휴대용 점검장비에서 윈도우즈의 지연처리호출(DPC)을 이용한 실시간 이식커널(RTiK)의 설계 및 구현

The Design and Implementation of Real-Time Implanted Kernel, RTiK with The Deferred Procedure Call of Windows on Portable Test Set

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

Lee jin-wook, Kim jong-jin*, Jo han-moo*,
Lee cheol-hoon
Dept. of Computer Engineering, Chungnam National Univ., LIG Nex1 Corp.*

요약

최근 IT산업의 발달과 더불어 내장형 시스템에서의 실시간성은 더욱 중요시되고 있다. 내장형 시스템에서 사용되는 많은 운영체제 중 윈도우즈는 실시간성 지원의 부재로 로봇 플랫폼이나 점검장비와 같은 실시간성이 필수적으로 요구되는 시스템에는 적합하지 않다. 이러한 결점을 보완하기 위해 개발된 실시간 이식커널(Real-Time implanted Kernel)은 윈도우즈에 실시간성을 보장해주지만 ISR(Interrupt Service Routine) 처리시간이 길어질 수 있는 문제가 있다. 본 논문에서는 ISR의 작업을 윈도우즈가 제공하는 지연처리호출(Deferred Procedure Call)에서 처리함으로써 인터럽트 지연시간을 줄이는 실시간 이식커널(RTiK)을 설계 및 구현하였다.

Ⅰ. 서론

최근 기술융합추세와 더불어 다양한 신무기들을 개발하기 위한 연구와 투자에 박차를 가하고 있다. 이런 최신 무기들은 성능검증을 위해 사전에 여러 조건들에 대하여 요구하는 성능을 발휘하는지 확인하기 위하여 기능시험, 즉 수락시험을 수행하여야 하며 이를 위해 점검장비가 필요하다. 특히, 유도무기체계 사업에서는 실시간으로 데이터를 획득하고 평가하는 휴대용 점검장비를 선호한다. 휴대용 점검장비는 실시간성을 지원하기 위해 실시간 운영체제를 사용해야 하는데 현재 휴대용 점검장비 개발에 사용되는 윈도우즈는 실시간성을 지원하지 못하기 때문에 RTX와 같은 고가의 써드파티(Third Party)를 사용하고 있다. 따라서 점검장비의 개발에 있어서 고가의 개발비용을 절감하기 위한 윈도우즈의 실시간성 지원에 관한 연구가 필요하다.

본 논문에서는 윈도우즈에 실시간성을 지원하기 위해 인텔 x86 하드웨어의 Local APIC(Advanced Programmable Interrupt Controller)를 이용하여 주기적인 타이머 인터럽트를 발생시키는 실시간 이식커널을 설계 및 구현하였다. 또한 ISR의 처리시간이 길어짐에 따라 발생하는 인터럽트 지연시간을 줄이기 위해 윈도우즈가 제공하는 지연처리호출을 사용하였다. 본 논문은 2장에서 관련연구로 Local APIC와 지연처리호출에 대해 기술하고 3장에서는 지연처리호출을 이용한 실시간 이식커널의 설계 및 구현 내용을 설명한다. 4장에서는 실험환경 및 결과를, 마지막 5장에서는 결론 및 향후 연구과제에 대해 기술한다.

Ⅱ. 관련 연구

1. Local APIC

APIC는 PIC(Programmable Interrupt Controller)의 확장으로 x86 아키텍쳐에서 제공하는 인터럽트 컨트롤러이다. APIC는 하드웨어 인터럽트를 인터럽트 핸들러의 주소를 가지고 있는 IDT(Interrupt Descriptor Table)로 전달해주는 역할을 한다. APIC는 Local APIC와 I/O APIC로 구성되어 있는데 I/O APIC는 외부장치로부터 IRQ 신호를 받아서 Local APIC에 전달하고, 이것을 Local APIC가 프로세서에 전달하여 인터럽트를 처리한다[1]. 또한 Local APIC는 타이머를 이용해 인터럽트를 발생시킬 수 있는데 실시간 이식커널은 Local APIC의 타이머를 통해 윈도우즈 독립적인 타이머 인터럽트를 발생시킨다.

