DOI : 10.5392/JKCA.2011.11.7.051

윈도우 유저 레벨 로봇 컴포넌트에 실시간성 지원 방법

A Method to Support Real-time for User-level Robot Components on Windows

주민규, 이진욱, 장철수*, 김성훈*, 이철훈
충남대학교 컴퓨터공학과, 한국전자통신연구원*

Min-Gyu Ju(mingyuman@cnu.ac.kr), Jin-Wook Lee(ljweve@cnu.ac.kr),
Choul-Soo Jang(jangcs@etri.re.kr)*, Sung-Hoon Kim(saint@etri.re.kr)*,
Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

미래 시장을 선도할 핵심 분야로 지능형 서비스 로봇을 꼽을 수 있으며, 지능형 서비스 로봇은 인간과 공존하면서 육체적, 정신적, 감성적으로 인간을 보조하는 로봇이다. 지능형 서비스 로봇은 인간과 밀접한 관계를 맺으며 동작하기 때문에 지능형 서비스 로봇이 제공하는 핵심 서비스의 안정적인 수행은 로봇 사용자의 안전을 보장하기 위한 필수적인 고려사항이다. 이러한 안전성을 위해서는 정해진 시간마다 주기적으로 핵심 서비스를 수행시키는 실시간성이 필수 불가결한 요소이다. 현재 많은 로봇 컴포넌트들이 개발의 편의성을 위해 범용 운영체제인 윈도우를 사용하지만, 윈도우는 실시간성을 지원하지 않는 문제점이 있으며 실시간성 제공을 위해 RTX나 INtime과 같은 고가의 써드파티를 별도로 설치하여 사용해야 한다. 또한 로봇 컴포넌트는 유저 레벨에서 동작하기 때문에 유저영역에서 실시간성을 제공할 수 있는 방법에 대한 연구가 필요하다. 본 논문에서는 범용 운영체제 윈도우의 커널 레벨에서 실시간성을 제공하는 RTiK을 이용하여 유저 레벨에서 동작하는 함수를 주기적으로 동작시키는 방법을 설계 및 구현하였다.

■ 중심어 : | 실시간성 | RTiK | 로봇 컴포넌트 | 유저 레벨 |

Abstract

Intelligent service robots leading the future market are robots which assist humans physically, mentally, and emotionally. Since intelligent service robots operate in a tightly coupled manner with humans, their safe operation should be an inevitable consideration. For this safety, real-time capabilities are necessary to execute certain services periodically. Currently, most robot components are being developed based on Windows for the sake of development convenience. However, since Windows does not support real-time, there is no option but to use expensive third-party software such as RTX and INTime. Also since most robot components are usually execute in user-level, we need to research how to support real-time in user-level. In this paper, we design and implement how to support real-time for components running in user-level on Windows using RTiK which actually supports real-time in kernel level on Windows.

■ keyword : | Real-Time | RTiK | Robot Components | User-Level |

* 본 연구는 지식경제부 및 정보통신 연구진흥원의 IT성장 동력기술 개발사업의 일환으로 수행 되었습니다.

접수번호 : #110308-007
접수일자 : 2011년 03월 08일
심사완료일 : 2011년 06월 21일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

I. 서 론

IT기술의 발달과 인간중심의 컴퓨팅 패러다임으로 인간 삶의 질을 향상시킬 수 있는 기술에 대한 연구가 활발히 이루어지고 있다. 특히 과거 수동적으로 동작하는 로봇에서, 스스로 주위 환경을 인식하고 판단하는 지능형 로봇이 등장하였다. 지능형 로봇은 간호 로봇, 청소 로봇, 교육용 로봇, 길안내 로봇 등이 있으며, 이러한 로봇들은 인간과 밀접한 관계를 맺으며 동작하기 때문에 언제 일어날지 모르는 상황에 대해 빠른 대처를 해야 한다[1]. 하지만, 현재 로봇 컴포넌트들은 개발의 편리성을 위해 범용 운영체제인 윈도우의 유저 레벨에서 개발하는 추세이며, 윈도우는 굶주림 현상(Starvation)을 방지하기 위해 런 타임 우선순위 값을 두어 스케줄링을 하므로 실시간성을 지원하지 못한다. 윈도우의 실시간성 지원을 위해서 써드파티 운영체제를 별도로 설치하여 사용해야 하며, 써드파티(Third-Party) 운영체제를 사용할 경우 고가의 구입비 및 유지 보수비 등을 지불해야 하는 문제점이 있다. 그리고 기존 RTiK(Real-Time implanted Kernel)의 경우 윈도우의 커널 레벨에서만 실시간성을 지원하는 문제점 때문에 유저 레벨의 로봇 컴포넌트에게 실시간성을 지원할 수 없다[13].

