Transactions of KSAE, Vol. 30, No. 11, pp.901-907 (November, 2022)

pISSN 1225-6382 / eISSN 2234-0149
DOI http://dx.doi.org/10.7467/KSAE.2022.30.11.901

차량 소프트웨어 통합을 위한 하이퍼바이저 인터럽트 지연 분석

오 성 빈[1] · 김 진 호[*2]

경남대학교 융합IT공학과[1] · 경남대학교 컴퓨터공학부[*2]

An Analysis on Interrupt Latency of Hypervisor for Automotive Software Integration

Sung Bhin Oh[1] · Jin Ho Kim[*2]

[1]Department of Convergence IT Engineering, Kyungnam University, Gyeongnam 51767, Korea
[2]School of Computer Engineering, Kyungnam University, Gyeongnam 51767, Korea
(Received 10 June 2022 / Revised 4 August 2022 / Accepted 10 August 2022)

Abstract：The increasing number of automotive software and their corresponding complexity is testing the scalability of the current E/E architecture. A domain-oriented architecture based on multi-core processors can overcome this limitation by consolidating multiple Electronic Controller Units(ECUs) into a few Domain Controller Units(DCUs). According to the ISO 26262 automotive safety standard, the integrated system should ensure full separation between software components. Virtualization technology can isolate software and its faults to meet these requirements. Furthermore, it could solve the re-engineering and validation cost problem in multi-core system migration. However, since current virtualization technology has low real-time performance compared to a native system, it is challenging to apply a virtualization platform to an automotive embedded system, especially automotive real-time systems, which require overcoming hard, real-time constraints of safety-critical system functions. In this paper, we will be measuring and evaluating the I/O latency of full and para-virtualized embedded systems. In our analysis, we will be using the ARM Cortex-A72 pi board, and implement the Xen and QEMU/KVM hypervisor. We will be showing the I/O latency time of each virtualization environment, and analyze the cause of the delay through experiments.

Key words：Virtualization(가상화), Automotive(자동차), E/E architecture(전기/전자 아키텍처), I/O latency(I/O 지연), Embedded hypervisor(임베디드 하이퍼바이저)

1. 서 론

오늘날의 자동차에는 사용자들의 다양한 요구사항을 만족하기 위해서 다양한 기능의 ECUs(Electronic Control Units)가 탑재되고 있으며, 이러한 ECU에 할당되는 차량 임베디드 소프트웨어의 복잡도와 그 개수는 계속해서 증가하고 있다.[1-4] 기존 차량의 전기/전자 아키텍처는 일반적으로 기능 하나를 구현하기 위해 한 개의 ECU가 추가되는 형태로, 현재 고급 차량의 경우 100개 이상의 ECU가 장착되어 있으며, 각 ECU는 CAN, FlexRay, Ethernet 등의 IVN(In-Vehicle Network) 프로토콜을 통해 여러 버스에 연결된다.[5] 이러한 구조에서 ECU 개수 증가는 공간 제한, 네트워크 및 배선 복잡도 증가, 소비 전력 및 비용 상승과 같은 다양한 문제를 초래하였다. 또한, 증가하는 시스템 복잡도로 인해 기존 차량 전기/전자 아키텍처의 확장성 역시 제한된다.[6]

이러한 ECU 증가 문제와 차량 아키텍처의 확장성 문제를 극복하기 위해 도메인 지향의 DCU(Domain Controller Unit) 기반 아키텍처가 제안되었다.[6,7] DCU 기반 아키텍처는 파워트레인, 새시, 인포테인먼트 등의 기능 기반으로 고성능의 멀티코어 프로세서를 활용하여 다수의 ECU를 하나의 DCU로 통합하는 방법으로, 총 ECU 개수를 줄이고 아키텍처의 새로운 기술에 대한 아키텍처의 확장성을 높여줄 수 있다.[8] 또한, 이러한 DCU 기반 아키텍처에서 발전하여 영역 기반으로 기능을 그룹화하는

*Corresponding author, E-mail: kimjh@kyungnam.ac.kr

Zonal 아키텍처 역시 여러 기능과 서비스를 하나의 ECU에 통합한다.[9)]

