DOI : 10.5392/JKCA.10.10.036

윈도우 기반의 점검장비에 실시간성을 지원하는 실시간 이식 커널의 설계 및 구현

Design and Implementation of Real-time Implanted Kernel, RTiK to Support Real-time for a Test Set based on Windows

이진욱*, 조문행*, 김종진**, 조한무**, 박영수**, 이철훈*
충남대학교 컴퓨터공학과*, (주)LIG넥스원**

Jin-Wook Lee(ljweve@cnu.ac.kr)*, Moon-Haeng Cho(root4567@pony.cnu.ac.kr)*,
Jong-Jin Kim(jjkim@lignex1.com)**, Han-Moo Jo(hanmoojo@lignex1.com)**,
Young-Soo Park(youngsoo.park@lignex1.com)**, Cheol-Hoon Lee(clee@cnu.ac.kr)*

요약

최근 다양한 신무기가 개발됨에 따라 개발된 무기들을 시험하기 위한 점검장비의 실시간성이 필수적으로 요구되고 있다. 하지만 윈도우 기반의 점검장비는 실시간성을 지원하지 못하기 때문에 RTX나 INtime같은 고가의 서드파티 운영체제를 사용함으로써 실시간성을 지원하며, 이는 점검장비 프로그램 개발비용의 증가를 초래하고 있다. 본 논문은 윈도우에 디바이스 드라이버 형태로 이식되어 동작하는 실시간 이식 커널인 RTiK을 제안한다. RTiK은 x86 하드웨어에서 제공하는 Local APIC를 이용하여 윈도우 별도의 타이머를 제공한다. 윈도우 독립적인 타이머 인터럽트의 발생으로 실시간성이 필요한 서비스를 주기적으로 동작시켜주며 마감시한을 보장해줌으로써 윈도우에 실시간성을 제공해준다. 또한 인터럽트 지연시간을 줄이기 위해 윈도우에서 제공하는 지연처리호출(Deferred Procedure Call)을 사용하였으며 지연처리호출에서 실행시켜줄 개발자 정의 함수를 실시간 이식커널 내부에 접근하지 않고 구현 및 수정할 수 있도록 Export Driver를 사용했다. 본 논문에서는 x86하드웨어에서 동작하는 윈도우 기반의 점검장비에 실시간성을 지원하는 실시간 이식커널을 설계 및 구현하고, 0.1ms 주기성 보장에 대해 오실로스코프로 검증한다.

■ 중심어 : | 윈도우 | 점검 장비 | 실시간 이식커널 | 지연처리호출 |

Abstract

Recently, as new weapons are being developed, test equipments to test their functions inevitably require real-time features. However, since test equipments based on Windows can not support real-time requirements, we have no choice but to use third-party solutions such as RTX or INtime. This leads to increase the development cost of each test equipment. This paper suggests an real-time implemented kernel(RTiK) which operates as a device driver on Windows. RTiK provides another timer using the Local APIC of x86 microprocessors. It supports real-time requirements by periodically executing the required services using Windows-independent timer interrupts to guarantee task deadlines. To reduce the interrupt latency, we used deferred procedure calls provided by Windows. We also used the export driver to implement and modify user-defined functions without accessing the RTiK internals. Using an oscilloscope, we prove that the RTiK kernel proposed in this paper guarantees up to 0.1ms periods.

■ keyword : | Windows | Test Equipments | RTiK | Deferred Procedure Calls |

* 본 연구는 LIG넥스원의 점검장비용 실시간 윈도우 운영체계(RTiK) 개발 연구과제로 수행되었습니다.

접수번호 : #100915-012
접수일자 : 2010년 09월 15일
심사완료일 : 2010년 10월 25일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

I. 서 론

최근 유도무기를 비롯한 무기의 발달에 따라 각국에서는 다양한 신무기들의 개발이 활발히 이루어지고 있다. 개발된 신무기는 여러가지 조건들에 대해 요구된 동작이 제대로 동작하는지 성능검증을 하기 위해 수락시험을 수행해야 한다. 특히 유도무기체계 사업에서는 실시간으로 데이터를 획득하고 평가하는 휴대용 점검장비를 선호한다. 현재 휴대용 점검장비는 실시간성의 제공을 위해 VxWorks와 같은 실시간 운영체제를 사용하거나 윈도우 기반하에 실시간 지원을 위한 RTX나 INtime과 같은 서드파티 운영체제를 사용하고 있다. 하지만 이러한 제품들은 고가의 구입비용과 라이선스 비용으로 인해 개발비용의 증가를 초래할 수 있다. 따라서 점검장비 개발시 고가의 서드파티 운영체제를 대체하여 개발비용을 절감하기 위해 윈도우의 실시간성 지원에 관한 연구가 필요하다.

