멀티프로세서 윈도우즈 상에서 실시간성 지원

Real-Time Support on Multi-Processor for Windows

송창인, 이승훈, 주민규, 이철훈
충남대학교 컴퓨터공학과

Chang-In Song(jjami1234@cnu.ac.kr), Seung-Hoon Lee(sh2011@cnu.ac.kr),
Min-Gyu Ju(mingyuman@cnu.ac.kr), Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

최근 시스템 개발 환경이 싱글프로세서 기반의 환경에서 멀티프로세서 기반의 환경으로 변화됨에 따라 개발된 시스템 프로그램들이 멀티프로세서 기반과 싱글프로세서 기반에서 호환이 가능하도록 설계 및 구현의 중요성이 부각되고 있다. 특히, 실시간성이 중요시 되는 내장형 소프트웨어 분야 및 측정 장비와 같이 정밀성을 요구하는 분야의 경우 멀티프로세서 기반의 실시간성을 제공하기 위한 연구가 필요하다. 운영체제에 종속적인 실시간성은 운영체제에 따라 지원여부를 결정하는데, 범용 운영체제인 윈도우즈의 경우 실시간성을 지원하기 위해 RTX나 INtime과 같은 고가의 서드파티를 사용해야 한다. 그러나 서드파티를 사용함으로써 제품 개발 시 고가의 구입비용 및 유지보수 비용으로 인한 개발 비용의 증가를 초래하는 문제점이 발생한다. 이러한 문제점을 해결하기 위해서, 본 논문에서는 멀티프로세서 기반의 윈도우즈에 실시간성을 제공하기 위해 x86 아키텍처의 Local APIC를 이용하여 윈도우즈에 실시간성을 제공하는 RTiK-MP(Real-Time implant Kernel-Multi Processor)를 설계 및 구현하였고, RTiK-MP의 성능 검증 및 평가를 위하여 휴대용 점검장비를 이용한 성능 측정 결과를 분석한다.

■ 중심어 : 윈도우즈 | 실시간성 | 이식커널 | 점검장비 |

Abstract

As the system development environment moves from single core to multi core-based platforms, it becomes more important to maintain compatibility between single core-based implementations and multi core-based implementations. Moreover, it is very important to support real-time on multi core platforms especially in cases of embedded software or test equipments which need real-time as well as correctness. Since Windows operating system does not support real-time in itself, it has been supporting real-time using expensive third-party solutions such as RTX or INtime. So as to reduce this kind of development expenses, in this paper, we propose RTiK-MP(Real-Time implant Kernel-Multi Processor) which supports real-time on Windows using the Local APIC of x86 architectures, and evaluate the performance of the proposed RTiK-MP after deploying it on portable missile test equipments.

■ keyword : Windows | Real-Time | Implant Kernel | Test Equipment |

* 본 연구는 2010년도 충남대학교 학술연구비에 의해 지원되었음

접수번호 : #120227-002

접수일자 : 2012년 02월 27일

심사완료일 : 2012년 06월 12일

교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

I. 서론

최근 일상생활에서 사용되고 있는 컴퓨터 및 휴대용 전자기기들의 CPU 클럭 스피드가 열, 전력소모, 전력 누출 등과 같은 여러 가지 물리적인 요인에 의해 한계에 도달함에 따라 CPU의 사양이 싱글프로세서에서 멀티프로세서로 변화되고 있는 추세이다. 마찬가지로 임베디드 시스템의 CPU도 싱글프로세서 기반에서 멀티프로세서 기반으로 변화됨에 따라 무기체계의 성능 검증을 위해 필요한 점검장비들 역시 멀티프로세서로 변화되고 있다. 이러한 점검장비들은 개발의 편의성을 제공하기 위해 운영체제로 윈도우즈를 사용하는데, 유도 무기체계에서 사용되고 있는 휴대용 점검장비의 경우, 실시간으로 데이터를 획득하고 평가하기 위해 실시간성을 지원해야 한다. 그러나 윈도우즈 운영체제는 실시간성을 지원하지 않기 때문에 휴대용 점검장비가 실시간성을 보장하기 위해서는 고가의 RTX와 INtime과 같은 서드파티를 구매하여 실시간성을 보장하고 있다. 이에 따라 새로운 무기 개발 시 서드파티의 사용으로 인한 개발 비용이 증가하는 문제점이 발생한다. 또한 윈도우즈의 버전이 윈도우즈 XP에서 윈도우즈 7으로 변화함에 따라 멀티프로세서 기반의 윈도우즈 7 환경에서 실시간성을 제공하기 위한 연구가 필요하다.