분산되어 있는 각기 다른 소프트웨어를 통합할 때, 이러한 소프트웨어 기능 차이를 처리하는 것은 예측하지 못한 문제를 발생시킬 수 있다. ISO26262 표준은 자동차의 안전 관련 전기/전자 기능에 대한 국제 규격이다.[10)] 차량의 기능 안전성을 위한 위험 분류를 나타내는 ASIL (Automotive Safety Integrity Level)은 ISO2622 표준의 핵심 요소이다. 높은 ASIL 등급을 가지는 차량 소프트웨어는 기능안정성을 위해 더 많은 요구사항을 달성해야 한다. 해당 표준에 따르면 서로 다른 다수의 소프트웨어들을 하나의 시스템으로 통합할 경우 통합되는 소프트웨어 간의 영향을 미치는 부분이 없는지 검증하여, 서로 간의 영향이 있을 경우 통합되는 소프트웨어 중 가장 높은 ASIL 등급을 기준으로 모든 소프트웨어를 개발하고 검증하여야 한다. 멀티코어 기반에 통합 방식은 ISO26262를 충족하는데 필요한 병렬 처리를 제공할 수 있다.[11)] 하지만 이러한 방식은 소프트웨어 이식 및 통합된 소프트웨어 검증을 필요로 하여 높은 개발 비용을 초래한다. 특히, 안전과 관련된 소프트웨어를 통합하는 경우 더 많은 노력을 필요로 한다.[12)]

가상화는 하나의 물리적 장치에서 여러 운영체제를 동시에 실행할 수 있게 하는 기술로, 서버나 데이터 센터에서 핵심기술로 사용되고 있으며 최근 모바일 혹은 차량과 같은 임베디드 환경에서도 주목받고 있다. 가상화 시스템에서 소프트웨어 혹은 운영체제는 가상 머신(Virtual Machine, VM)으로 동작하며, 각 가상머신은 독립적으로 수행되고 특정 가상머신의 오류가 다른 가상 머신으로 전파되지 않도록 각 가상 머신의 격리가 보장된다. 이러한 격리 기능은 소프트웨어 통합 시 안전을 위해 요구되는 소프웨어 간 분리를 실현하여 신뢰성을 확보할 수 있다.[13)]

가상화 환경은 하드웨어와 어플리케이션 소프트웨어 사이에 가상화 계층이 추가되는 형태로, 이에 따른 오버헤드가 발생하여 성능 저하가 뒤따른다.[14)] 이러한 성능 저하는 가상화 기술을 임베디드 실시간 시스템에 적용하는 것을 어렵게 한다. 실시간 시스템은 미리 정의된 타이밍 제약 조건을 준수해야 하며 결정론적이여야 한다. 특히, 차량 시스템은 타이밍 제약 조건을 지키지 못할 경우 치명적인 위험과 직결되는 안전 관련 어플리케이션이 존재하며, 이러한 어플리케이션들에게 요구되는 실시간성이 보장되어야만 한다. 가상화 환경의 성능저하로 인해 발생하는 인터럽트의 지연시간이 일관성 있지 않은 경우, 각 어플리케이션의 수행 시간 예측을 어렵게 하며, 지연으로 인한 시스템 처리의 수행 시간증가는 엔진과 같이 반드시 정해진 시간 내에 수행되어야 하는 경성 실시간 시스템(Hard real-time)에서 데드라인 내에 처리하는 것을 어렵게 한다. 현재 자동차에서 가상화 기술은 실시간이 중요하지 않은 네비게이션과 바디제어 기능과 같은 리눅스 기반의 소프트웨어를 하나의 제어기에 통합하는 용도로만 사용되고 있다.

본 논문은 임베디드 가상화 환경에서 인터럽트 지연에 대해 논의한다. 가상화를 실행하는 하이퍼바이저와 가상화 방식에 따른 인터럽트 지연시간의 차이를 측정하여 차량 가상화 시스템을 위한 가상화 기술 적용 방법을 제시한다. 이를 위해 서버 시장에서 인기 있는 오픈소스 하이퍼바이저인 Xen과 KVM을 사용하여 임베디드 가상화 환경을 구성하고, 해당 환경에서 인터럽트 지연을 측정하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 본 논문과 관련된 연구와 가상화 기술의 배경 지식에 대해서 간략하게 설명한다. 3장에서는 가상화 실험 환경과 방법을 설명하고 측정된 실험 결과를 제시한다. 4장에서는 논문의 결론을 맺고 향후 연구 방향을 제시한다.

2. 배경 지식 및 관련 연구

2.1 DCU 기반 차량 전기/전자 아키텍처

기존의 분산형 차량 전기/전자 아키텍처는 컴퓨팅 성능이 낮은 개별 ECU가 하드웨어의 한 부분만을 제어하도록 설계되었다. 복잡한 기능을 지원하기 위해 차량의 탑재되는 ECU의 개수가 증가하면서, 차량 내부 시스템의 복잡도 증가 및 자동차의 공간적 한계 등의 문제로 전기/전자 아키텍처의 확장성이 제한되었다.