본 논문에서는 윈도우의 실시간성 제공을 위해 디바이스 드라이버 형태로 윈도우에 이식되어 동작하는 실시간 이식 커널을 설계 및 구현하였다. 실시간 이식 커널은 인텔 x86 하드웨어에서 제공하는 Local APIC를 이용하여 윈도우 독립적인 타이머 인터럽트를 발생시킨다. 실시간 이식커널의 타이머는 최소 0.1ms의 주기로 동작하며 인터럽트 지연시간을 줄이기 위해 윈도우에서 제공하는 지연처리호출(Deferred Procedure Call)을 사용하였다. 또한 지연처리호출에서 주기적으로 수행하는 실시간 서비스를 개발자가 실시간 이식커널의 내부에 접근하지 않고 점검장비 프로그램을 개발할 수 있어야 하므로 Export Driver를 이용하여 개발자가 실시간 이식커널의 내부에 접근하지 않고 지연처리호출에서 실행시켜, 실시간성을 필요로 하는 점검장비 프로그램을 개발할 수 있도록 하였다.

본 논문의 구성은, 관련연구에서 윈도우의 실시간성 제공을 위한 서드파티 운영체제에 대해 알아보고, 실시간 이식 커널이 이용하는 x86하드웨어와 지연처리호출, Export Driver에 대해 기술한다. 3장에서는 실시간 이식 커널의 설계 및 구현에 대해 기술하고, 4장에서 실시간 이식 커널의 성능을 분석하였으며, 마지막 5장에서 결론을 맺는다.

II. 관련연구

1. 윈도우의 실시간성 제공을 위한 서드파티 운영체제

범용 운영체제인 윈도우에 실시간성 제공을 위한 대표적 서드파티 운영체제로 RTX와 INtime이 있다. RTX는 윈도우에 고성능 실시간 제어가 가능한 실시간 운영체제의 기능을 부가해주는 확장 소프트웨어로서, 순수 실시간 운영체제가 아니라 윈도우의 대중성 및 풍부한 GUI Library 등의 장점을 최대한 이용하며 실시간성을 보완하여 주는 소프트웨어이다. INtime은 동일 하드웨어 상에서 윈도우와 동시에 동작하는 실시간 운영체제이며 INtime이 설치된 윈도우는 2개의 커널이 동작하는 멀티 커널 시스템이 된다.

두 서드파티 운영체제 모두 x86에 특화된 소프트웨어로서 장비개발시 구입비용과 양산 라이선스 비용 부담이 있다. 그 특징은 아래의 [표 1]과 같다.

표 1. 윈도우의 서드파티 운영체제

	RTX	INtime
개발사	IntervalZero	TenAsys
커널변환	소프트웨어적인 커널변환	하드웨어적인 커널변환
동작모드	커널모드(Ring 0) 동작	사용자모드(Ring 3) 동작

2. x86 하드웨어

Local APIC는 x86 하드웨어의 PIC(Programmable Interrupt Controller)의 확장으로 프로그램 가능한 인터럽트 컨트롤러이다. Local APIC 타이머는 Initial Count 레지스터에 설정된 카운트 값으로부터 복사된 Current Count 레지스터의 카운트 값이 버스 클럭에 의해 감소하며 0에 도달하는 순간 인터럽트가 발생한다. 프로세서는 Local APIC로부터 인터럽트의 발생 사실을 전달받아 LVT 타이머 레지스터의 Vector 비트[0:7], 즉 IDT의 주소를 나타내는 인덱스를 참조하여 타이머 인터럽트 서비스 루틴을 수행한다. 타이머 인터럽트가 발생하면 Initial Count 레지스터의 카운터 값이 Current

Count레지스터로 다시 복사되어 같은 과정을 반복하게 된다[1][2].

IDT는 8바이트 크기의 디스크립터 256개의 집합으로, x86 하드웨어는 소프트웨어나 하드웨어에서 인터럽트가 발생 시 인터럽트의 처리루틴 정보를 통해 해당 인터럽트를 즉시 처리할 수 있도록 인터럽트 처리루틴의 정보를 IDT에 저장하고 있다. 인터럽트가 발생하면 해당 인터럽트 벡터에 해당하는 게이트를 찾아 인터럽트를 처리한다[3].