본 논문에서는 x86기반 윈도우의 커널 레벨에서만 실시간성을 지원하는 기존 RTiK의 내부를 수정하여, 윈도우 유저 레벨에서 사용자가 정의한 함수를 동작시키는 방법을 설계 및 구현 하였다. 이를 통해 윈도우 유저 레벨에서 동작하는 로봇 컴포넌트 쓰레드에 실시간성을 지원함으로써 정확한 주기에 로봇 컴포넌트가 동작하게 하여, 기존의 값 비싼 써드파티 운영체제를 대신하여 로봇 미들웨어에게 실시간성을 지원하는 방법을 연구하였다.

본 논문의 구성은 2장에서 관련연구로 써드파티 운영체제와 현재 상용화된 로봇 미들웨어와 윈도우 스케줄링, 기존 RTiK에 대해 기술하며, 3장에서 RTiK 내부코드를 수정하여, 사용자 함수를 주기적으로 수행 시키는 이벤트 기반 RTiK을 설계 및 구현하였다. 4장에서는 실험 및 결과를 기술 하였으며, 마지막으로 결론 및 향후 연구과제에 대해 기술하였다.

II. 관련 연구

현재 상용중인 써드파티 운영체제인 RTX와 INtime에 대해 기술하며, 범용 운영체제인 윈도우의 스케줄링 기법 및 RTiK에 대해 기술한다.

1. 써드파티 운영체제

1.1 RTX

IntervalZero의 RTX(Real-Time Extension)는 윈도우 XP 또는 윈도우 2000에 고성능 실시간 제어가 가능한 실시간 운영체제의 기능을 부가해주는 확장 소프트웨어로서, 대부분의 개발자들이 익숙하고 편리한 윈도우 환경에서 RTX를 이용하여 기존의 실시간 운영체제의 장점을 최대한 사용할 수 있도록 제공해 준다. 즉, RTX는 순수 실시간 운영체제가 아니라 윈도우의 대중성 및 풍부한 GUI Library의 장점을 최대한 이용하며 실시간성 등의 제약사항들을 보완해 주는 소프트웨어이다. 국방, 항공, 계측기, 시뮬레이션, 의료 및 로보틱스 등의 많은 산업분야에서 사용되고 있다[2].

1.2 INtime

인텔사가 개발한 iRMX커널을 사용하고 있는 인텔의 x86/CPU에 특화된 리얼타임 시스템 소프트웨어이다. INtime은 윈도우 XP/윈도우2000을 플랫폼으로 한 제어·계측 시스템에 INtime을 부가해 시스템상의 실시간이 필요한 처리를 INtime이 처리하게 하여, 윈도우의 유연성은 그대로 유지하면서 보다 높은 신뢰성과 정밀한 리얼타임 퍼포먼스를 보충할 수 있다.

INtime은 산업용 컴퓨터(IPC)나, x86 보드 컴퓨터, 범용 PC로, 윈도우(2000/XP/vista)와 협조 동작하는 실시간 운영체제이며, 동일 하드웨어 상에서 윈도우와 동시에 동작하는 실시간 운영체제이다. INtime이 설치 된 윈도우는 2개의 커널이 동작하는 멀티 커널 시스템이 된다. 즉, 2개의 가상 CPU를 사용하는 형태로 동작하게 된다[3].

2. 로봇 미들웨어

로봇 미들웨어는 로봇의 하드웨어가 PDA, 컴퓨터와 같은 다른 종류의 장치들과 통신 및 장치들을 제어하기 위한 프로그램이다. 로봇 미들웨어는 기계어로, 작동되는 로봇을 통제하며, 기계어 혹은 스크립트 형식의 명령을 받아 자신의 상태 및 작업 결과를 반환해주는 프로그램이라 할 수 있다. 프로그램 계층은 개발자로 하여금 최상위 계층 언어만을 사용해서 프로그래밍 할 수 있게 해준다. 현재 상용화된 외국의 로봇 미들웨어로는 OROCOS(Open Robot Control Software), MSRS(MicroSoft Robotic Studio), MIRO(Mi cro-Middleware for Mobile Robot Application)등이 있으며, 국내에서는 OPRoS(Open Platform for Robotic Service)라는 로봇미들웨어가 개발 중이다. 이러한 미들웨어는 실시간 운영체제 상에서만 실시간성을 지원하는 문제점이 있다[4-9].

