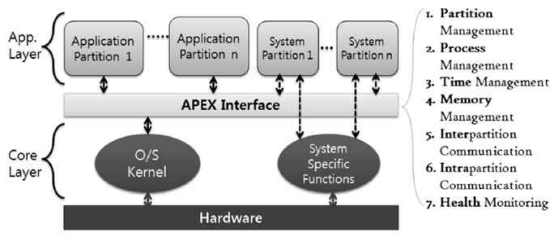
3) 시간 관리(Time Management) 기능은 RTOS에 있어서 가장 중요한 기능으로 손꼽힌다. ARINC 653 표준은 프로세스 대기, 현재 시간 읽기, 데드라인 연장 기능 등 시간 관리 API와 기능 요구사항을 기술하고 있다.

4) 메모리 관리(Memory Management)는 각 파티션이 물리적으로 각기 다른 주소를 갖도록 규정한다.

5) 파티션간 통신(Interpartition Communication)은 파티션의 시간적/공간적 독립성을 훼손하지 않으면서 파티션 간 데이터를 교환하기 위해 고정 길이의 메시지들의 1 : n 교환으로 이루어진다. 파티션 간의 논리적인 통신 링크는 채널이라고 하며, 포트는 특정 채널로 파티션에서 메시지를 보내거나 받기 위해 필요한 자원을 제공한다.

6) 동일 파티션 내의 프로세스들 간의 통신을 위해 파티션 내 통신(Intrapartition Communication) 기능이 사용되며 이는 메시지 교환 기능과 동기화를 위한 기능으로 구성된다. 메시지 교환 기능은 Blackboard와 Buffer가 있고, 동기화를 위한 기능은 Semaphore, Event 및 Mutex가 있다.

7) Health Monitoring(HM)은 하드웨어, 운영체제 및 응용프로그램의 오류 발생 시에 이를 모니터링 및 리포팅하고 처리하는 기능이다. HM은 오류를 격리하고 오류의 확산을 방지한다. 오류를 모듈 레벨, 파티션 레벨 및 프로세스 레벨로 나눠서 관리한다. 3개의 레벨에 대해서 HM callback을 등록하고 에러 처리 프로세스를 생성하기 위한 인터페이스를 제공한다[3][4][5].



(그림 3) ARINC 653 기능 구성

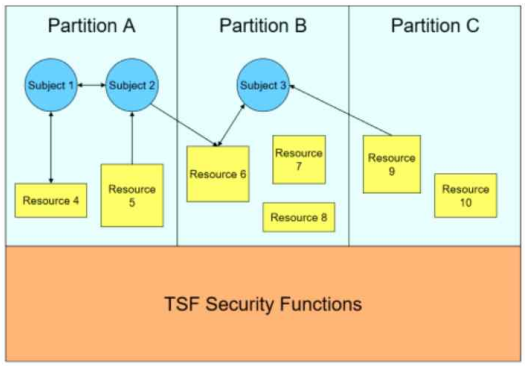
2.2 SKPP

2.2.1 SKPP 개요

SKPP(Separation Kernel Protection Profile)는 정보기기 및 소프트웨어에 대한 보안성을 검증할 수 있는 표준으로 제안된 공통 평가 기준(CC, Common Criteria)에서 정의하고 있는 보호 프로파일(PP, Protection Profile)의 일종으로 특히 운영체제 커널에 대한 공통적인 보안 요구사항을 정의하고 있는 문서이다. 미국 NSA에서는 정보 기기에서 사용되는 운영체제들의 인증을 진행할 수 있도록 가이드가 되는 보호 프로파일로 SKPP를 2007년에 버전 1.03으로 제안하였다.

SKPP는 높은 신뢰도와 안전성을 갖는 운영체제가 가져야 하는 기본적인 보안 사항들과 이를 달성하기 위한 기본 개념들을 설명하고 있다.

2.2.1 SKPP 분리 커널 개념



(그림 4) SKPP 분리 커널의 개념

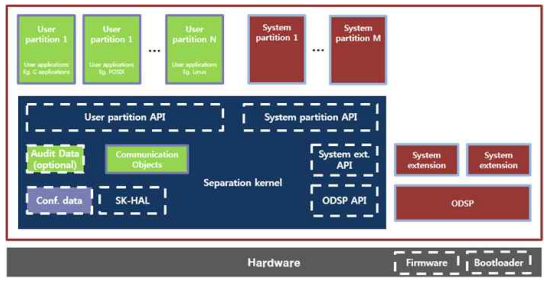
SKPP에서 제시하는 분리 커널(Separation Kernel)은 다음 (그림 4)와 같이 공유 자원과 응용을 가상화하고 이를 분리하여 영향 범위를 제한하도록 하는 것에 그 목적을 두고 있다. 이러한 이유로 분리 커널은 객체들에 대한 새로운 개념을 도입하고 이를 기반으로 커널의 구조를 정의하고 있다.

1) 파티션 : 파티션은 다수의 주체와 자원으로 구성되는 개념적인 객체이다. 일반적으로 파티션 하나는 하나의 기능을 갖는 응용 프로그램으로 구성되며, 파티션을 기반으로 자원에 대한 접근 제어를 수행한다.

2) 주체 : 자원을 사용하는 개체로 운영체제에서는 보통 스레드나 프로세스가 해당된다. 하나의 실행 흐름으로 볼 수 있다.