3. 윈도우의 스케줄링 및 지연처리호출

윈도우의 스케줄러는 31단계의 우선순위 기반하에 공평성에 초점을 맞춘 스케줄링 방식을 사용한다. 윈도우의 프로세스는 IDLE_PRIORITY_CLASS부터 REALTIME_PRIORITY_CLASS까지 6단계의 프로세스 기본 우선순위가 존재하고 쓰레드는 PRIORITY _IDLE부터 PRIORITY_TIME_CRITICAL까지 7단계의 쓰레드 우선순위를 갖는다[4].

프로세스를 생성하면 NORMAL_PRIORITY_CLASS의 PRIORITY_NORMAL 쓰레드가 생성되어 동작하며, 쓰레드가 동작되는 상황에 따라 우선순위 증가치값을 두어 런타임 우선순위를 높일 수 있다. 런타임 우선순위는 실제 윈도우의 스케줄러가 스케줄링시 판단기준이 되는 0부터 31까지의 상수값이다. 런타임 우선순위를 기반으로 타임퀀텀 값만큼 CPU를 할당받아 수행되고, 타임퀀텀이 만료되면 쓰레드 큐의 마지막에 연결된다. 이때 스케줄러는 기아상태를 방지하기 위해 CPU를 할당받지 못한 쓰레드의 런타임 우선순위를 높여주어 CPU를 할당받게 하기 때문에 우선순위 기반의 윈도우 스케줄러는 실시간성을 지원하지 못한다[5].

지연처리호출(Deferred Procedure Call : DPC)이란 하드웨어 인터럽트의 ISR(Interrupt Service Routine)이 CPU를 사용함에 있어서 중요한 작업에만 CPU를 사용하고, 나머지 작업은 차후에 낮은 우선순위의 IRQL (Interrupt Request Level)에서 다시 이어서 하도록 제공되는 윈도우의 메커니즘이다[6][7].

4. Export Driver

Export Driver는 Kernel mode DLL(Dynamic Link Library)라고도 하며 커널모드 드라이버에서 유저모드의 DLL과 비슷한 형태로 쓸 수 있는 드라이버이다. Export Driver에 정의된 특정한 함수는 또 다른 드라이버에서 호출하여 사용할 수 있다[8][9].

Ⅲ. 실시간 이식커널의 설계 및 구현

윈도우 기반의 점검장비는 개발된 신무기의 수락시험을 위해 실시간으로 데이터를 획득하고 평가할 수 있어야 한다. 수락시험은 주기적으로 데이터를 획득하고 전달하는 쓰레드와 그 데이터를 특정한 알고리즘을 통해 평가하는 쓰레드를 통해 진행된다. 특히 주기적으로 데이터를 획득하고 전달하는 쓰레드가 실시간성이 요구되며, 그 주기는 최소 0.1ms이다. 윈도우XP는 1ms의 최소 주기를 갖춘 큐타이머를 제공하지만 실제 해당주기대로 실시간성을 보장하지 못한다. RTiK 은 x86 하드웨어에서 제공하는 Local APIC의 타이머를 통해 최소 0.1ms의 주기적인 동작을 보장한다.

실시간 이식 커널 RTiK은 윈도우상에서 실시간성을 지원하기 위해 이식된 독립적인 커널로써 전체적인 구조는 [그림 1]과 같다. RTiK은 윈도우 디바이스 드라이버 형태로 구현되어 윈도우 커널영역에서 사용가능한 자원의 접근이 용이하며 Local APIC의 제어를 위해 윈도우와는 별도의 RTiK Real-time HAL(Hardware Abstraction Layer)과 RTiK Kernel을 가진다.

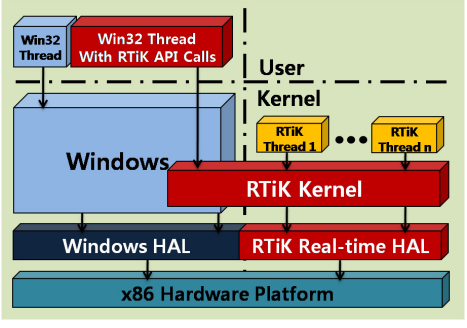

그림 1. 실시간 이식커널의 구조

1. 실시간 이식 커널을 위한 인터럽트 오브젝트 등록

윈도우는 기본적으로 프로세스와 쓰레드의 우선순위를 가지고 라운드로빈(Round-Robin) 스케줄링을 한다. 쓰레드의 우선순위가 낮더라도 기아상태를 방지하기 위해 CPU를 할당해주기 때문에 실시간 쓰레드의 마감시한(Dead-Line)을 보장하지 못하므로 실시간성을 지원하지 못한다. 이러한 문제를 해결하기 위해 실시간 이식커널은 Local APIC 타이머를 통해 윈도우 독립적인 타이머 인터럽트를 발생시킴으로써 실시간 쓰레드의 주기적인 수행을 가능하게 하였다.