본 논문에서는 멀티프로세서상의 윈도우즈에서 실시간성을 지원하기 위해 사용자가 지정한 CPU의 Local APIC(Advanced Programmable Interrupt Controller)를 이용하여 주기적인 타이머 인터럽트를 발생시키는 RTiK-MP를 설계 및 구현하였다. 또한 구현한 RTiK-MP를 실제 휴대용 점검장비에 이식하여 얻은 실험결과를 통해 본 논문에서 제안한 RTiK-MP가 커널 레벨에서 최소 1ms의 정확한 응답속도로 동작하는 것을 증명함으로써 멀티프로세서 기반의 윈도우즈에서 실시간성을 제공하는 것을 확인한다. 이를 통해 멀티프로세서 기반의 윈도우즈 환경을 가지는 휴대용 점검장비에 실시간성을 제공할 수 있음을 증명한다.

본 논문은 2장에서 관련연구로 서드파티와 Local APIC, 싱글프로세서 기반의 윈도우즈에 실시간성을 제공하는 RTiK에 대해 기술하고, 3장에서 멀티프로세서

상의 윈도우즈에 실시간성을 제공하기 위한 RTiK-MP의 설계 및 구현에 대해 설명한다. 4장에서는 실험환경 및 결과를 기술하며, 마지막 5장에서 결론 및 향후 연구 과제에 대해 기술한다.

II. 관련연구

현재 윈도우즈에서 실시간성을 제공하는 서드파티 운영체제인 RTX와 RTiK에 대해 기술하며, 윈도우즈의 버전에 따른 구현방식의 차이점에 관한 내용을 기술한다.

1. 멀티프로세서상의 RTX 구조

IntervalZero의 RTX(Real-Time eXtension)는 윈도우즈 XP 및 윈도우즈 7에 실시간 제어가 가능한 실시간 운영체제의 기능을 부가해주는 확장 소프트웨어로써, 대부분의 개발자들이 익숙하고 편리한 윈도우즈 환경에서 RTX를 이용하여 기존 윈도우즈의 장점을 최대한 사용할 수 있도록 제공해 준다. 즉, RTX는 순수 실시간 운영체제가 아니라 윈도우즈의 대중성 및 풍부한 GUI Library의 장점을 최대한 이용하여 실시간성 제공에 관한 제약사항들을 보완해 주는 소프트웨어이다. 이러한 RTX는 멀티프로세서상의 윈도우즈에 실시간성을 제공하기 위해 Dedicate Mode를 사용하는데, Dedicate Mode란 하나의 CPU를 RTX가 독점하여 실시간성을 제공하는 것을 의미한다. 이때 독점된 CPU에서는 윈도우즈의 일반 응용프로그램들은 동작하지 않으며, 오직 RTX의 실시간성을 제공하는 프로그램만 동작하게 된다. 마찬가지로 RTX의 실시간성을 제공하는 프로그램은 윈도우즈가 동작하는 CPU에서 동작하지 않는다. 즉, [그림 1]에서 보듯이 두 개의 CPU가 존재한다면 RTX를 설치 시 윈도우즈는 하나의 CPU만을 인식하게 되고, 다른 하나의 CPU는 RTX가 점유하게 된다. 이는 사용자의 응용 프로그램이 하나의 프로세스만을 사용하게 되기 때문에 다수의 응용 프로그램이 동작 시 전체적인 시스템의 성능이 떨어지게 된다. 이러한 Dedicate Mode를 통해 실시간성을 제공하는 RTX는

현재 국방, 항공, 계측기, 시뮬레이션, 의료기기 등의 많은 산업분야에서 사용되고 있다[1].