3. 윈도우 스케줄링 기법

윈도우는 특정 임무를 수행하기 위해 프로세스와 쓰레드가 존재한다. 쓰레드는 프로세스에 포함되어 코드(Code), 데이터(Data) 영역을 공유하며 자신만의 스택(Stack)영역을 가지고 맡은 임무를 수행한다. 윈도우에서는 쓰레드가 커널 스케줄러(Schedule)에 의해 CPU를 할당받아 수행되며, 시스템 콜 사용 시 블로킹(Blocking)되지 않는다. [그림 1]과 같이, 각 우선순위를 가진 레디 큐에 있는 쓰레드들을 라운드 로빈 방식으로 스케줄링 한다.

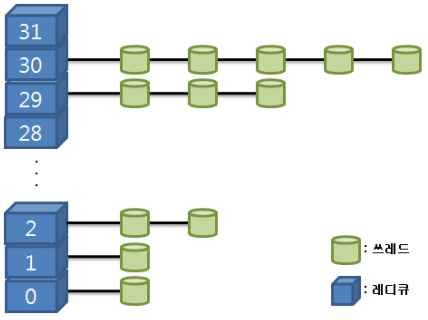


그림 1. 윈도우의 쓰레드 스케줄링

[그림 1]은 윈도우의 쓰레드 스케줄링을 나타낸 그림이다. 타원형의 그림은 준비(Ready) 상태에 있는 쓰레드들이고, 사각형의 그림은 레디 큐를 나타낸 것이며, 사각형안의 숫자는 각 레디 큐의 우선순위를 나타낸 것이다. 31번의 레디 큐에 있는 쓰레드들이 가장 우선순위가 높고 0번 레디 큐에 있는 쓰레드 들이 가장 우선순위가 낮다. 29번 레디 큐에 있는 쓰레드가 수행되기 위해서는 30번 레디 큐에 있는 쓰레드들이 모두 처리될 때까지 기다려야 하며, 30번에 있는 동일 우선순위 쓰레드들은 라운드로빈 스케줄 방식으로 처리가 된다.

윈도우는 프로세스 우선순위 값이 하나의 값으로 결정되어 있는 반면 쓰레드의 우선순위 값은 런 타임 우선순위 값과 베이스 우선순위 값으로 구성되어 진다. 베이스 우선순위 값은 쓰레드 생성시 기본 값인 ABOVE_NORMAL_PRIORITY_CLASS중 THREAD_PRIORITY_NORMAL로 결정된다. ABOVE_NORMAL_PRIORITY_CLASS는 윈도우 우선순위 중 15에서 0까지를 가질 수 있으며, THREAD_PRIORITY_NORMAL는 그 중 7의 우선순위를 나타낸다. 이러한 윈도우 우선순위 값은 API를 사용하여 유저 레벨의 사용자가 결정할 수 있다.

런 타임 우선순위 값은 베이스의 우선순위 값에 기준하여 이보다 높은 우선순위 값으로 시스템에 의해 임시로 바뀌는 값으로 스케줄러에 의해 실제로 적용되어지는 우선순위 값이다. 런 타임 우선순위 값은 15부터 0까지의 우선순위 쓰레드들에게만 일어난다. 쓰레드에서 이와 같은 런 타임 우선순위를 따로 가지는 이유는 대기 중에 있는 쓰레드에 대한 보다 빠른 응답과 우선순위 스케줄링에서 나타날 수 있는 낮은 우선순위 쓰레드의 CPU 굶주림 현상(Starvation)을 방지하기 위해서이다[10-12].