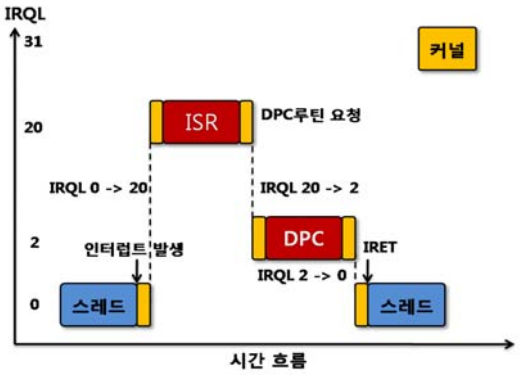
2. 윈도우즈의 지연처리호출 메커니즘

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 윈도우즈의 메커니즘이다[2]. IRQL은 인터럽트 요청 레벨을 의미하고 주어진 IRQL에서 수행되는 코드는 그보다 낮거나 동일한 IRQL 코드에 의해 인터럽트 될 수 없다. [표 1]은 각각의 IRQL과 그에 대한 설명이다.

표 1. IRQL

IRQL	명칭	설명
31	High	Bus error, address error, reset, ...
30	Power	Power failure
29	Inter CPU	Inter processing notification
28	Clock	Real time clock
27	Profile	Performance measurements
26	Device n	I/O Controllers
...	...	...
3	Device 1	I/O Controllers
2	Dispatch	Thread dispatching and DPC interrupts
1	APC	Asynchronous procedure calls
0	Passive	All Interrupt Enable

IRQL이 높은 인터럽트가 발생하게 되면, CPU는 현재 Thread를 저장하고, 해당 인터럽트를 처리하는데 이 인터럽트 중에는 중요하거나 우선 처리되지 않아도 되는 경우가 있을 수 있다. 이때 지연처리호출을 이용하게 된다.

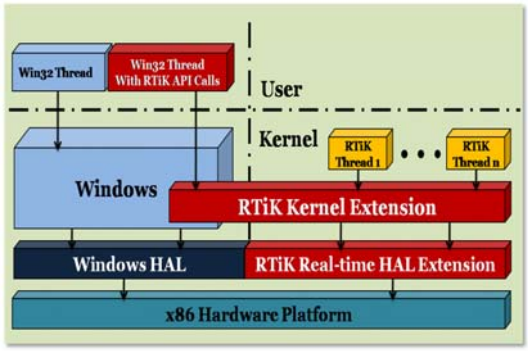


▶▶ 그림 1. 인터럽트 처리시의 IRQL의 변화

[그림 1]과 같이 인터럽트가 발생하면 IRQL을 높여 ISR을 실행한다. 만약 실행되는 스레드가 덜 중요한 경우 지연처리호출을 요청하고 ISR이 종료되는 시점에 IRQL을 변경하여 지연처리호출의 동작레벨인 Dispatch 레벨로 낮아질 때, 지연처리호출의 처리함수를 실행하고 원래의 스레드로 복귀한다. 이처럼 하드웨어 인터럽트의 ISR이 CPU를 사용함에 있어서 중요한 작업에만 CPU를 사용하고, 나머지 작업은 차후에 낮은 우선순위의 IRQL에서 다시 이어서 하도록 제공되는 메커니즘을 지연처리호출이라고 부른다.