이러한 문제점을 해결하기 위해 소수의 강력한 성능을 가진 DCU를 사용하여 다수의 ECU를 통합하는 DCU 기반 전기/전자 아키텍처가 제안되었다. Fig. 1은 DCU 기반 전기/전자 아키텍처의 개요를 보여준다. ECU를 파워트레인, 새시, 바디, 멀티미디어와 같은 기능 별로 통합 및 하나의 도메인으로 그룹화된다. 이러한 도메인은 DCU에 의해 제어되며, DCU는 고성능 프로세서를 활용하여 해당 도메인의 기능을 통합하고 높은 성능을 제공한다.

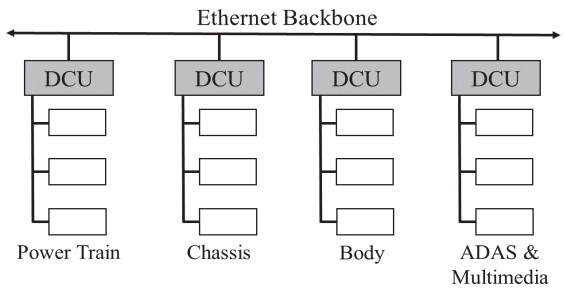


Fig. 1 DCU oriented E/E architecture

2.1.1 멀티코어 프로세서 기반 DCU

DCU 기반 아키텍처의 개념은 멀티코어 프로세서가 가지는 높은 수준의 성능을 바탕으로 한다. 많은 소프트웨어 구성 요소가 DCU에서 실행되기 때문에 이를 처리할 컴퓨팅 성능이 필요하며 멀티코어 시스템은 이러한 요구사항을 달성할 수 있는 계산능력을 제공한다. 이를 통해 여러 ECU를 통합하여 차량에 탑재되는 총 ECU 개수를 줄일 수 있으며 전기/전자 아키텍처의 확장성을 확보할 수 있다.

소프트웨어 통합 시 구성 요소들을 서로 격리하는 것은 차량의 안전과 ISO26262 표준을 만족하는데 중요한 사항이다. 통합 시스템에서 각 요소들은 외부의 간섭으로부터 보호되어야 한다. 멀티코어 시스템은 높은 수준의 병렬처리를 제공하여, 소프트웨어 간 격리를 실현할 수 있다. 하지만 멀티코어 프로세서는 격리된 계산 코어를 제공할 뿐 캐시, 보조 프로세서, 네트워크 I/O 인터페이스와 같은 주요 부분을 여전히 공유하여 안전한 격리를 보장하지 못한다.[11] 또한 싱글코어 시스템에서 실행되도록 설계된 레거시 소프트웨어들을 멀티코어 시스템에서 병렬로 실행되도록 하기 위한 수정은 힘든 작업이며, 이러한 수정은 멀티코어 환경에서 소프트웨어에 대한 전체적인 재검증을 필요로 하여 높은 금전적, 시간적 비용 문제를 초래하기 때문에 차량 소프트웨어 통합을 어렵게 한다.

2.2 가상화 개요

가상화는 가상 머신(Virtual Machine, VM)을 생성하여 효율적인 하드웨어 자원 사용과 가상 머신 간의 격리를 보장하는 기술로 현재 서버 시장은 물론 모바일, 항공 및 임베디드 분야에서도 주목받고 있다. 하이퍼바이저(Hypervisor)는 이러한 가상화를 구현하기 위해 호스트 시스템에서 여러 가상 머신을 실행할 수 있게 하는 소프트웨어로, VMM(Virtual Machine Monitor)라고도 불린다.

하이퍼바이저는 Fig. 2에 나타난 것과 같이 베어 메탈형과 호스트 기반형으로 분류될 수 있다. 베어 메탈 하이퍼바이저는 소프트웨어가 호스트 하드웨어 위에서 직접 동작하며, 하이퍼바이저 위에서 실행되는 가상머신을 관리한다. 호스트 기반 하이퍼바이저는 호스트 하드웨어 위에서 기존의 호스트 운영체제가 실행되고, 그 운영체제 위에서 하이퍼바이저가 일반적인 소프트웨어처럼 실행되어 가상머신을 관리하는 형태이다. 베어 메탈형은 호스트 기반형과 달리 직접 하드웨어를 제어할 수 있기 때문에 오버헤드가 낮아 호스트 기반형에 비해 낮은 성능 저하를 보여준다.