3) 자원 : 주체가 특정 목적을 위해서 사용하는 개체로 시스템 자원이 될 수도 있고, 커널 객체가 될 수도 있다. 일반적으로는 커널의 객체가 된다.

4) 파티션 정보흐름 정책(PIFP, Partioned Information Flow Policy) : 파티션과 주체, 자원 사이의 정보 흐름을 제어하는 방침으로 이를 기반으로 접근 제어를 수행한다.



(그림 5) SKPP 분리 커널의 SW 구조

SKPP에서는 파티션간의 정보 흐름을 정의하고 이를 PIFP로 나타내며, PIFP는 다음과 같이 두 가지 방식으로 기술될 수 있다.

1) Partition Abstraction : 파티션내의 모든 주체들은 동일한 권한을 갖게 된다. 좀 더 간편한 기술이지만, 보안성 측면에서는 세밀한 권한 부여와 조절에는 적합하지 않다.

2) Least Privilege Abstraction : 모든 주체-공유 자원 짝에 대해서 접근 권한을 따로 기술하는 것으로 세밀한 권한 부여와 조절이 가능하지만, 모든 경우를 다 기술해야 하는 설정에 어려움이 있으며 사용하기에는 불편할 수 있다.

		Partition D Resources			Partition E Resources			Partition F Resources			
		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Partition D Subjects	S1	RW	RW	RW	R	R	R	W	W	W	W
	S2	RW	RW	RW	R	R	R	W	W	W	W
	S3	RW	RW	RW	R	R	R	W	W	W	W
Partition E Subjects	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Partition F Subjects	S4	-	-	-	RW	RW	RW	R	R	R	R
	S5	-	-	-	RW	RW	RW	R	R	R	R
	S6	-	-	-	RW	RW	RW	R	R	R	R

(그림 6) Partition Abstraction이 적용된 PIFP

SKPP에서의 핵심적인 개념은 PIFP이다. 파티셔닝은 시스템 내의 커널 객체 및 자원들에 대한 분할 구조 개념을 의미한다. SKPP에서는 운영체제 내에서의 실행 흐름을 주체로 정의하고, 이 주체들이 사용하는 공유자원을 분할하여 파티션이라는 추상적인 객체로 묶어서 관리하게 된다. 그리고 이 파티션들 사이의 정보 흐름에 대한 정책인 PIFP를 정의하고 이를 강제하여 정보 흐름을 통제하고 강력한 보안성을 유지하게 된다[6].

3. 보안 RTOS 설계 및 구현

3.1 보안 RTOS의 기본 개념 설계

보안 RTOS는 안전성, 보안성을 위한 두 가지 표준인 ARINC 653과 SKPP에서 제시하고 있는 파티셔닝 개념을 기반으로 한 RTOS이다. 실시간성을 제공하면서, 두 가지 표준에서 제시하고 있는 파티셔닝 개념을 효과적으로 지원하여 안전성과 보안성을 강화한 RTOS 구조를 설계하였다.

1) 파티션 구조 : SKPP는 파티션 내부의 구성요소로 주체와 공유 자원을 정의하고, ARINC 653 표준은 파티션이 프로세스의 집합임을 정의하고 있지만, 두 표준의 명확한 차이점은 PIFP의 존재 유무이다. SKPP는 PIFP를 정의하고 있어 공유 자원에 한하여 파티션 외부의 자원에서 접근이 가능하도록 정책을 정의할 수 있는 반면 ARINC 653은 공유자원이 없이 엄격한 파티셔닝을 준수한다는 점에서 차이가 있다.

보안 RTOS는 기본적으로는 파티션 간에는 직접적인 접근이 불가능하도록 하였고, 예외적으로 PIFP를 통한 공유자원의 접근을 지원하도록 설계하였으며, ARINC 653과 동일하게 파티션 내의 프로세스는 모두 같은 권한을 갖도록 하였다.

2) 스케줄링 및 메모리 할당 : ARINC 653은 정적인 스케줄이 시스템에 정해져 있고, 파티션이 배타적인 메모리 영역을 사용한다. 반면에 SKPP는 시간 할당, 메모리 할당에 대한 최소 보장값, 최대 사용 가능치를 정의할 수 있도록 한다.

보안 RTOS에는 ARINC 653과 동일하게 정적인 스케줄과 정적인 메모리 영역을 미리 정의하고 이를 기반으로 응용 프로그램이 구동되도록 하였다. 파티션 내의 프로세스 스케줄링은 ARINC 653에서 정의한 우선순위 기반 스케줄링으로 설계하였다.

3) 예외 처리 : ARINC 653은 헬스 모니터 테이블을 정의하여 각각의 조건에 따라 예외 상황 처리를 모듈 레벨, 파티션 레벨, 응용 레벨로 구분하여 처리할 수 있도록 한다. SKPP는 보안 상태를 위배하지 않도록 시스템 구현자가 예외 발생 시의 동작에 대해 정의할 수 있게 자유도를 부여한다.

보안 RTOS는 시스템마다 예상되는 예외 상황에 대한 예외 처리 동작을 정의할 수 있도록 SKPP 기반 예외 상황 메커니즘을 제공하고, ARINC 653 기반 헬스 모니터 테이블을 통하여 시스템 내에 기본값을 보유하여 예외 상황에 해당하는 테이블의 일부만 조정할 수 있도록 하는 형태로 예외 처리 구조를 설계하였다.