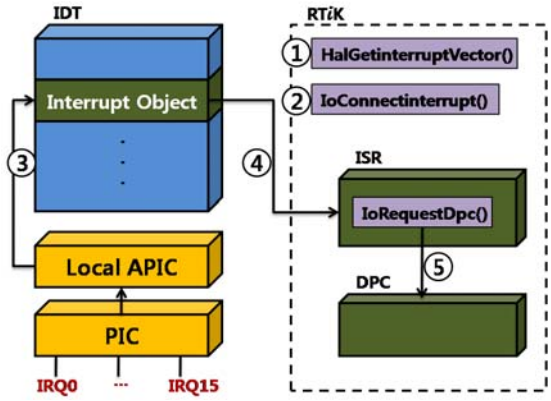
Ⅲ. 지연처리호출을 이용한 실시간 이식 커널의 설계 및 구현

윈도우즈에 실시간성을 제공해주는 실시간 이식커널은 디바이스 드라이버 형태로 이식되어 윈도우의 커널자원 및 하드웨어 자원을 이용할 수 있다[3][4].



▶▶ 그림 2. 실시간 이식커널의 구조

또한 x86 하드웨어가 제공하는 Local APIC의 제어를 위해 윈도우즈와는 별개의 HAL(Hardware Abstraction Layer)을 가진다. [그림 2]는 실시간 이식커널의 전체적인 구조이다.

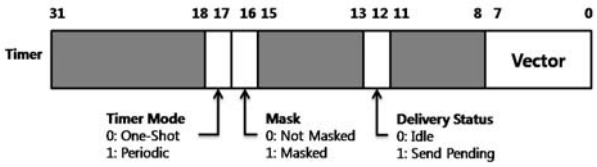


▶▶ 그림 3. 실시간 이식커널의 동작과정

[그림 3]과 같이 실시간 이식커널은 윈도우즈의 IDT에 인터럽트 오브젝트를 등록하여 Local APIC 타이머 인터럽트가 발생하면 윈도우즈로부터 할당받은 인터럽트 오브젝트로 분기하도록 함으로써 주기적인 동작을 하도록 설계하였다. 또한 실시간 이식커널의 ISR내에서 지연처리호출을 요청하여 더 높은 우선순위의 인터럽트에 대한 인터럽트 지연시간을 최소화하여 윈도우즈에 실시간성을 지원하도록 설계 및 구현하였다.

1. 실시간 이식커널 타이머 인터럽트의 등록

윈도우즈는 라운드 로빈(Round-Robin) 스케줄러에 의해 스케줄링을 한다. 이로 인해 윈도우즈는 실시간 쓰레드의 마감시간(Dead-Line)을 보장하지 못하므로 실시간성을 지원하지 못한다. 이러한 문제를 해결하기 위해 실시간 이식커널은 Local APIC 타이머를 통해 윈도우즈 독립적인 타이머 인터럽트를 발생시킴으로써 실시간 쓰레드의 주기적인 수행을 가능하게 하였다.



▶▶ 그림 4. 타이머 레지스터

x86하드웨어에서 제공하는 Local APIC는 LVT(Local Vector Table)라는 6개의 레지스터의 집합으로 이루어져 있는데 그 중 Timer 레지스터의 설정으로 주기적인 인터럽트를 발생시킬 수 있다. [그림 4]은 LVT의 Timer레지스터의 모습이다. Local APIC의 타이머 인터럽트가 발생하면 Timer 레지스터의 Vector에 해당하는 IDT의 벡터넘버로 분기하게 된다. IDT에 저장되어 있는 인터럽트의 핸들러 주소를 참조하여 주기적으로 발생되는 타이머 인터럽트의 핸들러를 수행한다[4]. 이러한 인터럽트 처리 과정을 위해 윈도우즈의 IDT에 인터럽트 오브젝트를 등록시켜야 한다.