그림 1. 멀티프로세서상의 Dedicated Mode를 사용한 RTX구조

2. RTiK(Real-Time implant Kernel)

2.1 RTiK의 동작방식

윈도우즈의 디바이스 드라이버 형태로 구현된 RTiK은 커널 레벨에서 동작하여 하드웨어의 자원 접근 및 윈도우즈의 커널에 접근을 할 수가 있다. 실시간성을 지원하기 위해 HAL(Hardware Abstraction Layer)을 통한 x86 기반의 하드웨어 플랫폼의 Local APIC의 제어를 통하여 실시간성을 제공하였다. 이는 윈도우즈의 Local APIC 타이머와 관련된 레지스터들의 제어를 통해 윈도우즈와는 다른 독립적인 타이머 인터럽트를 발생시켜 주기적인 동작을 보장한다. [그림 2]는 LVT(Local Vector Table)의 타이머 관련 레지스터에 관한 그림이다. LVT의 타이머 레지스터는 타이머의 상태정보를 유지하는 레지스터로써 타이머 인터럽트의 모드, 마스크 상태, 인터럽트 전달상태 및 인터럽트 벡터 번호를 가진다. Initial Count 레지스터의 경우 설정한 카운트가 감소되어 0에 도달하면 타이머 인터럽트를 발생시키게 된다. Local APIC의 타이머 인터럽트가 발생하면 타이머 레지스터의 벡터를 통해 해당하는 IDT(Interrupt Descriptor Table)의 벡터번호로 분기하게 된다. IDT에 저장되어 있는 인터럽트의 핸들러 주소를 참조하여 주기적으로 발생하는 타이머 인터럽트 핸들러를 수행한다. 또한 실시간 쓰레드는 지연처리 호출(DPC: Deferred Procedure Call)을 통하여 RTiK의 타이머 인터럽트 서비스 루틴이 실행된 직후에 CPU 권한을 받아 수행하게 된다. 이러한 지연처리 호출 메커니

즘을 사용함으로써 실시간 쓰레드가 주기적으로 수행됨을 보장함과 동시에, 윈도우즈의 쓰레드보다 먼저 CPU 권한을 받음으로써 실시간성을 보장할 수 있도록 구현되었다. 그러나 최근 점점장비의 하드웨어 사양이 멀티프로세서로 변화됨에 따라 기존에 개발된 RTiK의 경우 멀티프로세서 환경에서 동작하지 않는 문제점을 지니고 있다. 따라서 이러한 문제점을 개선하여 다수의 프로세서 환경에서도 동작이 가능하도록 구현될 필요가 있다[2-5].

그림 2. Local APIC 타이머 관련 레지스터

2.2 이벤트 기반의 RTiK

RTiK은 윈도우즈 사용자 레벨에서 동작하는 쓰레드들에게 실시간성을 지원하기 위한 방법을 제공한다. [그림 3]에서 보듯이 사용자가 RTiK에서 제공하는 API를 통해 실시간 쓰레드를 생성하고, 수행코드를 작성하게 되면 생성된 실시간 쓰레드는 커널 레벨로부터 전달되는 Signal을 기다리게 된다. 커널 레벨로부터 Signal을 전달 받게 되면 실시간 쓰레드에 정의된 작업을 수행함으로써 주기적인 동작을 보장한다[6].

그림 3. 이벤트 기반 RTiK의 동작방식

3. WDF(Windows Driver Foundation)의 구조

최근 윈도우즈 운영체제의 최신버전인 윈도우즈 7의

사용이 증가함에 따라 윈도우즈 XP버전과 윈도우즈 7 버전 모두 호환이 가능한 디바이스 드라이버의 개발이 필요하다. 기존에 개발된 RTiK의 경우 윈도우즈의 WDM(Windows Driver Model) 기반의 디바이스 드라이버 형태로 구현이 되어 윈도우즈 XP에서 실시간성을 보장한다. 그러나 WDM 기반의 RTiK은 윈도우즈 7 및 멀티프로세서 환경에서 동작하지 않기 때문에 윈도우즈 7 및 멀티프로세서 환경에서 정상 동작이 가능하게 하기 위해서는 WDF 기반으로 구현해야한다. [그림 4]는 WDM과 WDF의 기본구조를 나타낸 모습이다. 그림에서 보듯이 WDM의 경우 디바이스 드라이버가 커널에 인식된 후 장치를 제어하기 위한 Function 드라이버와 Function 드라이버의 기능 보완 역할을 해주는 Upper Filter와 Lower Filter가 디바이스 스택에 존재한다. 또한 버스 드라이버를 통해 직접적으로 물리적 계층에 접근하여 하드웨어 자원을 제어한다. 이에 반해 WDF의 구조는 커널용 디바이스 드라이버를 작성 시 WDM의 구조에서 가지는 Upper Filter와 Lower Filter, 버스 드라이버가 디바이스 스택에서 생략된 구조를 가지기 때문에 싱글 프로세서 및 멀티프로세서 환경 모두 호환이 가능하다. 따라서 WDF의 구조를 이용하여 멀티프로세서 환경에서 RTiK-MP가 동작 가능하도록 설계 및 구현이 되어야 한다.

그림 4. WDF와 WDM의 구조

III. 실시간성 지원을 위한 설계 및 구현

RTiK-MP는 x86 기반의 멀티프로세서 윈도우즈에

실시간성을 제공하기 위해 디바이스 드라이버 형태로 이식되며 윈도우즈와는 별개의 HAL을 가짐으로써 각각의 프로세서가 가지고 있는 Local APIC 접근을 통해 실시간성을 지원한다. [그림 5]는 RTiK-MP의 전체적인 동작과정을 나타내고 있다. BSP(Boot Strap Processor)는 윈도우즈 동작 시 부팅에 관련된 일을 하는 프로세서로써 운영체제의 전반적인 작업을 담당하고, AP(Application Processor)는 응용프로그램들이 동작하는 프로세서이다.