3.2 보안 RTOS의 상태와 모드 설계

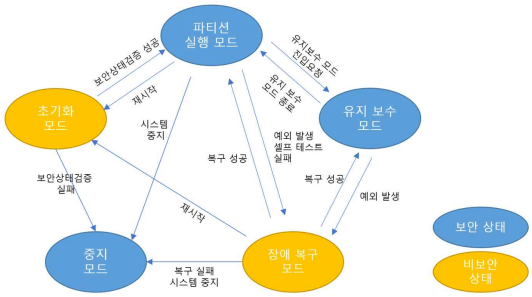
보안 RTOS의 모드는 초기화, 파티션 실행, 장애 복구, 유지보수, 중지 모드로 구분된다.

<표 1> 보안 RTOS의 모드

모드	설명
초기화 모드	시스템이 부팅되고 난 직후 시스템을 초기화 하는 모드

파티션 실행 모드	파티션 응용이 실행 되는 모드
장애 복구 모드	장애가 발생 한 경우 복구 동작을 수행하는 모드
유지보수 모드	인가된 주체에 의해 파티션 수행이 제한되는 모드
중지 모드	시스템 중지 모드

보안 RTOS는 위 5가지 모드와 별도로 보안 상태와 비보안 상태를 가진다. 보안 상태는 사용자에 의해 구성 벡터 형태로 사전 정의되어 있으며, 초기화 모드에서 보안 상태를 검증하여 보안 상태가 성립된 경우, 파티션 실행 모드로 진입한다. 파티션 실행 중 인가된 주체에 의해 유지보수 모드로 진입할 수 있고, 이 모드에서는 사용자에 의해 설정된 명령들이 수행되며 종료한 이후 정상 파티션 실행 모드로 돌아오게 된다. 보안 RTOS는 보안 상태 유지를 위해 지속적으로 시스템을 검사하며, 검사에 실패하거나 예외가 발생하면 비보안 상태인 장애 복구 모드로 진입하여 장애 복구 동작을 수행한다. 장애 복구가 성공하면 보안 상태인 파티션 실행 모드로 다시 진입하여 파티션을 실행한다. 만약 복구가 불가능한 경우라면 중지 모드로 진입하여 시스템을 중지하고 비보안 상태에서 시스템이 계속 수행되는 것을 방지한다. 따라서 보안 RTOS는 비보안 상태에서 파티션 실행이 불가능하다.



(그림 7) 보안 RTOS의 상태와 모드

3.3 보안 RTOS의 설계 및 구현

실시간성을 보장하면서 보안성과 안전성을 제공하기 위한 보안 RTOS의 주요 설계 및 구현 사항은 다음과 같다.

1) 파티션 분리 및 보호 기능

<표 2> 파티션 분리 및 보호 기능 설계

모듈명	설계 및 구현 사항
스케줄링	• 파티션별로 할당된 처리시간 수행을 보장하고, 시간적으로 파티션을 분리 및 보호 ✓ 다중 프로세스 (Multiple processes) : 다수의 코어가 동일한 파티션의 응용 구동 ✓ 다중 파티션 (Multiple partitions) : 각각의 코어가 각각의 파티션을 구동 • 우선순위 기반의 선점형 쓰레드 스케줄링(Preemptive priority-based scheduling) 사용 • 커널은 프로세서가 제공하는 특권모드(privileged mode)에서 동작, 파티션의 쓰레드는 사용자 모드(User mode)에서 동작하도록 구분
메모리 관리	• 파티션별 메모리 할당을 보장하여 공간적으로 파티션을 분리 및 보호 • MMU를 이용하여 파티션 메모리를 보호 ✓ 파티션 영역 : 다른 파티션의 접근이 완전히 차단 ✓ 공유 메모리 영역 : 구성 벡터에서 설정한 대로 메모리의 접근이 가능하도록 설정 • 커널과 사용자 파티션은 분리된 주소 영역으로 할당되며, 커널 모드에서는 현재 구동 중인 파티션의 메모리 영역에 대해 접근이 가능 • 코드 영역 보호를 위해 코드와 데이터 영역을 구분하여 할당, 코드 영역은 하드웨어에 쓰기 권한을 제거하여 잘못된 접근으로부터 보호

2) 이벤트 처리 기능

<표 3> 이벤트 처리 기능 설계

모듈명	설계 및 구현 사항
타이머 처리	• 스케줄링, 셀프테스트, 커널무결성 모니터링을 위한 타이머처리 기능 제공, 타이머 인터럽트 발생시 타이머처리 함수를 호출하여 수행 • 최대 타이머 개수를 제한하여 복잡도를 줄이는 방법으로 틱리스 타이머(다음에 있을 이벤트를 계산하여 필요

	한 이벤트가 있을 때 수행하는 방식)를 사용
예외 처리	• 시스템 콜, 하드웨어 예외, 소프트웨어 예외를 포함한 시스템 예외상황 처리 • IPC 시스템 콜 예외인 경우 IPC 서비스를 호출하고, 보안서비스 시스템 콜인 경우 보안서비스를 호출 • 하드웨어 및 소프트웨어 예외인 경우 파티션 또는 커널 구동 중 예외 발생에 따라 장애복구를 진행