4. RTiK

[그림 2]에서와 같이 RTiK은 윈도우 디바이스 드라이버의 형태로 이식된다. 이는 RTiK이 윈도우의 커널 레벨에서 동작하게 함으로써 하드웨어의 접근이나 윈도우 커널 자원의 접근을 가능하게 한다. RTiK의 하드웨어 추상화 계층(Hardware Abstraction Layer)을 통

해 x86 기반의 하드웨어 플랫폼에 접근하여 Local APIC(Advanced Programmable Interrupt Controller)를 제어할 수 있으며, Local APIC의 타이머 레지스터를 통해 윈도우와 독립적인 타이머 인터럽트를 발생시킨다. 또한 RTiK 쓰레드는 지연처리 호출(DPC: Deferred procedure call)을 통하여 RTiK의 타이머 인터럽트 서비스 루틴이 실행된 직후에 CPU 권한을 받아 수행하게 된다[13].

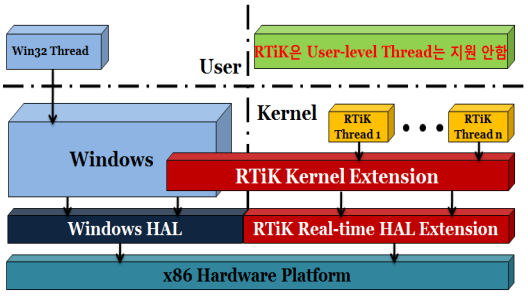


그림 2. 기존 RTiK의 구조

III. 윈도우 유저 레벨 로봇 컴포넌트에 실시간성 지원 방법 설계 및 구현

1. 이벤트 기반 RTiK의 설계 및 구현

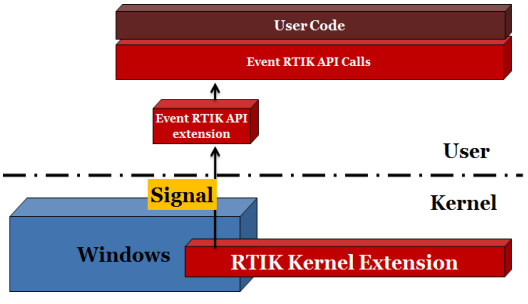


그림 3. 이벤트 기반 RTiK의 구조

RTiK을 이용한 윈도우 유저 레벨에서의 실시간성 지원 방식의 설계는 [그림 3]과 같다. 사용자는 이벤트 기반 RTiK으로부터 제공되는 API를 이용하여 실시간 쓰레드의 수행코드를 작성하고, 생성된 실시간 쓰레드는 커널 영역으로부터 전달 될 신호를 기다린다. 이벤트 기반 RTiK의 타이머가 활성화 되면 사용자가 설정한 주기로 타이머 인터럽트가 발생하며, 지연 처리 호출과정에서 유저영역에 신호를 전달해 주게 된다. 유저 레벨의 쓰레드는 주기적인 신호를 받을 때 마다 자신의 루틴을 수행하게 된다. 이때 유저 레벨의 실시간 쓰레드는 윈도우에서 관리하는 프로세스 및 쓰레드의 우선순위 중 REALTIME_PRIORITY _CLASS의 PRIORITY_TIME_CRITICAL의 단계로서 가장 높은 우선순위로 동작한다. 따라서 윈도우에서 동작하는 다른 프로세스 및 쓰레드의 영향을 받지 않고 항상 설정된 주기를 지키며 동작함으로써 유저영역에 실시간성을 제공해 줄 수 있다[14-22].

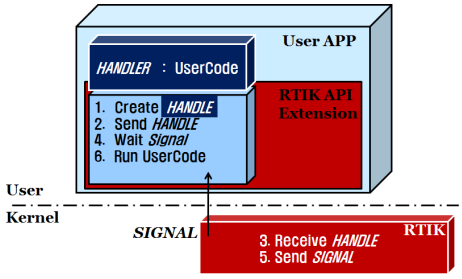


그림 4. 이벤트 기반 RTiK의 동작과정

이벤트 기반 RTiK의 동작과정은 [그림 4]와 같다. 유저 레벨에서 Handle을 생성하고, 유저 레벨과 커널 레벨의 통신수단인 IOCTL 코드를 통하여 생성된 Handle을 커널 레벨에 넘겨주게 된다. 이때 생성한 Handle은 커널영역에서 동작하는 RTiK과 유저 레벨간의 동기화 수단으로 사용된다. 유저 레벨의 실시간 쓰레드는 커널 레벨의 RTiK으로부터 신호를 기다리다가, 신호를 받으면 해당 쓰레드가 깨어나 유저 레벨에 작성된 유저코드 부분을 수행한다. 커널영역에서는 유저 레벨로부터 전달받은 공유 이벤트들을 리스트로 연결하여 관리한다. 사용자가 설정한 주기에 따라 타이머 인터럽트가 활성화 되면, 해당 인터럽트 서비스 루틴을 완료 후, DPC메커니즘에서 handle의 이벤트에 신호를 보내 주게 된다.