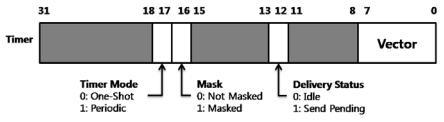

그림 2. 타이머 레지스터

x86 하드웨어에서 제공하는 Local APIC는 LVT (Local Vector Table)라는 6개의 레지스터의 집합 중 타이머 레지스터의 설정으로 주기적인 인터럽트를 발생시킬 수 있다. [그림 2]는 LVT의 타이머 레지스터의 모습이다. Local APIC의 타이머 인터럽트가 발생하면 인터럽트를 처리할 인터럽트 오브젝트의 IDT 벡터넘버가 저장되어 있는 타이머 레지스터의 벡터비트를 참조하여 해당하는 인터럽트 오브젝트로 분기하게 된다.

```
MappedVector = HalGetInterruptVector(
                Isa,
                0,
                deviceExtension->Level,
                deviceExtension->InterruptVector,
                &Irql,
                &deviceExtension->Affinity);

returnStatus = IoConnectInterrupt(
                &deviceExtension->InterruptObject,
                InterruptIsr,
                DeviceObject,
                NULL,
                MappedVector,
                Irql,
                Irql,
                Latched,
                TRUE,
                deviceExtension->Affinity,
                FALSE);
```

그림 3. 인터럽트 오브젝트 생성 코드

인터럽트 오브젝트에 저장되어 있는 인터럽트의 핸들러 주소를 참조하여 주기적으로 발생되는 타이머 인터럽트의 핸들러를 수행한다. 이러한 인터럽트 처리 과정을 위해 윈도우의 IDT에 실시간 이식 커널을 위한 인터럽트 오브젝트를 등록시켜야 한다.

[그림 3]과 같이 실시간 이식커널 내에서 윈도우가 제공하는 API인 HalGetInterruptVector()함수로 IDT의 벡터넘버를 얻는다. 이 함수의 인자값으로는 하드웨어 인터럽트의 IRQ넘버가 필요하며 윈도우는 해당 인터럽트를 처리하기 위한 인터럽트 오브젝트를 등록시킬 수 있는 적절한 벡터를 할당해 준다. 윈도우로부터 할당받은 벡터값을 나타내는 MappedVector와 실시간 이식커널이 사용할 ISR 함수명인 InterruptIsr을 인자값으로 IoConnectInterrupt()함수를 통해 인터럽트 오브젝트 및 ISR을 등록시켜준다.

2. 실시간 이식 커널을 위한 지연처리호출 루틴의 등록

실시간 이식커널은 주기적인 타이머 인터럽트를 통해 실시간성을 보장할 수 있지만 낮은 우선순위의 인터럽트는 인터럽트 지연시간이 길어질 수 있다. 따라서 본 논문에서는 실시간 이식커널이 윈도우가 제공하는 지연처리호출 루틴을 사용하도록 설계 및 구현하였다. 지연처리호출 메커니즘을 사용하려면 윈도우로부터 IDT의 벡터넘버를 할당 받고 인터럽트 오브젝트를 등록하기 전에 지연처리호출을 위한 초기화 작업이 필요하다.

```
IoInitializeDpcRequest(
    DeviceObject, InterruptDpcRoutine);
```

그림 4. 지연처리호출 메커니즘의 초기화

[그림 4]에서와 같이 윈도우의 I/O 관리자가 제공하는 API인 IoInitializeDpcRequest()함수는 ISR내에서 지연처리호출을 요청한 후에 IRQL이 Passive레벨로 떨어지며 Dispatch레벨에서 수행될 함수를 등록해준다. 즉, 실제로 지연처리호출 레벨에서 동작할 [그림 5]의 InterruptDPCRoutine()함수를 등록해준다.

```
BOOLEAN InterruptIsr(IN PKINTERRUPT Interrupt,
                     IN OUT PVOID Context){

    PDEVICE_OBJECT DeviceObject
        =(PDEVICE_OBJECT)Context;

    IoRequestDpc(DeviceObject,
                 DeviceObject->CurrentIrp,
                 InterruptDpcRoutine);
    return TRUE;
}
VOID InterruptDpcRoutine(IN PKDPC Dpc,
                 PDEVICE_OBJECT DeviceObject,
                 IN PIRP Irp, IN PVOID Context){

    DbgPrint("DPC Routine !!!\n");
}
```

그림 5. 실시간 이식커널의 ISR과 DPC

실시간 확장커널의 ISR에 해당하는 [그림 5]의 InterruptIsr()함수에서 윈도우의 I/O매니저에서 제공하는 IoRequestDpc()함수를 이용해 지연처리호출을 요청한다. ISR 처리가 끝난 후 IRQL이 떨어지며 Dispatch 레벨이 될 때 [그림 4]에서 등록한 지연처리호출의 처리함수인 InterruptDpcRoutine()함수가 실행된다.