3) 커널 서비스 기능

<표 4> 커널 서비스 기능 설계

모듈명	설계 및 구현 사항
보안 서비스	• 인가된 주체에 의해 수행되는 보안 서비스 기능 ✓ 시스템 재시작, 시스템 중지, 셀프테스트 실행 등 • 보안서비스 시스템 콜이 발생한 경우 권한을 가지고있는지 접근통제 모듈을 통해 확인한 후 서비스 허용
IPC	• 구성데이터에 정의된 공유자원(공유메모리)을 할당하고 접근/해제 • IPC 시스템 콜이 발생한 경우 요청한 주체가 공유자원에 권한을 가지고있는지 접근통제 모듈을 통해 확인

4) 구성관리 및 접근통제 기능

<표 5> 구성관리 및 접근통제 기능 설계

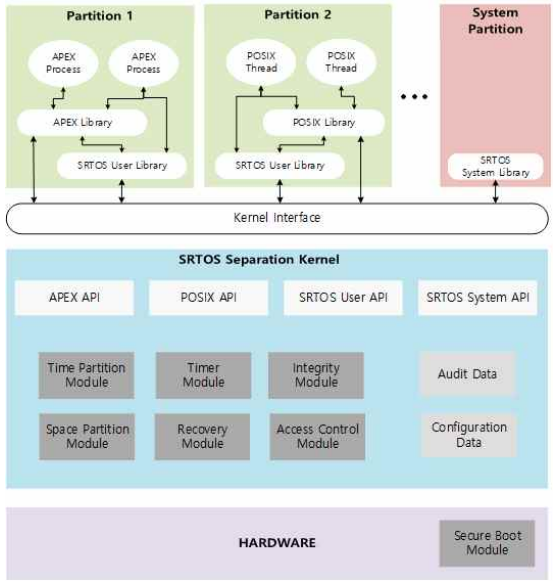
모듈명	설계 및 구현 사항
구성 관리	• 파티션, 주체, 공유자원 등 시스템의 유효성 및 무결성 검사와 관련된 구성데이터 관리 ✓ 주체 권한 ✓ 정보흐름통제 권한 등
접근 통제	• 구성데이터에 설정된 파티션 간 정보흐름 정책에 따라 주체와 공유자원의 정보흐름 통제 • 구성데이터에 설정된 보안서비스(시스템 재시작, 시스템 중지, 셀프테스트 실행 등) 실행 권한 확인

5) 보안 상태관리 기능

<표 6> 보안 상태관리 기능 설계

모듈명	설계 및 구현 사항
보안 경보	• 보안 위반 탐지 시 사용자 콜백을 호출
보안 상태 검증	• 구성데이터에서 설정된 보안 상태의 검증 • 커널무결성 검증, 구성데이터무결성 검증, 기본 하드웨어 테스트 수행 ✓ 기본 하드웨어 테스트 : 메모리 데이터버스 테스트, 메모리 주소버스 테스트 등
장애 복구	• 구성데이터에 설정된 장애복구 동작에 따라 장애복구 기능을 수행 ✓ 예외 무시, 시스템 중지, 시스템 재시작, 파티션 재시작 등
셀프 테스트	• 응용 요청이나 주기적 검사에 의해 커널무결성, 응용무결성 검증, 구성데이터무결성 검증을 수행 • 셀프테스트가 실패한 경우 보안경보 발생하고, 장애복구 수행
보안 감사	• 타임스탬프를 이용하여 감사사건을 기록, 관리

보안 RTOS의 SW 동작 구조는 다음과 같다.



(그림 8) 보안 RTOS SW 동작 구조

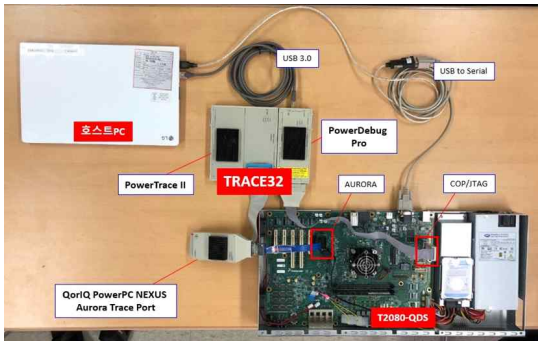
4. 보안 RTOS 성능 측정 및 분석

4.1 시험 환경 구성

보안 RTOS의 성능 측정은 아래와 같은 시험 환경에서 수행하였다. 타겟 환경은 국방/항공분야에서 많이 사용하는 PowerPC 프로세서 기반의 평가보드를 선정하고, 정밀한 수행시간 측정을 위하여 디버깅 전용 장비를 활용하였다. 또한 성능의 비교를 위하여 상용 RTOS를 동일한 하드웨어에서 동일한 조건의 시험 응용을 구동하여 성능 측정하였다.

<표 7> 성능 측정 시험 환경

구분	성능 측정 환경
타겟 하드웨어	NXP T2080QDS
시험 대상 운영체제	보안 RTOS / 상용 RTOS
시간 측정 도구	TRACE32 PowerTrace II
	TRACE32 PowerDebug PRO
	TRACE32 QorIQ PowerPC NEXUS Aurora Trace Port
호스트 PC	OS : Windows 10



(그림 9) 성능 측정 시험 환경 구성

4.2 시험 항목 및 내용

성능 측정 항목은 RTOS 성능 측정 관련 참고문헌을 기반으로 RTOS에서 사용도가 높은 API 및 기능을 선별하여 선정하였다[8][9][10][11]. TD-001~TD-003 시험 항목은 RTOS의 실시간성을 측정하기 위한 항목이고, TD-004~TD-007은 RTOS의 통신 기능 성능을 측정하기 위한 항목이다. 추가적으로 메모리 할당/해제 성능(TD-008)과 드라이스톤 벤치마크(TD-009)을 측정하였다. 상세한 측정 항목 및 내용은 <표 8>과 같다.