그림 5. RTiK-MP의 동작방식

그림에서 보듯이 CPU 두 개를 가지는 시스템의 경우, 각각의 프로세서는 고유의 Local APIC를 가지며, RTiK-MP의 경우 AP프로세서의 Local APIC를 이용하여 실시간성을 제공한다. AP에서 Local APIC 타이머가 발생하게 되면 AP의 IDT(Interrupt Descriptor Table)을 이용하여 타이머 핸들러를 수행하게 되는데 수행되는 핸들러는 각각의 프로세서가 가지고 있는 DPC Queue에 DPC를 등록하게 된다. 인터럽트의 수행이 끝나고 IRQL(Interrupt Request Level) 값이 DPC Dispatch 레벨이 되면 등록된 DPC들을 DPC 루틴에서 처리하게 된다. RTiK-MP는 실시간성을 제공하기 위해 사용자가 하나의 프로세서를 선택하여 해당 프로세서의 하드웨어 자원을 이용할 수 있도록 설계 및 구현하였다. 이를 통해 RTX의 경우 실시간성을 제공하기 위해 사용되는 CPU를 사용자 및 다른 응용프로그램이 사용할 수 없는 문제점이 발생하지만, RTiK-MP의 경우 운영체제가 모든 CPU를 인식하여 사용할 수 있도록 구현하였다.

APIC : Advanced Programmable Interrupt Controller

그림 6. AP의 Local APIC 타이머

1. AP의 Local APIC

Local APIC는 PIC(Programmable Interrupt Controller)의 확장으로 x86 아키텍처에서 제공하는 인터럽트 컨트롤러이다. APIC는 하드웨어 인터럽트를 인터럽트 핸들러의 주소를 가지고 있는 IDT로 전달해주는 역할을 한다. 싱글프로세서와 멀티프로세서는 CPU에서 Local APIC에 관한 처리가 다르다. 멀티프로세서 기반에서의 Local APIC는 각각의 프로세서마다 존재하게 되었고, 각각의 Local APIC에서 인터럽트를 발생시킬 수 있게 되었다. 이에 따라 인터럽트가 발생 시 접근되는 IDT도 각 프로세서마다 존재하게 된다. 따라서 두 개의 프로세서 중 하나의 프로세서를 선택하여 해당 프로세서의 Local APIC를 이용해 타이머 인터럽트가 발생하도록 설계 및 구현되어야 한다. RTiK-MP의 경우 프로세서들 중 AP를 선택하여 AP의 Local APIC를 이용해 타이머 인터럽트가 발생하도록 구현하였다. [그림 6]은 프로세서가 두 개일 경우 AP의 Local APIC의 구조를 나타내고 있다. [그림 6]에서 보듯이 RTiK-MP는 프로세서들 중 BSP가 아닌 AP의 Local APIC의 타이머 관련 레지스터들을 이용하여 윈도우즈와는 독립적인 타이머 인터럽트를 발생시켜 주기적인 동작을 보장한다[8-12].

2. 멀티프로세서에서의 CPU Affinity설정

윈도우즈의 CPU Affinity란 인터럽트 발생 시 사용자 및 프로그램에서 정해진 Affinity의 값에 따라 인터럽트가 처리될 CPU를 지정하는 메커니즘이다. CPU의

Affinity 값을 설정하기 위해서는 디바이스 드라이버의 설치 정보를 가지고 있는 INF 파일에서 해당 값을 설정해야 한다. RTiK-MP의 경우에도 디바이스 드라이버가 커널에 인식되기 전 INF 파일에서 사용할 CPU를 설정함으로써 윈도우즈와는 독립적으로 발생하는 인터럽트를 지정한 CPU에서 처리하도록 구현하였다. 또한 RTiK-MP는 CPU의 Affinity 값을 BSP의 CPU를 제외한 AP의 CPU를 선택할 수 있게 함으로써 AP의 Local APIC를 사용할 수 있도록 구현하였다. [그림 7]은 RTiK-MP의 INF 파일에서 CPU Affinity를 설정한 모습이다. [그림 7]에서 보듯이 RTiK-MP가 INF 파일을 이용하여 CPU의 Affinity 설정 및 디바이스 드라이버의 우선순위와 정책의 설정을 통해 지정한 CPU에서 동작하도록 구현하였다.