3. 실시간 이식커널을 위한 Export Driver의 구현

실시간 이식커널은 윈도우 독립적인 타이머 인터럽트를 발생시키며 지연처리호출에서 개발자가 정의한 동작을 주기적으로 처리해준다. 이때 개발자가 실시간 이식커널의 내부에 접근하지 않고 지연처리호출에서 호출될 개발자 함수를 구현하거나 수정할 수 있어야 한다. 실시간 이식커널은 Export Driver에 개발자가 원하는 함수를 구현하고 실시간 이식커널의 타이머가 활성화되면, 윈도우 독립적인 타이머 인터럽트가 발생하여 지연처리호출에서 개발자가 정의한 함수 호출을 통해 주기적인 동작을 한다.

[그림 6]은 Export Driver를 이용한 실시간 이식커널의 개발자 정의함수를 호출하는 과정을 나타낸 그림이다. 개발자는 Export Driver에 Func0(), Func1()과 같이 함수를 구현하고, 실시간 이식커널은 지연처리호출에서 주기에 따라 정의된 함수를 호출한다. 개발자는 실시간 이식커널의 API를 사용하여 실시간 쓰레드의 생성을 요청하며, 인자 값으로는 실시간 이식커널 타이머의 주기와 실시간 쓰레드들을 관리할 배열의 개수, 실시간 쓰레드의 주기, 생성한 실시간 쓰레드가 사용할 함수번호, 마지막으로 실시간 쓰레드의 ID를 나타내는 번호가 들어간다.

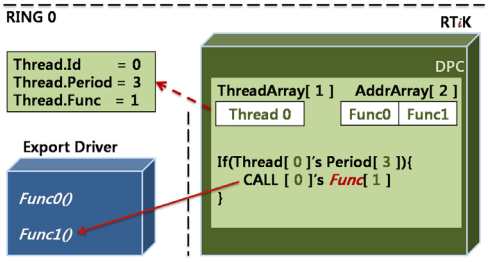

그림 6. 실시간 이식커널의 Export Driver

실시간 이식커널의 내부에서는 개발자의 요청을 받아 커널영역에서 실시간 쓰레드를 생성하며 생성된 실시간 쓰레드는 ID넘버 순서로 연결되어 배열 형태로 관리된다. 실시간 이식커널의 타이머 인터럽트가 발생하면 지연처리호출에서 배열 ThreadArray에 저장되어 있는 구조체를 검사하여 해당 실시간 쓰레드의 주기에 맞게 각각의 함수를 호출해준다.

4. 실시간 이식 커널의 동작과정

[그림 7]과 같이 실시간 이식커널은 윈도우에 이식되어 가장 먼저 초기화 단계를 거친다. 초기화 단계에서는 실시간 이식커널을 위한 인터럽트 오브젝트 등록, 지연처리호출의 초기화, 타이머 레지스터의 설정 등이 이루어진다. 그 후 개발자로부터 실시간 쓰레드의 생성과 타이머 인터럽트의 활성화 명령을 전달 받으면 실시간 이식 커널의 내부에서 실시간 쓰레드가 생성되며 타이머 활성화 명령에 따라 윈도우 독립적인 타이머 인터럽트가 발생한다. 발생된 인터럽트는 타이머 레지스터의 벡터비트에 설정된 실시간 이식커널을 위한 인터럽트 오브젝트의 주소를 참조하여 인터럽트 서비스 루틴을 처리한다.

실시간 이식커널의 내부에 정의되어 있는 인터럽트 서비스 루틴에서는 지연처리호출 메커니즘을 요청하고 실제로 지연처리호출 루틴에서 개발자가 미리 정의해 놓은 Export Driver내에 개발자가 정의한 함수를 수행한다.