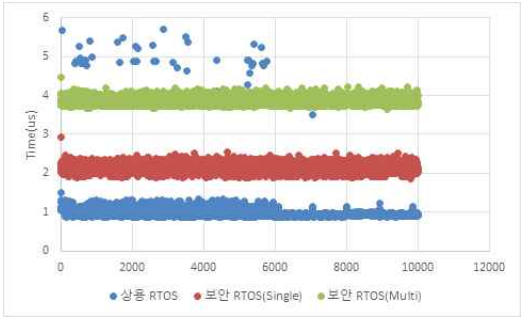
<표 8> 성능 측정 시험 항목

[식별자] 시험 항목	시험 내용
[TD-001] 파티션 간 문맥 전환 지연시간	• 실행 중인 파티션이 스케줄러에 의해 다른 파티션으로 제어권이 넘어가는 문맥 전환 지연시간을 측정(1천회)
[TD-002] 태스크 간 문맥 전환 지연시간	• 실행 중인 태스크의 제어권을 다른 태스크에게 넘기는 태스크 간 문맥 전환 지연시간을 측정(1만회)
[TD-003] 인터럽트 응답시간	• 인터럽트 발생시 커널에서 인터럽트 핸들러로 진입하는 인터럽트 응답시간을 측정(1만회)
[TD-004] 세마포어 대기/해제 지연시간	• 제어권 획득을 위한 세마포어 대기와 이를 해제하는 지연시간을 측정(1만회)
[TD-005] 태스크 간 세마포어 해제 지연시간	• 태스크 간 획득한 세마포어를 해제하는데 소요되는 지연시간을 측정(1만회)
[TD-006] 버퍼 전송 지연시간	• 실행 중인 태스크의 send 함수로 다른 태스크의 receive 함수 호출 시 문맥 간 전환과 전송 지연시간을 측정(1만회)
[TD-007] 샘플링 포트 전송 지연시간	• 실행 중인 파티션에서 다른 파티션으로 송수신 동작을 통하여 데이터 송수신 과정 중 수행되는 시간을 측정(1만회)
[TD-008] 메모리 할당/해제 지연시간	• 실행 중인 태스크에서 특정 시점에서 메모리 할당과 해제를 시도할 시에 지연시간을 측정(1천회)
[TD-009] 드라이스톤 벤치마크	• 드라이스톤 벤치마크 프로그램 수행 시간을 측정(1백회)

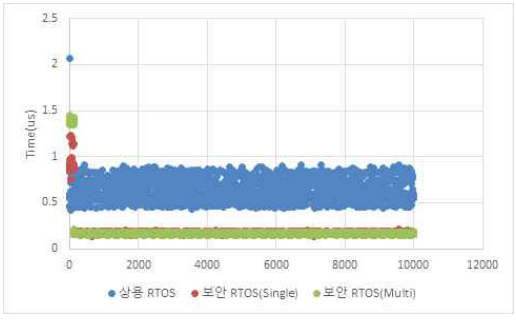
4.3 시험 결과

보안 RTOS와 상용 RTOS의 성능 측정 시험 항목에 대한 시험 결과는 다음과 같다.