또한, 실시간 쓰레드는 윈도우에서 동작하고 있는 유저 레벨의 다른 쓰레드들보다 먼저 수행되어야 하기 때문에 쓰레드의 우선순위를 변경하였다.

표 1. 프로세스 우선순위 API

우선순위	프로세스 우선순위
실시간	REALTIME_PRIORITY_CLASS
높은	HIGHT_PRIORITY_CLASS
보통초과	ABOVE_NORMAL_PRIORITY_CLASS
보통	NORMAL_PRIORITY_CLASS
보통미만	BELOW_NORMAL_PRIORITY_CLASS
유휴	IDLE_PRIORITY_CLASS

[표 1]과 같이 윈도우의 프로세스는 총 6개의 우선순위 그룹을 가질 수 있으며, 생성된 프로세스의 우선순위는 하나의 값으로 결정된다.

로봇 컴포넌트의 프로세스를 생성할 때 가장 높은 우선순위인 REALTIME_PRIORITY_CLASS 우선순위 단계로 설정하여 프로세스를 생성해야 한다.

REALTIME_PRIORITY_CLASS 우선순위는 31부터 16까지의 쓰레드 우선순위 값을 가진다. 윈도우의 런 타임 우선순위 변경은 15부터 0까지만 일어나므로 다른 쓰레드들에 의해 CPU를 선점 받지 않는다[10-12].

표 2. 쓰레드 우선순위

우선순위	쓰레드 우선순위
시간우선	THREAD_PRIORITY_TIME_CRITICAL
아주높음	THREAD_PRIORITY_HIGHEST
보통초과	THREAD_PRIORITY_ABOVE_NORMAL
보통	THREAD_PRIORITY_NORMAL
보통미만	THREAD_PRIORITY_BELOW_NORMAL
아주낮음	THREAD_PRIORITY_LOWEST
유휴	THREAD_PRIORITY_IDLE

[표 2]에서와 같이 윈도우가 제공하는 가장 높은 우선순위, THREAD_PRIORITY_TIME_CRITICAL 로 쓰레드들의 우선순위를 높여주어 윈도우에서 동작하고 있는 다른 쓰레드들보다 우선순위를 가장 높게 설정해 주었다.

2. 유저 레벨에서 이벤트 기반 RTiK 타이머 제어 API

커널 레벨에서 동작하는 이벤트 기반 RTiK 타이머를 유저 레벨에서 쉽게 접근할 수 있게 하기 위하여 이벤트 기반 RTiK 내부의 DeviceIOControl루틴에서 특정한 루틴을 작성하고 유저 레벨에서 IOCTL 코드를 통하여 특정한 루틴을 호출하는 방식으로 API를 구현하였다[23-26].

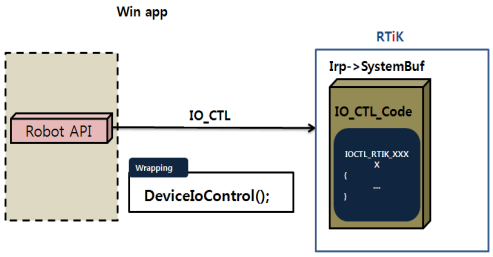


그림 5. IOCTL을 이용한 API 설계

표 3. RTiK 타이머 제어 API

프로토 타입	함수기능
Int RTIK_Resolution (int Resolution)	RTIK 타이머의 기본주기를 설정하는 함수
Int RTIK_DisableWithDefault Resolution(void)	타이머의 기본주기를 1000ms로 설정하고 Disable 시키는 함수
Int RTIK_CreateThread (void *Function)(void) ,int times)	사용자가 정의한 함수를 (기본주기 * times)의 주기로 수행 시키는 함수
Int RTIK_Enable(void)	타이머를 활성화 시키는 함수
Int RTIK_Disable(void)	타이머를 비활성화 시키는 함수
Int RTIK_GetInformation (void)	타이머의 상태 정보를 얻어 오는 함수
Int RTIK_DeleteAllThread (void)	생성된 실시간 쓰레드를 모두 삭제하는 함수

[그림 5]와 같이 이벤트 기반 RTiK API는 RTiK의 미리 정의된 IOCTL코드를 래퍼(Warpper)하는 형태로 구성되어 있으며, 사용자가 이벤트 기반 RTiK의 내부 코드를 모르더라도 제공되는 API를 통해 RTiK의 타이머 주기와 실시간 쓰레드를 생성할 수 있게 하여 로봇 컴포넌트에게 실시간성을 지원할 수 있도록 하였다. 사용자에게 제공되는 API는 [표 3]과 같다.