그림 7. RTiK-MP의 INF 파일

[그림 8]은 INF파일에서 CPU의 Affinity를 설정 후 Local APIC 타이머 인터럽트를 발생시켜 인터럽트가 처리되는 CPU의 번호를 윈도우즈에서 제공하는 API를 사용하여 WinDBG를 통해 출력한 화면이다. INF파

일의 CPU Affinity 값은 0x1부터 시작되며, CPU의 수가 증가할수록 1씩 증가하고, 윈도우즈에서 제공하는 API를 사용하여 출력한 CPU의 번호는 0번부터 시작하여 CPU의 수가 증가할수록 1씩 증가한다. 따라서 INF 파일의 CPU Affinity 설정 값에 따라 해당 CPU에서 인터럽트가 동작하는 것을 확인할 수 있다[13][14].

그림 8. 지정된 CPU에서 동작하는 타이머 인터럽트

3. RTiK-MP의 인터럽트 자원할당

그림 9. RTiK-MP에서 자원 할당을 위한 알고리즘

[그림 9]는 RTiK-MP에서 인터럽트 자원을 할당 받기 위해 사용한 알고리즘이다. 그림에서 보듯이 Driver Entry에서 디바이스 드라이버를 구성하는 드라이버 오브젝트를 생성하고 생성된 드라이버 오브젝트를 RTiK_EvtAddDevice 함수로 전달함으로써 드라이버에 대한 디바이스 스택을 생성 및 설정하게 된다. 이를 통해 디바이스 스택이 포함해야 할 자원을 할당받기 위해 RTiK_PrepareHardware 함수를 호출하게 된다. 이때 INF 파일에서 설정한 IRQ Number에 따른 인터럽트 백터를 할당받게 되며, 할당받은 인터럽트 백터를 통해 RTiK_EvtAddDevice에서 자동적으로 Local APIC Timer 인터럽트와 인터럽트 백터를 연결시키게 된다. 이를 통해 Local APIC Timer 인터럽트가 발생하게 되면 해당 인터럽트 백터에 존재하는 핸들러 주소로 분기함으로써, 인터럽트에 대한 처리를 수행하게 된다.

4. 타이머 인터럽트의 등록

RTiK-MP는 윈도우즈에 실시간성을 제공하기 위하여 Local APIC를 이용해 윈도우즈와는 독립적인 타이머 인터럽트를 발생시킴으로써 쓰레드의 주기적인 수행을 보장하도록 구현하였다. 즉, Local APIC의 타이머 인터럽트가 발생하면 타이머 레지스터의 백터에 해당하는 IDT의 백터번호로 분기하게 된다. IDT의 백터에 저장되어 있는 인터럽트의 핸들러 주소를 참조하여 주기적으로 발생하는 타이머 인터럽트를 수행한다. 이러한 인터럽트 처리 과정을 위해 윈도우즈의 IDT에 인터럽트 오브젝트를 등록시켜야한다. 멀티프로세서 상에서 IDT는 프로세서마다 존재하므로 지정한 프로세서의 IDT에 인터럽트 오브젝트를 등록시킴으로써 인터럽트의 처리가 해당 프로세서에서만 처리되도록 설계 및 구현하였다. [그림 10]은 타이머 인터럽트가 발생한 후 타이머 인터럽트를 처리하는 과정을 나타낸 그림이다. 그림에서 보듯이 타이머 인터럽트가 발생하기 전에 Local APIC의 타이머 관련 레지스터의 설정과 IDT에 등록되어 있는 인터럽트 백터를 얻어오게 된다. 이를 통해 Local APIC 타이머 인터럽트가 발생하게 되면 IDT에 등록된 백터에 의해 ISR로 분기하게 된다. ISR에서는 인터럽트 지연시간의 최소화를 위해 DPC를 등록함으로써 주기적인 동작을 보장하도록 구현하였다.

그림 10. 인터럽트 처리과정

[그림 11]은 RTiK-MP를 윈도우즈에 이식 후 WinDbg를 통해 LVT의 타이머 레지스터 값을 출력한 화면으로써 선택한 프로세서에 인터럽트 오브젝트가 정상적으로 등록된 것을 보이고 있다. 그림에서 보듯이 BSP의 LVT의 타이머 레지스터 값은 초기 값인 0x300fd 값을 가지는 것을 확인할 수 있다. 이 값을 통해 BSP의 Local APIC 타이머 인터럽트는 비활성화 상

타이머, IDT의 벡터번호가 0xfd인 것을 확인할 수 있다. 반면에 RTiK-MP가 이식된 AP의 경우 LVT의 타이머 레지스터의 값은 벡터 값이 0xb2로 변경되었고, 타이머 인터럽트가 활성화된 것을 확인할 수 있다. 이를 통해 AP의 Local APIC 타이머 인터럽트가 활성화 되면 윈도우즈와는 별개의 인터럽트가 발생하게 되고, IDT의 0xb2번지로 분기하여 등록된 인터럽트 핸들러를 통해 인터럽트를 처리하게 된다[15][16].