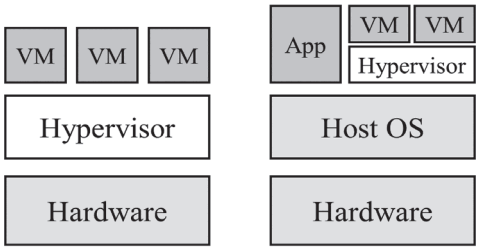


Fig. 2 Hypervisor-based system architecture. (a) Bare metal hypervisor, (b) Hosted hypervisor

시스템 가상화 방식은 일반적으로 수정되지 않은 소프트웨어 혹은 운영체제를 실행하는 전가상화 방식과 가상화되는 소프트웨어나 운영체제가 하이퍼바이저의 인터페이스를 사용하도록 수정하는 반가상화 방식이 존재한다. 전가상화 방식은 가상된 시스템의 명령을 하이퍼바이저가 트랩과 에뮬레이션을 사용하여 조정한다. 때문에 가상화되는 시스템에 수정이 필요하지는 않지만 상당한 오버헤드가 발생한다. 반가상화 방식은 가상화되는 소프트웨어나 운영체제를 수정하여 가상화된 시스템이 특정 요청에 대해서 하드웨어 레벨에 접근을 위한 요청(Hypercall)을 가능하게 하여 오버헤드를 줄인다. 하지만 이러한 수정이 비용 혹은 소스 코드에 문제로 불가능할 수 있다.

현재 상용화 되어있는 임베디드 하이퍼바이저로는 ETAS 사의 ETAS RTA lightweight hypervisor, OpenSynergy 사의 COQOS hypervisor, Green HIlls Software 사의 INTEGRITY multivisor 등이 존재한다. 이러한 임베디드 하이퍼바이저들 자동차의 엔진과 같은 Hard real-time 어플리케이션에는 적용하기 어려우며, 바디 제어나 네비게이션과 같은 인포테인먼트 기능을 통합하는 용도로만 사용되고 있다.

2.3 관련연구

Nicolas Navet 외 연구자들은 다중 소프트웨어 멀티코어 차량 ECU의 운영 체제 보호 메커니즘과 스케줄링 방법을 제안함과 동시에 효율적인 보호을 보장하기 위해 코드 간의 분리를 제공할수 있는 방법으로 가상화 기술을 제안하였다.[15] 가상 머신을 제공하여 신뢰할 수 있는 코드와 없는 코드를 분리하여 멀티코어 ECU의 안전성을 향상 시킬 수 방법에 대해 논의하였다. 또한, AUDI와 Infineon의 연구에서도 다양한 안전 요구 사항을 가진 어플리케이션을 가상 머신을 통해 캡슐화하는 멀티코어 ECU 아키텍처의 접근방식을 제안하였다.[16] 가상 머신을 통해 각 운영체제를 완전히 격리된 환경에서 호스팅하고 각 VM 내에서 발생하는 일시적인 오류를 VM을 재설정하여 다른 VM에 전파되지 않도록하는 안전 메커니즘의 구현 방법을 제시하였다.

BMW와 ETAS의 연구자들은 가상화 지원이 없는 MCU 기반 하드웨어 위에 반가상화 방식의 하이퍼바이저 가상화 플랫폼을 구현하여 해당 환경의 인터럽트 관련 명령의 소요 시간을 측정 및 제시하였다.[17] 이를 통해 하드웨어 가상화를 지원하는 MCU 사용이 필수임을 밝혔다. 또한 BMW 사의 다른 연구를 통해 하드웨어 지원을 받은 반가상화 방식에 가상화 환경에서 I/O 가상화의 지연시간을 측정하였다.[12] 해당 실험에서 지연시간은 차량에 적용하기에 충분히 작음을 보였지만, 차량에서 사용하는 MCU가 가상화 지원을 하지 않아 x86 플랫폼에서 가상화 실험을 진행하였으며, 동일한 임베디드 환경에서 반가상화와 전가상화 방식에 대한 비교는 충분치 않은 실정이다.

3. 실험 환경 및 결과

이 장에서는 위에서 소개한 두 가지 가상화 방식, 반가상화, 전가상화 환경의 인터럽트 지연 시간을 비교하기 위해 구성한 하이퍼바이저를 탑재한 임베디드 시스템 실험 환경과 실험 방법에 대해 설명하고, 각 가상화 환경의 인터럽트 지연 시간을 측정하여 실험 결과로 제시한다.