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

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

▶▶ 그림 5. 인터럽트 오브젝트 등록 함수

[그림 5]와 같이 실시간 이식커널 내에서 윈도우즈가 제공하는 API인 HalGetInterruptVector()함수로 IDT의 벡터넘버를 얻는다. 이 함수의 인자값으로는 하드웨어

인터럽트의 IRQ넘버가 필요하며 윈도우즈는 해당 인터럽트를 처리하기 위한 인터럽트 오브젝트를 등록시킬 수 있는 적절한 벡터를 할당해 준다. 윈도우즈로부터 할당받은 벡터값과 실시간 이식커널이 사용할 ISR 함수를 인자값으로 IoConnectInterrupt()함수를 통해 인터럽트 오브젝트 및 ISR을 등록시켜준다. [그림 6]에서와 같이 HalGetInterruptVector()함수를 통해 윈도우즈로부터 할당받은 IDT의 벡터넘버를 Timer 레지스터의 벡터비트에 설정해줌으로써 Local APIC의 타이머로부터 발생되는 주기적인 인터럽트를 통해 핸들러를 수행시켜준다.

```
CalVector = MappedVector & 0xff;// -0x100
(*apic_timer) = (0x20000 | CalVector);
```

▶▶ 그림 6. Local APIC의 벡터비트 설정

HalGetInterruptVector()함수를 통해 할당받은 벡터넘버는 0xff만큼 큰 값이 반환되기 때문에 실제로 IoConnectInterrupt()함수를 통해 인터럽트 오브젝트를 등록시켜줄 때에는 0x100만큼 뺀 값을 인자값으로 사용해야 한다. 따라서 할당받은 값과 0xff를 비트논리곱(&) 연산해준 값을 벡터넘버로 설정해줌으로써 Local APIC의 타이머 인터럽트가 발생할 때마다 윈도우즈로부터 할당받은 벡터넘버의 ISR로 분기하여 동작하게 된다.

2. 지연처리호출루틴 등록

실시간 이식커널은 주기적인 타이머 인터럽트를 통해 실시간성을 보장할 수 있지만 더 높은 우선순위의 인터럽트 지연시간이 길어질 수 있다. 따라서 본 논문에서는 실시간 이식커널이 윈도우즈가 제공하는 지연처리호출루틴을 사용하도록 설계 및 구현하였다.

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

▶▶ 그림 7. 지연처리호출을 초기화하는 함수

윈도우즈로부터 IDT의 벡터넘버를 할당 받고 인터럽트 오브젝트를 등록하기 전에 지연처리호출 사용을 위한 초기화 작업이 필요하다. [그림 7]에서와 같이 윈도우즈의 I/O 관리자가 제공하는 API인 IoInitializeDpcRequest() 함수는 ISR내에서 지연처리호출을 요청후에 IRQL이 Passive레벨로 떨어지며 Dispatch레벨에서 수행될 함수를 등록해준다.

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

    PDEVICE_OBJECT DeviceObject = (PDEVICE_OBJECT)Context;

    DbgPrint("Interrupt Service Routine !!!\n");
    IoRequestDpc(DeviceObject,
                 DeviceObject->CurrentIrp,
                 InterruptDpcRoutine);

    return 1;
}

VOID InterruptDpcRoutine(IN PKDPC Dpc,
                         PDEVICE_OBJECT DeviceObject,
                         IN PIRP Irp, IN PVOID Context) {