IV. 실험 및 결과

1. 실험 환경 및 실험 방법

그림 6. 실험환경

이벤트 기반 RTiK이 윈도우 운영체제에 정상적으로 이식되어 유저 레벨의 실시간 쓰레드가 주기적으로 동작함을 검증하기 위해 구성한 실험 환경은 [그림 6]과 같다. 호스트 컴퓨터와 타겟 컴퓨터에 실시간 이식 커널을 이식하고, 정상적인 동작을 확인하기 위해 [표 4]와 같은 환경의 호스트 컴퓨터와 타겟 컴퓨터를 시리얼 포트로 연결하여 이벤트 기반 RTiK을 디버깅 및 모니터링 하였다.

표 4. 호스트와 타겟 컴퓨터 환경

실험 환경		
	HOST	TARGET
CPU	Intel Core2Duo 3.00GHz	Intel Pentium M Processor 1.80GHz
Operating System	Windows XP SP3	Windows XP SP3
DDK	WDK 6001.18002	WDK 6001.18002
Development Tool	MS Visual Studio 2005	WDK Checked Builder

실험 방법은 유저 레벨에서 1ms의 주기를 가지는 실시간 쓰레드를 생성하여, 주기별 시간을 측정하였다. 또한 현재 로봇 컴포넌트를 실행시키는 실행엔진이 사용하는 큐 타이머와의 비교를 위해, 동일한 조건에서 큐 타이머를 실험하였다. 이때 윈도우에서 동작하고 있는 다른 쓰레드가 있을 경우에 대한 시간을 측정하기 위하여 무한으로 while문을 수행하는 프로세스를 생성하여 시간을 측정하였다. 실험결과의 정확성을 위해 매 주기마다 I/O포트에 신호를 출력하며 오실로스코프를 이용해 측정하였다.

2. 실험 결과

아래의 [그림 7][그림 8]은 이벤트 기반 RTiK의 주기를 1ms로 설정하고, 워크로드가 없을 때와 10개의 워크로드를 동시에 수행시키며 매주기마다 유저 레벨에서 I/O포트에 출력한 신호를 측정한 결과이다.

오실로스코프 한 구간은 1ms을 의미 하며, [그림 7]의 파형은 이벤트 기반 RTiK이 1ms 주기마다 유저 레벨에서 I/O포트에 출력하는 모습을 나타낸다. 이는 유저 레벨에서 동작하는 로봇 컴포넌트가 주기적으로 수행 시킬 수 있음을 의미한다.

그림 7. 이벤트 기반 RTiK 실험결과

그림 8. 이벤트 기반 RTiK & workload 10 실험결과

[그림 8]은 윈도우의 다른 쓰레드가 있을 경우에도 이벤트 기반 RTiK이 1ms 주기를 지키며 동작하는지에 대한 실험 결과이다. 실험방법은 [그림 7]과 동일하며, while문만을 수행하는 프로세스 10개를 수행시키며, I/O 포트에 출력하는 모습을 나타낸 것이다.

[그림 8]은 이벤트 기반 RTiK은 윈도우의 다른 프로세스의 영향을 받지 않으며, [그림 7]에서와 같이 유저 레벨에서 1ms 마다 I/O 포트에 정상 출력한다.

[표 5]는 [그림 7][그림 8]의 실시간 쓰레드 주기의 최대, 최소값을 나타낸 표이다.

표 5. 이벤트 기반 RTiK 1ms 최대, 최소값

주기	로드 갯수	최대	최소
1ms	로드 없음	1.00626ms	0.99463ms
	로드 1개	1.00901ms	0.99138ms
	로드 10개	1.01001ms	0.99026ms

[그림 9][그림 10]은 윈도우가 제공하는 큐 타이머와의 비교를 위해, 동일한 조건에서 이벤트 기반 RTiK과 큐 타이머를 15ms 주기로 설정하고 실험한 결과화면이다. 주기적으로 동작하는 [그림 9]의 이벤트 기반 RTiK과는 달리 [그림 10]의 큐 타이머는 워크로드가 많을수록 주기를 크게 벗어나는 모습을 보인다. 이와 같은 큐 타이머는 윈도우의 다른 프로세스 및 쓰레드가 늘어나면 실시간성을 제공하지 못하는 것을 의미한다.