그림 11. IDT에 인터럽트 오브젝트 등록

IV. 실험 환경 및 결과

1. 실험 환경 및 실험 방법

멀티프로세서 기반의 윈도우즈 7에 RTiK-MP를 구현하기 위한 실험 환경은 [표 1]과 같이 구성하였다. Target PC에서 RTiK-MP를 이식하여 동작을 시켰으며, Host PC에서 Target PC를 모니터링 하였다.

표 1. 실험 환경

	Host	Target
CPU	Pentium(R) 4 CPU 3.00GHz	Intel Core i7 640LM @ 2.13GHz
OS	Windows XP SP3	Windows 7 Ultimate K
DDK	WDK 7600.16385.1	WDK 7600.16385.1
Development Tool	MS Visual Studio 2005	WDK Checked Builder

[그림 12]은 RTiK-MP의 주기 측정을 위한 실험 환경을 나타낸 그림이다. 그림에서 보듯이 휴대용 점검장비를 Target PC로 하였고, RTiK-MP의 동작 및 주기를 확인하기 위해 휴대용 점검장비의 Acromag APC 8625 PCI Carrier 보드와 IP470 Digital OutPut Module

를 이용하여, 점검장비의 포트에 신호를 출력하였다. IP470 레지스터의 포트 0에 해당하는 6개의 채널을 이용하여, 맨 처음 Local APIC 타이머 인터럽트가 발생할 시 0x01로 설정하고, 다음의 인터럽트가 발생하면 0x00로 설정하여 동작하는 모습을 볼 수 있도록 실험하였다. 실험 방법은 휴대용 점검장비에서 커널 레벨의 1ms의 주기를 가지는 실시간 쓰레드와 여러 개의 주기를 가지는 실시간 쓰레드를 생성하였고, 생성된 주기 쓰레드의 정확한 주기 측정을 위해 점검장비의 포트 신호를 오실로스코프를 이용하여 확인하였다. 또한 윈도우즈에서 동작하는 다른 프로세스와 동시에 실행시켰을 때의 주기 변화를 확인하기 위해 무한으로 While문만을 수행하는 프로세스를 생성하여 주기 측정을 하였다.

그림 12. 실험 환경

2. 실험 결과

[그림 13] 및 [그림 14]는 RTiK-MP의 주기를 최소 주기인 1ms로 설정하고, 워크로드가 없을 경우와 10개의 워크로드와 RTiK-MP의 실시간 쓰레드를 동시에 수행 했을 때의 주기를 측정한 화면이다. 그림에서 보듯이 신호를 출력하는 구간과 신호를 출력하지 않는 구간이 2ms동안 유지되는 것을 확인할 수 있다. 이는 출력하는 구간과 출력하지 않는 구간이 1ms동안 동작하는 것을 의미하고, 또한 각 구간에서 오실로스코프의 파형이 흔들리지 않고 출력이 되는 것은 실시간 쓰레드가 지터 없이 주기적인 동작한다는 것을 의미한다.

그림 13. RTiK-MP 주기 1ms 주기측정

그림 14. RTiK-MP 주기 1ms & Workload 10개 주기 측정

표 2. RTiK-MP의 주기측정 결과

주기	While 개수	평균(ms)	최소(ms)	최대(ms)
1ms	0	1.00003	0.98765	1.01250
	10	1.00003	0.97565	1.03035

[표 2]는 [그림 13]과 [그림 14]의 측정값을 나타낸 표이다. [표 2]에서 보듯이 RTiK-MP의 실시간 쓰레드는 워크로드가 없을 경우 약 2%내외의 오차를 가지고 워크로드가 있을 경우 약 3%정도의 오차를 가지며 주기적으로 동작하는 것을 확인할 수 있다. 이를 통해 RTiK-MP의 실시간 쓰레드가 윈도우즈의 사용자 레벨에서 동작하는 다수의 병렬 프로세스에 영향을 받지 않고 설정된 주기를 벗어나지 않는 것을 알 수 있다. 이는 RTiK-MP의 실시간 쓰레드가 윈도우즈의 스케줄링에 영향을 받지 않고, 항상 설정된 주기를 지키며 동작함으로써 멀티프로세서 기반의 윈도우즈에 실시간성을 제공하는 것을 의미한다.