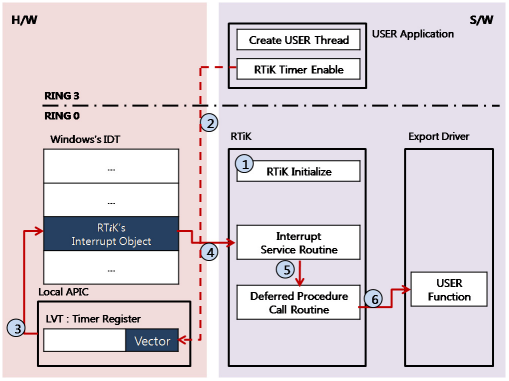

그림 7. 실시간 이식 커널의 동작과정

실시간 이식커널의 타이머는 최소 0.1ms의 주기를 지원하며 1000ms까지 설정이 가능하다. 또한 윈도우에서 제공하는 지연처리호출 메커니즘을 이용해서 인터럽트 지연시간을 줄이며 주기적인 동작을 함으로써 윈도우에 실시간성을 제공해 줄 수 있다.

Ⅳ. 실험환경 및 결과

1. 실험 환경

윈도우에 실시간성을 제공하는 실시간 이식 커널을 구현하기 위해 [표 2]와 같은 실험환경을 구성하였다.

타겟에서 실시간 이식커널을 동작시켰으며 호스트에서 모니터링을 하였다. 유도탄의 수락시험을 위한 휴대용 점검장비를 타겟으로 하여 윈도우를 설치하고 실시간 이식커널을 이식하여 실험하였다. 실시간 이식 커널의 동작 및 주기를 확인하기 위해 휴대용 점검장비에 Acromag APC 8625 PCI Carrier 보드를 사용하였고, IP470 Digital OutPut Module을 사용하여 포트에 신호를 출력하였다.

실시간 이식커널 타이머의 주기를 확인하기 위해 [그림 8]과 같이 실험환경을 구성하였다. 정확한 주기측정을 위해 IP470 모듈을 이용하여 점검장비의 포트에 신호를 출력해주고 오실로스코프로 확인하였다.

표 2. 실험환경

	호스트	타겟
CPU	Intel Core(TM)2 Duo 3GHz	Intel Pentium M 1.8GHz
OS	Windows XP SP3	Windows XP SP3
DDK	WDK 6001.18002	WDK 6001.18002

Acromag APC 8625 PCI Carrier 보드에 탑재되어 동작하는 IP470 Digital OutPut Module의 물리적 주소에 맵핑되어 있는 가상주소, 즉 해당포트를 통해 시그널을 출력하였다. IP470은 총 6개의 포트를 사용하며 각각의 포트는 8개의 채널을 사용한다. Export Driver에 정의된 함수는 Acromag APC 8625 PCI Carrier 보드를 제어하고 실제로 점검장비의 포트에 신호를 전송해주는 IP470 Digital OutPut Module을 제어하는 기능을 한다.

본 논문에서는 IP470 레지스터의 포트0에 해당하는 8개의 채널을 이용해 실험하였고, 타이머 인터럽트 발생시 0x01을 세팅하고 다음 인터럽트 발생시 0x00을 세팅하여 동작하는 모습을 볼 수 있도록 실험하였다. 또한 디스크를 사용하는 응용 프로그램을 동시에 실행시켰을 때의 반응속도를 확인하기 위해 텍스트 파일을 만들어 입력과 저장을 반복하는 프로세스를 생성해 동시에 실행시키며 실험했다.

그림 8. 실험환경

2. 실험 결과

[그림 9]는 10개의 워크로드를 실행시키며 실시간 이식 커널의 타이머 주기를 0.1ms로 설정했을 때의 주기

를 측정한 오실로스코프의 결과화면이다.

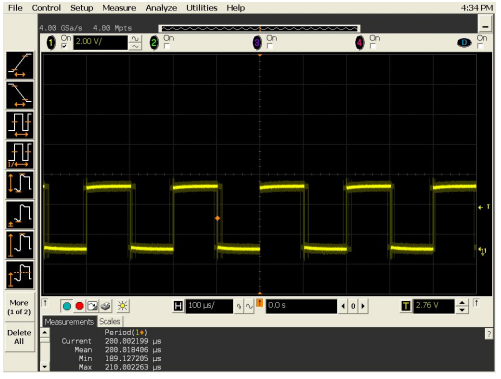

그림 9. 실시간 이식커널의 동작 – 10개의 워크로드가 있는 상태의 0.1ms

10개의 워크로드는 사용자 영역에서 동작하는 다수의 병렁 프로세스들로써 각각의 프로세스는 While문을 무한 반복한다. 오실로스코프의 한 구간은 0.1ms이며, 오차 없이 신호출력구간과 신호를 출력하지 않은 구간이 각각 0.1ms동안 유지되는 것을 확인할 수 있다. 이것은 실시간 이식커널이 윈도우의 스케줄링에 영향을 받지 않고 항상 설정된 주기를 지키며 동작함으로써 윈도우에 실시간성을 제공할 수 있음을 의미한다.