3.1 실험 환경

각 하이퍼바이저와 가상화 방식의 인터럽트 지연시간 분석을 위해 각 하이퍼바이저를 동일한 하드웨어 위에 구현하여 지연시간을 측정하였다. 구현의 편의성을 고려하여 라즈베리 파이 4 모델 B를 호스트 하드웨어로 선택하였다. 해당 호스트 하드웨어 상에 오픈소스 하이퍼바이저인 Xen과 QEMU/KVM을 각각 구현하였으며, 각각의 가상 머신은 리눅스를 운영체제로 선택하였다. Xen을 사용한 가상화 플랫폼은 반가상화 방식을 사용하여 도메인을 가상화 하였으며, QEMU/KVM 환경은 전가상화 방식을 사용하여 가상 머신을 생성하였다. 또한, 각 플랫폼에 동일한 외부 인터럽트 신호를 전송하기 위해 STM32 Nucleo 보드를 사용하여 외부 신호를 송수신할 수 있는 환경을 구성하였다.

3.1.1 Xen 실험 환경

Xen은 서버 환경에서 인기있는 오픈소스 하이퍼바이저이며, 하드웨어 위에서 직접 실행되는 베어메탈 형인 Type 1 하이퍼바이저이며 대표적인 반가상화 하이퍼바이저이다. Xen 환경에서 동작하는 가상 머신은 도메인(Domain)이라고 불린다. 다른 도메인을 관리하고 실제 물리 장치에 접근가능한 특권을 가진 도메인을 Dom0이라고 하며 특권이 없는 나머지 도메인은 DomU라고 한다.

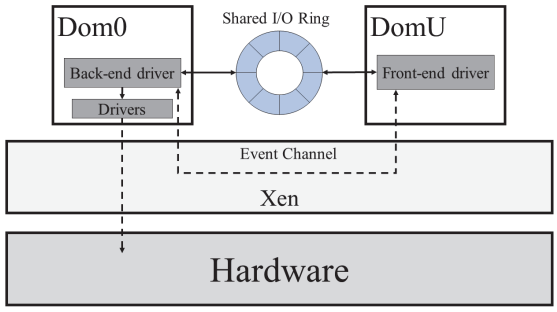


Fig. 3 Xen I/O para-virtualization mechanism

Xen에서 가상화되는 운영체제나 디바이스 드라이버는 하이퍼콜을 이용하여 하이퍼바이저에게 입/출력을 요청할 수 있도록 수정해야한다.

Xen 환경에서 각 도메인의 I/O는 분리 디바이스 드라이버 모델 방식으로 반가상화된다. Fig. 3은 Xen의 I/O 반가상화 방식을 보여준다. Dom0에는 Back-end 드라이버와 실제 디바이스 드라이버, DomU에는 Front-end 드라이버가 존재한다. DomU에서 발생한 I/O 요청을 Front-end 드라이버가 받아, Xen 하이퍼바이저의 I/O ring에 요청을 저장한 다음, 이벤트를 발생시켜 Dom0에게 알린다. 요청을 받은 Dom0의 Back-end 드라이버는 I/O ring의 요청을 읽고 실제 디바이스 드라이버에게 전달한다. 해당 I/O 처리가 완료되면 Back-end 드라이버는 결과를 받아 I/O ring에 저장한 후 이벤트를 발생시킨다. 해당 결과를 Front-end 드라이버가 읽으면 해당 I/O 요청이 완료된다.

본 실험에서 Xen 환경은 Fig. 4와 같이 구성되었다. 라즈베리 파이 4 하드웨어 위에 4.14.3 버전의 Xen 하이퍼

Table 1 Xen paravirtualization experimental environment

	Dom0	DomU-1
Kernel	Linux 5.10.95-v8+	
Memory	1 GB	512 MB
# of vCPUs	2	1

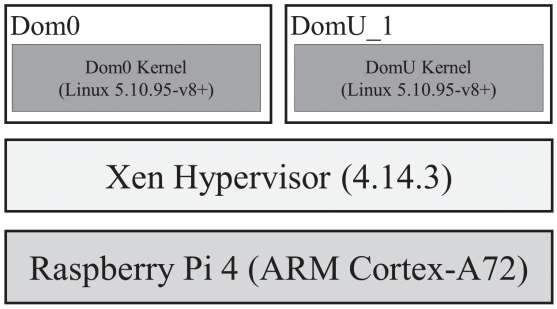


Fig. 4 Xen platform experimental environment

바이저를 포팅하였다. Dom0의 경우, 리눅스 5.10.95-v8+ 버전을 사용하였으며, Xen 환경에서 사용하기 위해 하이퍼콜이 가능하도록 커널을 수정하여 적용하였다. 게스트 도메인 DomU의 커널도 Dom0과 같은 버전의 리눅스를 사용하였으며, 1개의 게스트 도메인만을 생성하였다. Table 1은 Xen 실험 환경의 구성을 간략히 보여준다. 물리 하드웨어를 직접 제어하는 Dom0은 1 GB 메모리와 가상 CPU(vCPU) 코어 두 개를 할당하였다. 게스트 도메인 DomU에는 512 MB 메모리와 가상 CPU 코어를 하나 할당했으며, 두 도메인의 커널은 똑같은 버전의 커널을 구현하였다.