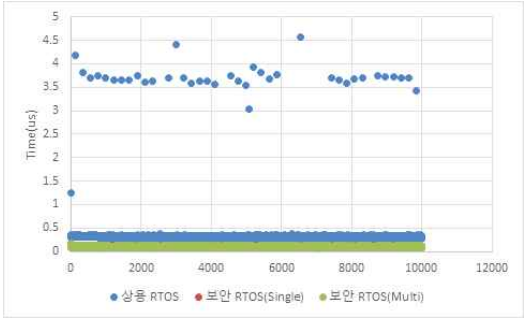
(그림 10) TD-001 파티션 간 문맥 전환 지연시간



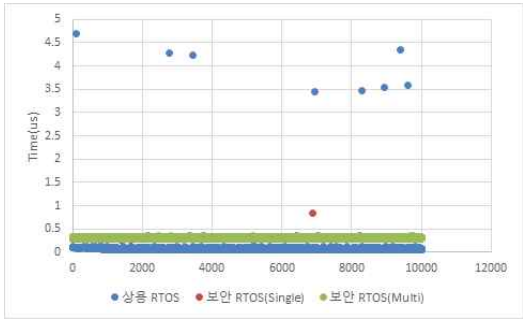
(그림 11) TD-002 태스크 간 문맥 전환 지연시간



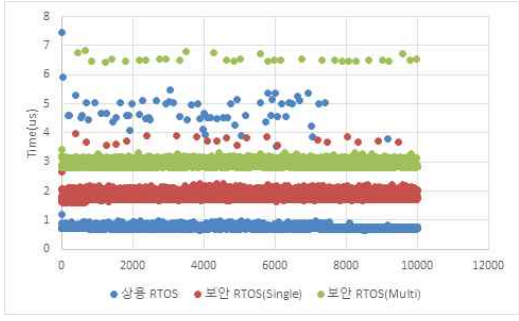
(그림 12) TD-003 인터럽트 응답시간



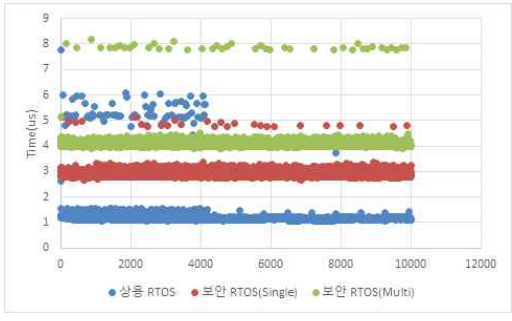
(그림 13) TD-004 세마포어 대기/해제 지연시간 – 대기



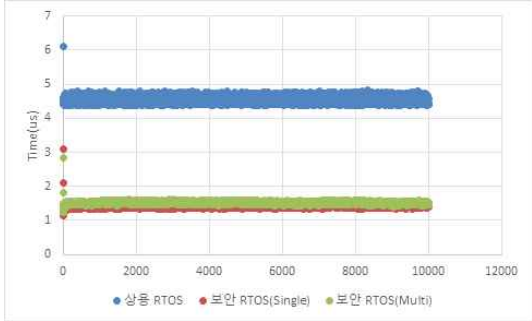
(그림 14) TD-004 세마포어 대기/해제 지연시간 – 해제



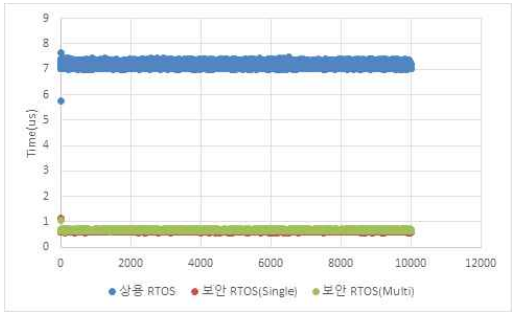
(그림 15) TD-005 태스크 간 세마포어 해제 지연시간



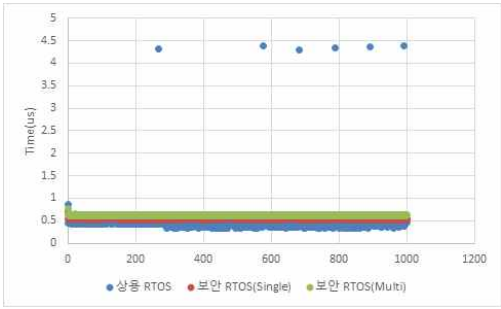
(그림 16) TD-006 버퍼 전송 지연시간



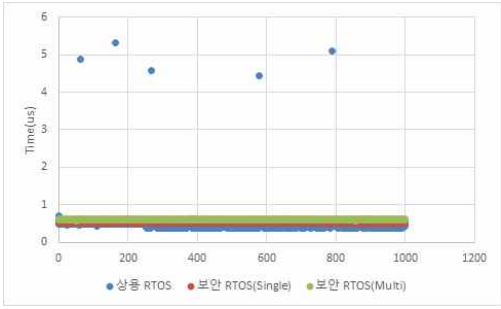
(그림 17) TD-007 샘플링 포트 전송 지연시간 – 송신



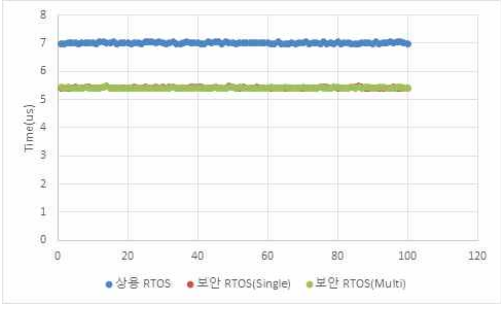
(그림 18) TD-007 샘플링 포트 전송 지연시간 – 수신



(그림 19) TD-008 메모리 할당/해제 지연시간 – 할당



(그림 20) TD-008 메모리 할당/해제 지연시간 – 해제

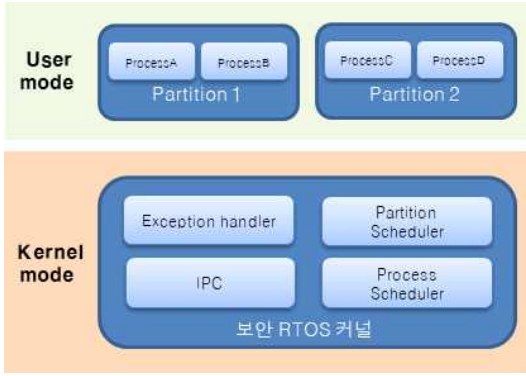


(그림 21) TD-009 드라이스톤 벤치마크

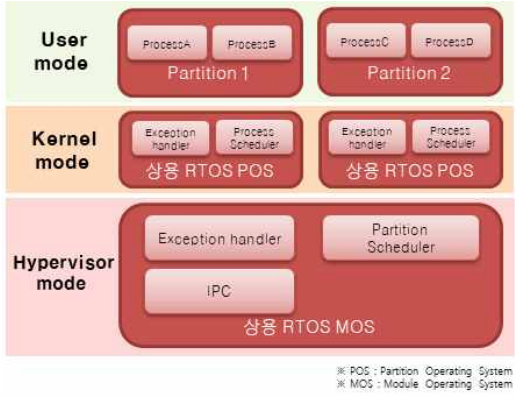
4.4 시험 결과의 분석

4.4.1 보안 RTOS와 상용 RTOS의 구조 비교

성능의 비교를 위하여 비교 대상으로 사용한 상용 RTOS와 보안 RTOS의 구조를 비교하면 (그림 22), (그림 23)과 같다.