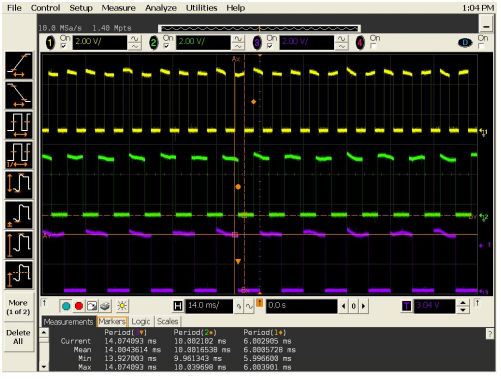

그림 10. 실시간 이식커널의 동작 – 3,5,7ms

[그림 10]은 실시간 이식 커널의 타이머 주기를 1ms로 설정하고 3개의 실시간 쓰레드를 생성하여 각각의 주기를 3, 5, 7ms로 수행한 결과 화면이다. 오실로스코프의 한 구간은 14ms이며 가장 위쪽부터 색깔별로 3, 5, 7ms의 구형파를 나타낸다. 가장 밑에 보이는 보라색의 구형파는 7ms의 주기를 나타내는 것으로 7ms마다 신호를 출력하는 함수와 신호를 출력하지 않는 함수를 번갈아 호출함으로써 한 구간인 14ms동안에 7ms동안은 신호출력을 유지하고, 나머지 7ms동안에는 신호를 출력하지 않는 상태를 반복한 결과를 나타낸다. 실험 결과와 같이 실시간 이식 커널이 윈도우의 스케줄러 및 다른 프로세스에 영향을 받지 않고 항상 설정된 주기를 지키며 동작함을 확인할 수 있다.

[표 3]은 실시간 이식커널의 타이머에 의한 실시간 쓰레드의 주기적 동작을 동시에 동작하는 다른 프로세스의 개수에 따른 주기변화를 나타낸 표이다. 0.1, 1, 15ms의 주기를 각각 실험하였으며 워크로드에 대한 주기변화를 확인하기 위해 While문을 무한 반복하는 윈도우 프로세스를 생성하였다.

표 3. While 개수에 따른 실시간 이식커널의 주기변화

주기	While 개수	평균(ms)	최소(ms)	최대(ms)
0.1 ms	0	0.100006	0.0960	0.1030
	1	0.100025	0.0987	0.1091
	10	0.100039	0.0887	0.1135
1 ms	0	1.000090	0.9946	1.0062
	1	1.000114	0.9905	1.0096
	10	1.000111	0.9902	1.0101
15 ms	0	15.001673	14.9958	15.0054
	1	15.001545	14.9919	15.0067
	10	15.001581	14.9924	15.0122

실험결과 윈도우에서 동작하는 프로세스가 많아질수록 윈도우의 스케줄링을 위한 인터럽트에 의해 인터럽트 지연시간이 길어져 실시간 쓰레드의 미미한 영향을 미치지만 주기가 커질수록 거의 영향을 받지 않고 동작하는 것을 확인할 수 있다.

또한 디스크를 사용하는 응용프로그램이 동시에 실행되어도 실시간 이식 커널은 영향을 받지 않고 설정된 주기대로 동작하는 것을 확인할 수 있다.

V. 결론 및 향후연구과제

휴대용 점검장비는 실시간으로 데이터를 획득하고 평가하기 때문에 운영체제로부터의 실시간성 제공이 필수적으로 요구된다. 실시간 이식커널은 디바이스 드라이버 형태로 구현되어 윈도우의 커널자원과 x86하드웨어의 자원 접근 및 제어가 가능하다. 또한 Local APIC를 제어하여 윈도우 독립적인 타이머 인터럽트를 기반으로 실시간 이식커널의 주기성을 갖는 함수실행을 통해 윈도우에 실시간성을 제공한다.

본 논문에서는 윈도우가 제공하는 지연처리호출을 사용하여 ISR의 작업을 하드웨어 인터럽트 레벨보다 낮은 Dispatch레벨에서 처리함으로써 낮은 우선순위의 인터럽트를 방해하지 않고, 인터럽트 지연시간을 줄였다. 이를 위해 윈도우의 I/O 관리자와 커널이 제공하는 API를 이용해 IDT에 실시간 이식커널을 위한 인터럽트 오브젝트를 등록하고 지연처리호출을 사용하였으며 Export Driver를 이용해 개발자가 지연처리호출의 내부수정 없이 실시간성을 필요로 하는 점검장비 프로그램을 개발할 수 있도록 하였다.