3.1.2 KVM/QEMU 실험 환경

KVM(Kernel-based Virtual Machine)은 리눅스 커널 기반 하이퍼바이저로 Xen과 같이 널리 쓰이는 오픈소스 하이퍼바이저 중 하나이다. KVM은 커널의 서브 모듈로 취급되며, Type 2 하이퍼바이저에 가까운 형태로 주로 전가상화 방식을 사용한다. KVM은 일반적으로 CPU의 하드웨어 가상화 지원을 기반으로 CPU 및 메모리 가상화를 담당하며, QEMU(Quick Emulator)를 기반으로 디바이스, 하드웨어에 대한 에뮬레이팅을 실행한다.

KVM의 I/O 전가상화는 디바이스 드라이버를 가상 머신에 맞게 수정하지 않고 호스트 하드웨어의 디바이스 드라이버를 그대로 사용한다. Fig. 5는 KVM/QEMU의 I/O 전가상화 처리 과정을 보여준다. 게스트 가상머신의 디바이스 드라이버에서 I/O를 요청하면 Exception이 발생하여 호스트 운영체제 내에 KVM 모듈이 이를 처리한다. 모듈은 Exception 발생에 관한 내용을 QEMU로 전달하고, QEMU는 이를 바탕으로 실제 I/O 디바이스 드라이버에게 I/O를 요청한다. 실제 I/O 처리가 완료되면 QEMU는 KVM 모듈에게 게스트 가상머신에게 가상 인터럽트(vIRQ)를 보내라고 명령하고 가상머신이 해당 vIRQ를 받으면 가상 머신이 요청한 I/O 처리가 완료된다.

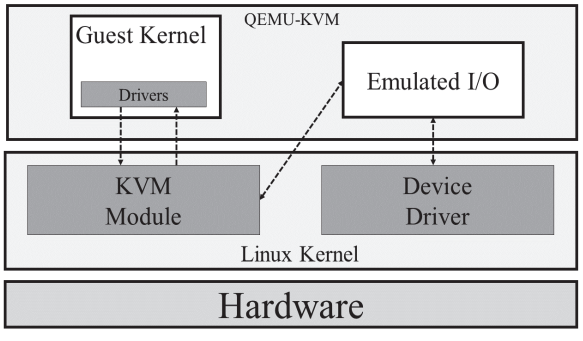


Fig. 5 KVM I/O full virtualization mechanism

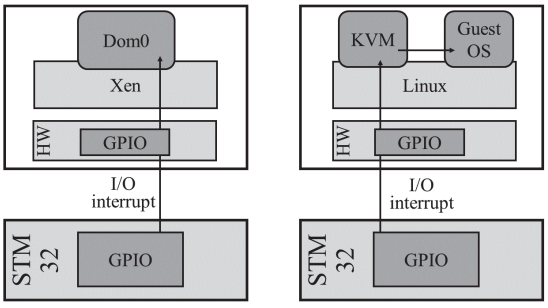
Table 2 QEMU/KVM experimental environment

Host	Hypervisor	QEMU-KVM 5.2.0
VM	# of vCPUs	1
	Memory	1GB
	Kernel	5.10.95-v8+

본 실험에서 KVM/QEMU 실험 환경은 Table 2와 같다. 라즈베리 파이 4 하드웨어 위에 5.10.95-v8+ 버전의 리눅스 커널을 포팅하였으며, QEMU-KVM 5.2.0을 설치하였다. 포팅한 리눅스 커널에서 라즈베리 파이 4 하드웨어의 하드웨어 가속화 기능을 사용할 수 없어 하드웨어 지원 없는 소프트웨어 기반 전가상화 방식을 사용하였다.

3.2 실험 방법

실험의 구성도는 Fig. 6과 같다. 각 가상화 플랫폼들이 동일한 소스에서 인터럽트 신호를 받게 하기 위해 추가로 인터럽트 신호를 출력하는 플랫폼을 STM32 보드를 사용하여 구현하였다. 해당 플랫폼에서 1초마다 I/O 신호를 출력하며, 각 가상화 플랫폼은 GPIO 핀을 통해 신호를 입력받고 이를 수신한 타이밍을 마이크로초 단위까지 측정한다. 해당 작업은 100번 반복 수행하여 각 가상화 플랫폼의 최대 지연시간과 평균 지연시간을 구하였다.