그림 9. 이벤트 기반 RTiK 15ms & while 10 실험결과

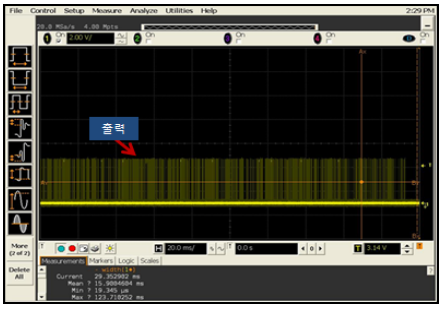


그림 10. 큐타이머 15ms & while 10 실험결과

[표 6]은 이벤트 기반 RTiK과 큐 타이머에 15ms주기를 설정하고 워크로드를 각각 0, 1, 10개 일때, 실시간 쓰레드와 큐타이머의 최대, 최소 주기를 나타낸 표이다.

표 6. 이벤트 기반 RTiK 타이머와 큐 타이머의 측정결과

주기	로드 갯수	이벤트 RTiK	큐타이머
15ms	로드 없음	15.0105ms	16.7583ms
	로드 1개	15.0135ms	97.8642ms
	로드 10개	15.0155ms	123.710ms

또한, 기존 써드파티 운영체제인 RTX와의 성능비교를 위하여 RTX와 이벤트 기반 RTiK을 비교하였다. [표 7]은 이벤트 기반 RTiK과 RTX를 위의 실험환경과 동일한 조건 실험하여 비교한 표이다. 각각 1ms주기로 설정 하고 오실로스코프를 통하여 결과를 측정하였다.

표 7. 이벤트 기반 RTiK 타이머와 RTX 타이머의 측정결과

주기	로드 갯수	이벤트 RTiK	RTX
15ms	로드 없음	1.00626ms	1.00722ms
	로드 1개	1.00901ms	1.01333ms
	로드 10개	1.01001ms	1.09223ms

[표 7]의 결과를 통하여 유저 레벨에서 동작하는 이벤트 기반 RTiK과 RTX는 성능이 비슷하다는 것을 알 수 있다. 이는 RTX와 이벤트 기반 RTiK모두 윈도우 유저 레벨에서 동작하는 로봇컴포넌트에게 실시간성을 지원함을 의미한다.

IV. 결론 및 향후 연구과제

로봇 미들웨어에서 구현되는 로봇 컴포넌트는 운영체제의 쓰레드에 의해 실행된다. 로봇의 특성상 외부로부터 발생한 비동기적인 이벤트에 대해 로봇 시스템이 정해진 시간 안에 응답해야 하기 때문에 실시간성이 필수적으로 요구된다. 또한 로봇 컴포넌트를 수행하는 쓰레드는 유저 레벨에서 동작하기 때문에 유저 레벨에서 실시간을 보장하며 동작할 수 있어야 한다.

본 논문에서는 현재 상용화되어 사용되고 있는 로봇들과의 호환성을 유지하면서, 유저 레벨에서 동작하는 로봇 컴포넌트에 실시간성을 지원하기 위해 커널 레벨에서만 동작하는 RTiK을 이용하여 윈도우 유저 레벨에서 동작하는 함수를 호출 하는 방법을 설계 및 구현하였다.

RTiK은 디바이스 드라이버 형태로 구현되어, 윈도우의 커널 자원은 물론 x86 하드웨어의 자원 접근 및 제어가 가능하다. 하지만, 윈도우 커널 레벨의 쓰레드만 접근 가능하다는 문제점이 있다. 이러한 문제점을 해결하기 위해 윈도우 커널 레벨에서만 동작하는 RTiK을 윈도우 유저 레벨의 로봇 컴포넌트를 실행하는 실시간 쓰레드에게 신호를 보내도록 수정하여, 유저 레벨에서 실시간 쓰레드가 주기적인 동작을 할 수 있게 하였다. 또한 실시간 쓰레드의 우선순위를 높여, 다른 윈도우 쓰레드 보다 먼저 수행시킴으로써 유저 레벨에서 로봇 컴포넌트에게 실시간성을 지원할 수 있다.