그림 15. 큐타이머 15ms 주기측정

[그림 15]와 [그림 16]은 동일한 환경에서 RTiK-MP를 이식시키지 않고 윈도우즈에서 제공하는 큐 타이머

의 성능을 측정 결과 화면이다. 큐 타이머는 최소 1ms의 주기를 지원하지만 실험에 사용된 점검장비 및 실험 환경에서는 1ms의 주기를 지원할 수 없고, 최소 15ms의 주기를 지원하기 때문에 설정 가능한 최소주기인 15ms로 설정하여 주기측정을 실험하였다.

그림 16. 큐타이머 15ms & Workload 10개 주기측정

[그림 16]에서 볼 수 있듯이 다른 실험 결과와 같이 일정한 간격을 두고 출력되는 것이 아니라 워크로드에 의한 지터로 오실로스코프에 잔상이 남는 것을 확인할 수 있다. 또한 [표 3]에서 볼 수 있듯이 워크로드를 적용하지 않았을 경우 약 3%정도의 오차가 생기는 것을 확인할 수 있고, 워크로드를 적용 시 주기를 크게 벗어나는 것을 확인할 수 있다.

표 3. 큐 타이머의 주기측정 결과

주기	While 개수	평균(ms)	최소(ms)	최대(ms)
15ms	0	15.6245	15.5650	15.7017
	10	19.4150	8.3509	39.0791

그림 17. RTiK-MP 3,5,7ms 주기측정

[그림 17]은 RTiK-MP의 주기를 1ms로 설정하고, 3개의 실시간 쓰레드를 생성하여 각각의 주기를 3ms,

5ms, 7ms로 수행한 결과 화면이다. 오실로스코프의 구간은 10ms이고, 각각 위쪽부터 7ms, 3ms, 5ms의 구형파를 나타낸다. 가장 위에 보이는 노란색의 구형파는 7ms의 주기를 나타내는 것으로 7ms마다 신호를 출력하는 함수와 신호를 출력하지 않는 함수를 번갈아 호출함으로써 한 구간인 14ms동안에 7ms동안은 신호의 출력을 유지하고, 나머지 7ms동안은 신호를 출력하지 않는 상태를 반복한 결과를 나타낸다. 실험 결과와 같이 RTiK-MP가 다수의 실시간 쓰레드에 영향을 받지 않고 설정된 주기를 지키며 동작함을 확인할 수 있다.

표 4. RTiK-MP의 워크로드 개수에 따른 주기 변화

주기	While 개수	평균(ms)	최소(ms)	최대(ms)
1ms	0	0.9943	0.9716	1.0159
	1	1.0001	0.9822	1.0168
	10	1.0001	0.9785	1.0168
10ms	0	10.0011	9.9939	10.0105
	1	10.0010	9.9843	10.0062
	10	10.0011	9.9540	10.0638
15ms	0	15.0016	14.9936	15.0052
	1	15.0016	14.9938	15.0108
	10	15.0016	14.9801	15.0104

[표 4]는 RTiK-MP의 Local APIC 타이머에 의한 실시간 쓰레드의 주기적 동작을 동시에 동작하는 다른 프로세스의 개수에 따른 주기변화를 나타낸 표이다. 1, 10, 15ms의 주기를 각각 실험 하였으며, 워크로드에 대한 주기 변화를 확인하기 위해 While문만을 무한 반복하는 프로세스를 생성하여 RTiK-MP의 실시간 쓰레드와 동시에 실행시켰다. 실험 결과 RTiK-MP의 실시간 쓰레드는 윈도우즈에서 동작하는 프로세스의 수에 영향을 받지 않고, 설정된 주기를 지키며 동작하는 것을 확인할 수 있다.

V. 결론 및 향후 연구과제

최근 많은 내장형 시스템들이 싱글프로세서에서 멀티프로세서로 변화함에 따라 점검장비 역시 싱글프로세서에서 멀티프로세서로 변화되고, 이에 따라 점검장비에서 사용되는 운영체제도 윈도우즈 XP에서 윈도우즈 7 기반으로 변화되고 있다. 따라서 점검장비의 환경

변화에 따른 실시간성 제공을 위한 연구가 필요하다.