(a) Xen (b) QEMU/KVM

Fig. 6 Experimental diagram for latency measurement

3.3 실험 결과

Table 3은 네이티브 환경과 각 가상화 플랫폼 내에서 측정된 평균 지연 시간을 보여준다. 지연 시간 측정을 위해 인터럽트 측정 시간 출력 및 기록에 따른 추가적인 지연 시간은 동일하다고 가정한다.

해당 실험에서 가상화가 적용되지 않은 네이티브 실험 환경의 경우, 평균 182 μs의 지연 시간이 측정되었다. 측정 시간 기록 및 기타 전파 지연 등을 고려하였을 때 해당 호스트의 가상화하지 않은 네이티브 환경에서는 유의미한 지연 시간을 보이지 않는다.

Table 3 Interrupt latency (ms)

	Avg	Min	Max
Native	0.182	0.175	0.27
Xen	1.25	1.19	2.98
KVM/QEMU	52	49	1054

Xen 하이퍼바이저 기반의 반가상화 실험 환경의 경우, 평균 지연 시간은 약 1.25 ms로 측정되었다. 이는 네이티브 실험 환경과 비교하였을 때 약 7배 높은 지연 시간이다.

QEMU/KVM 기반의 전가상화 실험 환경에서는 평균 지연 시간이 약 52 ms로 측정되었다. 이는 가상화하지 않은 네이티브 실험 환경에 비해 수백배 높은 지연 시간이다. 하드웨어의 가상화 지원이 없는 전가상화 방식의 오버헤드로 인한 성능 저하가 매우 큰 것으로 나타났다.

3.4 실험 결과 고찰

Xen 가상화 환경의 경우, Xen 하이퍼바이저에서 물리 CPU를 실행할 VCPU를 스케줄링하는데, Xen 4.14의 기본 스케줄러인 Credit2 scheduler는 각 VM들 간에 선점으로 인해 발생하는 컨텍스트 스위칭으로 시간을 소비하는 것을 줄이기 위해 컨텍스트 스위칭 속도 제한 기능을 제공한다. 이를 통해 VM 실행을 시작하였을 때 우선 순위가 더 높은 VM이 깨어나더라도, 컨텍스트 스위칭 속도 제한으로 정해진 시간 만큼은 계속 실행될 수 있다. 이러한 속도 제한 기능의 기본값은 1 ms이다. 해당 실험 환경에서 Xen의 지연 시간은 Dom0에서 측정하였으며, Dom0은 다른 게스트인 DomU와는 달리 특별한 권한은 가진 도메인으로 I/O 등 권한이 필요한 작업 시 하이퍼콜과 관련된 함수를 실행하지 않아 하이퍼콜에 인한 지연은 발생하지 않지만, 이러한 스케줄러의 특성으로 지연이 측정된 것으로 보인다.

QEMU/KVM 환경의 경우, 하드웨어적 한계로 인해 하드웨어 지원 가상화가 아닌 소프트웨어적인 전가상화 방식으로 가상화되었다. 소프트웨어적으로 가상화된 경우, Binary translation으로 가상머신의 명령어를 변환하는데, 하드웨어적 한계로 KVM 가속을 활용할 수 없어 지연 시간이 대폭 상승한 것으로 보인다. 이로 인해 반가상화 환경에 비해 많은 오버헤드가 발생하여 반가상화에 비해 높은 평균 지연 시간을 보였다.

하드웨어 지원 가상화는 x86 환경에서 Intel의 VT, AMD의 AMD-V로 대표되는 기술로 CPU의 하드웨어 지원을 통해 관련 명령을 하드웨어에서 직접 처리하여 성능 오버헤드를 줄인다. 하지만 자동차 환경에서 상용되는 CPU의 경우 이러한 기능 지원이 부족하여 소프트웨어적으로 가상화해야 하기 때문에 성능 측면에서 큰 손실을 보게된다. 또한, 전가상화 방식에서는 하드웨어를 에뮬레이션하여 하드웨어를 소프트웨어로 동작시켜야 하기 때문에 오버헤드가 커져 반가상화 방식에 비해 성능 저하가 더 커지게 된다.

현재 자동차 환경에서 사용되는 CPU의 가상화 지원 문제와 해당 실험에서 보인 반가상화와 전가상화 기술의 성능 차이를 고려하였을 때, 반가상화 기술이 현재 자동차 환경에 적용되기에 더 바람직하다.

4. 결론 및 향후 연구 계획

본 논문에서는 차량 시스템 가상화를 위한 하이퍼바이저 지연 인터럽트 지연 시간을 측정하기 위해 대표적인 오픈소스 하이퍼바이저인 Xen, KVM 기반의 임베디드 가상화 환경을 구축하고, 가상화 방식에 따른 인터럽트 지연 시간을 측정하였다.