(그림 22) 보안 RTOS 커널 구조

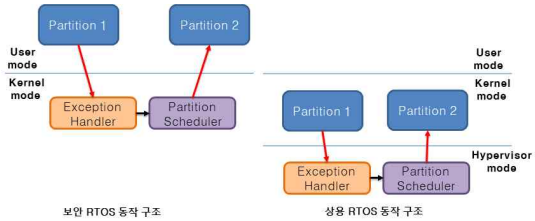


(그림 23) 상용 RTOS 커널 구조

보안 RTOS에서 파티션 코드는 사용자 수준에서 동작하고, 커널 코드는 커널 수준에서 동작하지만, 상용 RTOS에서 파티션 및 POS(Partition Operating System) 코드는 커널 수준에서 동작하고, MOS(Module Operating System) 코드는 하이퍼바이저 수준에서 동작한다[12].

4.4.2 TD-001, TD-003, TD-004-대기, TD-007

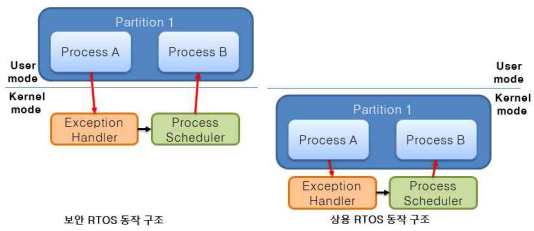
시험 결과와 같이 보안 RTOS는 파티션 간 전환, 인터럽트 응답, 세마포어 대기, 샘플링 포트 전송 시간에 있어 상용 RTOS와 동등하거나 우월한 성능을 보여주었다. 특히, 파티션 간 전환 시간의 경우는 커널 수준에서 구현한 파티션 관리 기능과 파티션을 게스트 OS로 구현한 상용 RTOS와의 구조적인 차이로 인하여 현격한 성능 차이를 보임을 확인할 수 있었다. 하이퍼바이저 수준에서의 문맥 전환은 커널 수준의 문맥에 가상머신 상태 문맥을 추가적으로 저장하므로 상대적으로 높은 오버헤드를 갖는다.



(그림 24) 파티션 간 문맥전환 동작 구조

4.4.3 TD-002, TD-004-해제, TD-005, TD-006

프로세스 변경은 보안 RTOS와 상용 RTOS 모두 커널 수준으로 수행한다. 보안 RTOS는 프로세스들이 사용자 수준에서 동작하는 반면, 상용 RTOS에서는 프로세스들이 커널 수준에서 동작하므로, 사용자 수준으로 변경이 필요 없는 상용 RTOS가 보안 RTOS 대비 좋은 성능을 보인다. 이 성능 차이는 보안성을 위하여 커널 모드와 사용자 모드를 구분하여 동작하도록 한 보안 RTOS 커널의 구조에서 기인하는 성능의 차이이며, 이는 마이크로초 단위의 성능 차이로 실시간 시스템을 개발하는 데는 그다지 큰 영향을 주지 않을 것으로 평가된다.



(그림 25) 태스크 간 문맥전환 동작 구조

4.4.4 TD-008

메모리 할당/해제 지연시간의 성능 차이는 관련 라이브러리의 최적화 차이로 예상된다. 보안 RTOS는 최적화하지 않은 범용 임베디드 C 라이브러리의 메모리 관리 기능을 사용하나, 상용 RTOS는 자체의 기술로 최적화된 메모리 할당/해제 루틴을 사용할 것으로 예상되므로, 보안 RTOS의 관련 루틴 최적화를 통하여 성능 개선이 가능할 것으로 보인다.

4.4.5 TD-009

버퍼 전송 테스트의 경우 송신/수신 스레드가 모두 대기없이 데이터의 R/W가 수행되므로 비슷한 성능을 보여야 하며, 드라이스톤은 CPU 성능 측정 워크로드로 CPU 성능이 같으면 비슷한 결과가 측정되어야 정상이다. 본 시험의 결과 측정된 두 커널의 성능 차이는 틱 기반 커널, 틱 리스 커널의 차이로 발생된 것으로 예상되며, 틱리스 커널인 보안 RTOS와 달리 상용 RTOS의 경우 타이머 틱기반 커널로 매 틱마다 커널이 선점하여 타이머 인터럽트 처리를 수행하게 되므로 낮은 성능을 보인 것으로 추측된다.

5. 결 론

RTOS는 자동차, 항공, 국방 등 분야에서 실시간 반응성, 경량 SW가 요구되는 Mission-Critical 시스템이나 Safety-Critical 시스템의 핵심 기술로 적용되고 있다. 현대의 무기체계는 SW 복잡도와 연결성이 증가되고 있고, 고성능 하드웨어 안에서 다수의 응용들이 동시에 안전하게 구동되는 시스템으로 변모하고 있으며, 전방위적인 사이버보안 위협의 대상이 되고 있다.