본 논문에서는 멀티프로세서 환경의 윈도우즈 7 환경에서 실시간성을 제공하기 위한 연구를 하였다. RTiK-MP는 디바이스 드라이버 형태로 구현된 확장 커널로써 멀티프로세서 기반의 윈도우즈의 커널 자원 및 x86 하드웨어의 자원 접근 및 제어가 가능하다. 이를 통해 RTiK-MP는 윈도우즈의 Local APIC를 이용하여 윈도우즈와는 독립적인 타이머 인터럽트를 발생 시킴으로써 윈도우즈에 실시간성을 제공하였다. 이를 위해 INF파일을 이용하여 CPU Affinity를 설정하고, 인터럽트 벡터를 할당받아 지정된 CPU에서 RTiK-MP를 위한 인터럽트 오브젝트를 IDT에 등록하였다. 이를 통해 Dedicate Mode와는 다르게 모든 CPU를 사용할 수 있게 함으로써 전체적인 시스템의 성능을 저해하지 않고 실시간성을 제공하였다.

향후 연구과제로는 우선순위 기반의 실시간 스케줄러를 구현함으로써 현재 RTiK-MP가 제공하는 동일 주기에 우선순위를 부여하기 위한 연구의 진행이 필요하다. 또한 개발자들이 RTiK-MP를 보다 편리하게 사용하기 위한 다양한 API의 제공이 필요하고, RTiK-MP의 높은 신뢰성을 검증하기 위한 여러 장비에서의 성능 검증이 필요하다.

참고 문헌

- [1] <http://www.intervalzero.com>
- [2] D. A. Solomon and M. E. Russinovich, "Inside Windows 2000, Third Edition," Microsoft, 2000.
- [3] W. Oney, "Programming the Microsoft Windows Driver Model 2nd Edition", 정보문화사, 2004.
- [4] 이진욱, 조문행, 김종진, 조한무, 박영수, 이철훈, "윈도우 기반의 점검장비에 실시간성을 지원하는 실시간 이식 커널의 설계 및 구현", 한국콘텐츠학회논문지, 제10권, 제10호, pp.36-44, 2010.
- [5] 김주만, 송창인, 이철훈, "RTiK-Linux: 리눅스용 실시간 이식 커널의 설계", 한국콘텐츠학회논문지, 제11권, 제9호, pp.45-53, 2011.
- [6] 주민규, 이진욱, 장철수, 김성훈, 이철훈, "윈도우

유저 레벨 로봇 컴포넌트에 실시간성 지원 방법”, 한국콘텐츠학회논문지, 제11권, 제7호, pp.51-59, 2011.

- [7] Ronald D. Reeves, Ph.d, "Windows 7 Device Driver," Addison Wesley, 2011
- [8] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1 : Basic Architecture," Intel, 2009.
- [9] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3 : System Programming Guide," Intel, 2009.
- [10] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual Vol.2 : Instruction Set Reference," Intel, 2009.
- [11] Intel, "Intel 64 Architectures x2APIC Specification," Intel, 2008.
- [12] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3 : System Programming Guide," Intel, 2009.
- [13] [http://msdn.microsoft.com/en-us/library/ms685100\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms685100(VS.85).aspx)
- [14] <http://www.osronline.com>
- [15] Johnson M.Hart, "Windows System Programming 3rd Edition," Addison Wesley, 2005.
- [16] JOHNSON M. HART, "Windows 시스템 프로그래밍 3판", 정보문화사, 2008.

저 자 소 개

송 창 인(Chang-In Song)

준회원

- 2010년 8월 : 충남대학교 컴퓨터 공학과(공학사)
- 2010년 8월 ~ 현재 : 충남대학교 컴퓨터 공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 내장형 시스템, 로봇 미들웨어

이 승 훈(Seung-Hoon Lee)

준회원

- 2011년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2011년 2월 ~ 현재 : 충남대학교 컴퓨터 공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템, 로봇 미들웨어

주 민 규(Min-Gyu Jo)

준회원

- 2010년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2010년 2월 ~ 현재 : 충남대학교 컴퓨터 공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 운영체제, 로봇 미들웨어

이 철 훈(Cheol-Hoon Lee)

정회원

- 1983년 2월 : 서울대학교 전자 공학과(공학사)
- 1988년 2월 : 한국과학기술원 전기 및 전자공학과(공학석사)
- 1992년 2월 : 한국과학기술원 전기 및 전자공학과(공학박사)

- 1983년 3월 ~ 1986년 2월 : 삼성전자 컴퓨터 사업부 연구원
- 1992년 3월 ~ 1994년 2월 : 삼성전자 컴퓨터 사업부 선임연구원
- 1994년 2월 ~ 1995년 2월 : Univ. of Michigan 객원 연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙 연구원

<관심분야> : 실시간시스템, 운영체제, 고장허용 컴퓨팅, 로봇 미들웨어