전가상화 방식은 실험 결과 평균 50 ms의 지연 시간을 보이며 하드웨어 가상화 지원 없이는 차량 임베디드 시스템에서 사용하기에는 바람직하지 않음을 보였다. 현재 차량에서 상용되는 CPU에 하드웨어 가상화 지원이 도입되어야만 자동차 환경에 적용 가능할 것으로 보인다. Xen 기반의 반가상화의 경우 평균 약 1 ms 정도의 지연 시간을 보여 전가상화 방식에 비해 낮은 지연 시간에 인터럽트를 처리할 수 있음을 보였다.

하지만 차량의 실시간 시스템에서 사용하기 위해서 요구되는 실시간 성능에는 미치지 못해 실시간 성능에 대한 추가적인 연구가 필요하다. 따라서 향후에는 Xen 하이퍼바이저의 실시간 인터럽트 처리를 위해 지연 구간 분석과 이를 위한 해결방안에 대한 연구할 계획이다.

후 기

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2020R1G1A1015210).

References

1) M. Broy, “Challenges in Automotive Software Engineering,” Proceedings of the 28th International Conference on Software Engineering, pp.33-42, 2006.
2) S. Fürst, “Challenges in the Design of Automotive Software,” 2010 Design, Automation & Test in Europe Conference & Exhibition(DATE 2010),

pp.256-258, 2010.
3) I. Park, W. Lee and M. SunWoo, “Application Software Modeling and Integration Methodology Using AUTOSAR-ready Light Software Architecture,” Transactions of KSAE, Vol.20, No.6, pp.117-125, 2012.
4) K. Lee, I. Park, M. Sunwoo and W. Lee, “AUTOSAR-ready Light Software Architecture for Automotive Embedded Control Systems,” Transactions of KSAE, Vol.21, No.1, pp.68-77, 2013.
5) A. Monot, N. Navet, B. Bavoux and F. Simonot-Lion, “Multisource Software on Multicore Automotive ECUs—Combining Runnable Sequencing with Task Scheduling,” IEEE Transactions on Industrial Electronics, Vol.59, No.10, pp.3934-3942, 2012.
6) R. Dominik and M. Kucera, “Domain Controlled Architecture,” 3rd International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), 2013.
7) R. Dominik, D. Kaule and M. Kucera, “Achieving a Scalable E/E-architecture Using Autosar and Virtualization,” SAE International Journal of Passenger Cars-Electronic and Electrical Systems, Vol.6, No.2, pp.489-497, 2013.
8) M. Di Natale and A. L. Sangiovanni-Vincentelli, “Moving from Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools,” Proceedings of the IEEE, Vol.98, No.4, pp.603-620, 2010.
9) Renesas Electronics Corporation, Zone-ECU Virtualization Solution Platform, Renesas White Paper, 2022.
10) ISO 26262, Road Vehicles – Functional Safety – Part 1-10, International Organization for Standardization, 2011.
11) J. Y. Moon, D. Y. Kim, J. H. Kim and J. W. Jeon, “The Migration of Engine ECU Software from Single-core to Multi-core,” IEEE Access, Vol.9, pp.55742-55753, 2021.
12) C. Herber, D. Reinhardt, A. Richter and A. Herkersdorf, “HW/SW Trade-offs in I/O Virtualization for Controller Area Network,” 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp.1-6, 2015.
13) M. Strobl, M. Kucera, A. Foeldi, T. Waas, N. Balbierer and C. Hilbert, “Towards Automotive Virtualization,” International Conference on Applied Electronics, pp.1-6, 2013.
14) A. Masrur, S. Drossler, T. Pfeuffer and S. Chakraborty, “VM-Based Real-Time Services for Automotive Control Applications,” IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications, pp.218-223, 2010.
15) N. Navet, A. Monot, B. Bavoux and F. Simonot-Lion, “Multi-source and Multicore Automotive ECUs - OS Protection Mechanisms and Scheduling,” IEEE International Symposium on Industrial Electronics, pp.3734-3741, 2010.
16) A. Kohn, M. Käßmeyer, R. Schneider, A. Roger, C. Stellwag and A. Herkersdorf, “Fail-operational in Safety-related Automotive Multi-core Systems,” 10th IEEE International Symposium on Industrial Embedded Systems(SIES), pp.1-4, 2015
17) D. Reinhardt and G. Morgan, “An Embedded Hypervisor for Safety-relevant Automotive E/E-systems,” Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems(SIES 2014), pp.189-198, 2014.