본 논문에서는 안전성과 보안성을 위한 두가지 표준인 ARINC 653과 SKPP에서 정의하고 있는 파티셔닝 개념을 기반으로 안전성과 보안성을 모두 강화한 보안 RTOS의 설계 및 구현, 성능 측정 결과를 제시하였고, 그 결과 보안성을 강화하였음

에도 실시간성 및 성능이 동등한 수준임을 확인하였다. 보안 RTOS의 대표적인 특징은 다음과 같다.

1) 파티셔닝 기반의 보안 실시간 운영체제 : 정보기기 및 Safety-Critical 시스템에서 공통적으로 요구되는 파티셔닝 개념을 기반으로 하여 보안성과 안전성을 모두 보장하는 보안 실시간 운영체제다.

2) 하이브리드 커널 구조를 통한 강화된 안전성 및 보안성 : 마이크로 커널 구조와 모놀리틱 커널의 장점을 결합하여 성능과 보안의 두 가지 요구사항을 모두 만족할 수 있도록 하였다.

3) Safety-Critical 시스템에 최적화된 실시간성 : 보안 RTOS는 예측 가능하고 결정적인(Predictable and Deterministic) 작업 스케줄링을 위한 최적의 하이브리드 커널 구조로 보안성과 안전성을 모두 만족하면서도 실시간성을 보장할 수 있도록 하였다.

향후 보안 RTOS의 실 무기체계 적용을 위하여 BSP, 사용자 라이브러리 및 필수 지원 서비스 확장 개발과 실제 운용 중인 무기체계를 대상으로 한 시험 개발을 통한 무기체계 적용성 및 보안성 검증이 필요하다.

참고문헌

[1] 손동환, “항공기 시스템의 SW를 위한 표준”, TTA Journal, Vol.154, pp.38-43, 2014.

[2] P.J. Prisaznuk, “ARINC 653 role in Integrated Modular Avionics”, In the Proceedings of the IEEE/AIAA 27th Digital Avionics Systems Conference, pp1.E.6-1~1.E. 5-10, 2008

[3] AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE PART 1 REQUIRED SERVICES, SAE-ITC, August 21, 2015.

[4] 김태호, 손동환, 신창민, 박사천, 임동혁, 이화영, 김병호, “Qplus-AIR의 DO-178B 인증 경험”, 정보과학회지, 제31권, 제5호, pp.32-39, 2013.

[5] 손동환, “고신뢰 시스템을 위한 소프트웨어 플랫폼 요구사항 및 발전 방향”, 정보과학회지, 제33권, 제12호, pp.20-23, 2015.

[6] U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, version 1.03, 2007.

[7] 임동혁, 서경덕, 이상훈, “보안성과 안전성을 동시에 고려한 파티션 기반 보안 실시간 운영체제의 설계”, 한국군사과학기술학회 종합학술대회 논문집, pp.1518-1519, 2019.

[8] W.A. Halang, R. Gumzej, M. Colnaric, M. Druzovec, “Measuring the Performance of Real-Time Systems”, The International Journal of Time-Critical Computing Systems, 18, pp.59-68, 2000.

[9] S.K. Nisar, M. Ahmed, H. Ayub, I. Baig, “Operating System Performance Analyzer for Low-End Embedded Systems”, International Journal of Computer Science Issues, Vol. 8, Issue 6, No 3, pp.341-348, November 2011.

[10] K. Weiß, T. Steckstor, W. Rosenstiel, “Performance Analysis of a RTOS by Emulation of an Embedded System”, Proceedings of the Tenth IEEE International Workshop on Rapid System Prototyping, June 1999.

[11] M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll, A. Crespo, J.J. Metge, “LithOS: a ARINC-653 guest operating for XtratuM”, 12th Real-Time Linux Workshop, October 2010.

[12] https://resources.windriver.com/i/1018991-safety-critical-software-development-for-integrated-modular-avionics/0?.

[13] 임동혁, 설진호, 서경덕, 이상훈, “무기체계를 위한 보안 실시간 운영체제의 성능 분석”, 한국군사과학기술학회 종합학술대회 논문집, 2021.

[14] 이상훈, 권미영, 김훈규, 강태인, 이성기, 이노복 “파티션 기반 보안 실시간 운영체제에서 무결성 검증 및 장애 복구 방법”, 한국군사과학기술학회 종합학술대회 논문집, pp.1139-1140, 2020.

[저 자 소 개]

서 경 덕 (Kyungdeok Seo)
2001년 8월 경희대학교 전자계산공학과 학사
2003년 8월 경희대학교 전자계산공학과 석사
email : kyungdeok.seo@hanwha.com

이 우 진 (Woojin Lee)
2018년 7월 부산대학교 컴퓨터공학부 학사
email : holinder4s@gmail.com

채 병 민 (Byeongmin Chae)
2007년 2월 충남대학교 물리학과 학사
2012년 2월 충남대학교 컴퓨터공학과 석사
email : byeongmin.chae@hanwha.com

김 훈 규 (Hoonkyu Kim)
1986년 2월 홍익대학교 전자계산학과 학사
1992년 2월 홍익대학교 전자계산학과 석사
2015년 8월 홍익대학교 컴퓨터공학과 박사
email : hunk@add.re.kr

이 상 훈 (Sanghoon Lee)
1994년 2월 홍익대학교 컴퓨터공학과 학사
1996년 2월 홍익대학교 전자계산학과 석사
email : shljhl@add.re.kr