향후 연구과제로는 우선순위 기반 실시간 쓰레드 스케줄러를 구현함으로써 다수의 실시간 쓰레드의 우선순위에 의한 동작을 보장하는 것이다. 또한 다양한 API를 제공함으로써 개발자로 하여금 보다 편리하게 실시간 이식커널을 사용하도록 하는 연구와 여러 장비에서의 성능검증이 필요하다.

참 고 문 헌

[1] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1 : Basic Architecture,* September, 2009.

[2] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3 : System Programming Guide,* September, 2009.

[3] David A. Solomon, Mark E. Russinovich, *Inside Windows 2000, Third Edition,* Microsoft, 2000.

[4] http://msdn.microsoft.com/en-us/library/ms685100(v=VS.85).aspx

[5] 정덕영, *Windows 구조와 원리,* 한빛미디어, 2009.

[6] Mark E. Russinovich and David A. Solomon, *Microsoft Windows Internals 4th Edition,* Microsoft Press, 2006.

[7] http://msdn.microsoft.com/en-us/library/ms810029.aspx.

[8] http://www.microsoft.com/whdc/driver/tips/KmDLL.mspx

[9] http://msdn.microsoft.com/en-us/library/ff542891(VS.85).aspx

[10] 이봉석, *윈도우 디바이스 드라이버,* 한빛미디어, 2009.

[11] Walter Oney, *Programming the Microsoft Windows Driver Model 2nd Edition,* 정보문화사, 2004.

[12] 노재현, *개발자를 위한 나만의 운영체제 만들기,* 정보문화사, 2007.

[13] 임재석, *Windows 상에서의 지능형 서비스 로봇을 위한 실시간성 지원에 대한 연구,* 제어로봇시스템학회, pp213-216, 2010.

[14] John Robbins, *DEBUGGING APPLICATIONS For Microsoft .NET And Microsoft WINDOWS,* Microsoft, 2003.

[15] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2 : Instruction Set Reference,* September, 2009.

[16] Johnson M.Hart, *Windows System Programming 3rd Edition,* Addison Wesley, 2005.

[17] http://technet.microsoft.com/en-us/sysinternals/default.aspx

[18] Intel, *Intel I/O Controller Hub 6(ICH6) Family Datasheet,* January, 2005.

[19] Macmillan Technical Publishing Staff, *Windows NT/2000 Native API Reference,* NewRiders Publishing, 2000.

[20] http://msdn.microsoft.com/en-us/library/ff542078.aspx.

[21] http://www.ntkernel.com/

저 자 소 개

이 진 욱(Jin-Wook Lee) 준회원

- 2009년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2009년 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 내장형 시스템, 로봇 미들웨어

조 문 행(Moon-Haeng Cho) 정회원

- 2004년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2006년 2월 : 충남대학교 컴퓨터공학과(공학석사)
- 2008년 2월 : 충남대학교 컴퓨터공학과(공학박사수료)
- 2008년 ~ 현재 : 충남대학교 컴퓨터공학과 박사과정 재학

<관심분야> : 실시간 컴퓨팅, 실시간 운영체제, 초소형 초절전 실시간 운영체제

김 종 진(Jong-Jin Kim) 정회원

- 1995년 : 한국해양대학교(공학사)
- 1995년 ~ 현재 : LIG 넥스원 책임연구원
- 2009년 ~ 현재 : 충남대학교 석사과정

<관심분야> : 실시간 운영체제, 임베디드 시스템, 고장허용 시스템

조 한 무(Han-Moo Jo) 정회원

- 1995년 : 숭실대학교(공학사)
- 1994년 ~ 현재 : LIG 넥스원 책임연구원

<관심분야> : 임베디드 시스템, 실시간 운영체제

박 영 수(Young-Soo Park) 정회원

- 1987년 : 한양대학교(공학사)
- 1997년 ~ 현재 : LIG 넥스원 수석연구원

<관심분야> : 실시간 시스템, 실시간 운영체제

이 철 훈(Cheol-Hoon Lee) 정회원

- 1983년 2월 : 서울대학교 전자공학과(공학사)
- 1988년 2월 : 한국과학기술원 전기및전자공학과(공학석사)
- 1992년 2월 : 한국과학기술원 전기및전자공학과(공학박사)
- 1983년 3월 ~ 1986년 2월 : 삼성전자 컴퓨터사업부 연구원
- 1992년 3월 ~ 1994년 2월 : 삼성전자 컴퓨터사업부 선임연구원
- 1994년 2월 ~ 1995년 2월 : Univ. of Michigan 객원연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙연구원

<관심분야> : 실시간시스템, 운영체제, 로봇 미들웨어