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

}
```

▶▶ 그림 8. ISR과 지연처리호출 처리함수

실시간 확장커널의 ISR에 해당하는 [그림 8]의 InterruptIsr()함수에서 지연처리호출을 요청하기 위해 윈도우즈의 I/O 매니저에서 제공하는 IoRequestDpc()함수를 이용해 지연처리호출을 요청한다. 그 후 IRQL이 떨어지며 Dispatch 레벨이 될 때 지연처리호출의 처리함수인 InterruptDpcRoutine()함수가 실행된다.

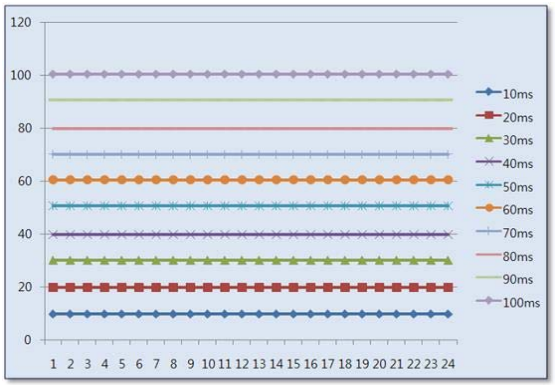
Ⅳ. 실험 환경 및 결과

윈도우즈의 지연처리호출을 이용한 실시간 이식 커널을 구현하기 위해 [표 2]와 같은 실험환경을 구성하였다. 타겟에서 실시간 이식커널을 동작시켰으며 호스트에서는 모니터링을 하였다. 실시간 이식커널은 싱글코어를 기반으로 설계하였기 때문에 타겟의 CPU는 Pentium 4 프로세서를 사용하였다. 실시간 이식커널은 Local APIC 타이머를 이용해 윈도우즈 독립적인 타이머 인터럽트를 주기적으로 발생시킴으로써 윈도우즈에 실시간성을 제공한다.

표 2. 호스트와 타겟 컴퓨터의 실험 환경

	호스트	타겟
CPU	Intel® Core(TM)2 Duo 3GHz	Intel® Pentium 4 3GHz
OS	Windows WP SP3	Windows WP SP3
DDK	WDK 6001.18002	WDK 6001.18002

실시간 이식커널은 윈도우즈의 IDT에 인터럽트 오브젝트와 지연처리호출을 등록줌으로써 타이머 인터럽트 발생시 ISR에서 지연처리호출을 요청하여 지연처리호출 처리함수가 동작을 하는 것을 확인할 수 있었다.



▶▶ 그림 10. 실시간 이식커널의 주기적 동작

또한 실시간 이식커널이 윈도우즈의 스케줄링에 영향을 받지 않고 항상 설정된 주기를 지키며 동작함으로써 윈도우즈에 실시간성을 제공할 수 있음을 확인할 수 있었다. [그림 10]은 10ms 단위로 주기를 변화시킨 실시간 이식커널의 타이머 인터럽트 주기 측정결과를 나타낸다.

Ⅴ. 결론 및 향후연구과제

휴대용 점검장비는 실시간으로 데이터를 획득하고 평가하기 때문에 운영체제로부터의 실시간성 제공이 필수적으로 요구된다. 실시간 이식커널은 디바이스 드라이버 형태로 구현된 확장 커널로써 윈도우즈의 커널자원 및 x86하드웨어의 자원 접근 및 제어가 가능하다. 이를 이용해 Local APIC를 이용한 윈도우즈 독립적인 타이머 인터럽트를 가지며 윈도우즈에 실시간성을 제공한다. 하지만 ISR에서의 처리시간이 길어지면 더 높은 우선순위를 가진 인터럽트의 지연시간이 길어질 수 있는 문제가 있다. 본 논문에서는 윈도우즈가 제공하는 지연처리호출을 사용하여 ISR의 작업을 하드웨어 인터럽트 레벨보다 낮은 Dispatch레벨에서 처리함으로써 높은 우선순위의 인터럽트를 막지 않도록 하고, 인터럽트 지연시간을 줄였다. 이를 위해 윈도우즈의 I/O 관리자와 커널이 제공하는 API를 이용해 IDT에 인터럽트 오브젝트를 등록과 지연처리호출을 사용하도록 하였고, Local APIC의 타이머 인터럽트로 인한 지연처리호출의 주기적인 동작을 가능하게 함으로써 윈도우즈에 실시간성을 지원하도록 하였다.

향후 연구과제로는 다수의 실시간 쓰레드가 필요할 경우 실시간 이식커널의 실시간 쓰레드간의 우선순위 스케줄러를 구현함으로써 다수의 실시간 쓰레드의 동작을 보장해야 할 것이다. 또한 다른 주기를 가진 다수의 실시간 쓰레드의 동작 시 각각의 실시간 쓰레드가 지연처리호출을 사용함으로써 인터럽트 지연시간을 줄이고 실시간성을 제공해야 할 것이다.

▌참 고 문 헌▐

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

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

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

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

[5] 이봉석, 윈도우 디바이스 드라이버, 한빛미디어, 2009.