향후 연구과제로는 이벤트 기반 RTiK이 사용할 수 있는 여러 가지 스케줄링 기법에 대한 연구가 필요하다.

참 고 문 헌

[1] D. Yu, H. Lin, R. Guo, J. Yang, and P. Xiao, "Research on real-time middleware for open architecture controller," Embedded and Real-Time Computing Systems and Applications, 2005. Proceddings. 11th IEEE International Conference on, pp.80-83, 2005(8).

[2] http://www.intervalzero.com

[3] http://www.tenasys.com

[4] http://www.orocos.org

[5] H. Bruyninckx, P. Soetens, and B. Koninckx, "The Real-Time Motion Control Core of the Orocos Project," IEEE International Conference on Robotics & Automation, Vol.2, pp. 2766-2771, 2003.

[6] H. Bruyninckx, "Open Robot Control Software: the OROCOS project," IEEE International Conference on Robotics & Automation, Vol.3, pp.2523-2528, 2001.

[7] http://www.microsoft.com/korea/robotics/default.mspx

[8] http://msdn.microsoft.com/en-us/library/cc998480.aspx

[9] http://miro-middleware.berlios.de/

[10] 장덕영, *Windows 구조와 원리,* 한빛미디어, 2009.

[11] JOHNSON M. HART, Windows 시스템 프로그래밍 3판, 정보문화사, 2008.

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

[13] 이진욱, 조문행, 김종진, 조한무, 박영수, 이철훈, "윈도우 기반의 점검장비에 실시간성을 지원하는 실시간 이식 커널의 설계 및 구현", 한국콘텐츠학회논문지, 제10권, 제10호, pp.36-44, 2010(10).

[14] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual Vol.1 : Basic Architecture, Intel, 2009(9).

[15] Intel, Intel I/O Controller Hub 6(ICH6) Family Datasheet, Intel, 2005(1).

[16] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual Vol.3 : System Programming Guide, Intel, 2009(9).

[17] Intel, Intel 64 Architectures x2APIC Specification, Intel, Intel 2008(6).

[18] Intel, Intel 64 and IA-32 Architectures Software Developer's Manual Vol.2 : Instruction Set Reference, Intel, 2009(9).

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

[20] 이봉석, IT Expert 윈도우 디바이스 드라이버, 한빛미디어, 2009.

[21] http://technet.microsoft.com/en-us/sysinternals

[22] http://www.ntkernel.com
[23] http://www.osronline.com
[24] Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals 4th Edition, Microsoft Press, 2006.
[25] Johnson M.Hart, Windows System Programming 3rd Edition, Addison Wesley, 2005.
[26] John Robbins, DEBUGGING APPLICA TIONS For Microsoft .NET And Microsoft WINDOWS, Microsoft, 2003.

저 자 소 개

주 민 규(Min-Gyu Ju) 준회원

- 2010년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2010년 2월 ~ 현재 : 충남대학교 컴퓨터 공학과(공학석사과정)

<관심분야> : 실시간 시스템, 임베디드 시스템, 운영체제, 로봇

이 진 욱(Jin-Wook Lee) 준회원

- 2009년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2009년 2월 ~ 현재 : 충남대학교 컴퓨터 공학과(공학석사과정)

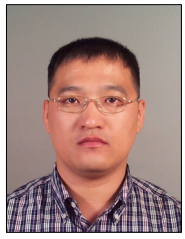
<관심분야> : 실시간시스템, 운영체제, 고장허용 컴퓨팅, 로봇

장 철 수(Choul-Soo Jang) 정회원

- 1995년 2월 : 인하대학교 전자계산공학과(공학사)
- 1997년 2월 : 광주과학기술원 정보통신공학과(공학석사)
- 1997년 1월 ~ 현재 : 한국전자통신연구원 선임연구원
- 2006년 9월 ~ 현재 : 충남대학교 컴퓨터공학과 (박사과정)

<관심분야> : 로봇 소프트웨어 플랫폼, 실시간 시스템,임베디드 시스템

김 성 훈(Sung-Hoon Kim) 정회원



- 1995년 2월 : 광운대학교 전자공학과(공학사)
- 1997년 2월 : 광운대학교 전자공학과(공학석사)
- 1997년 1월 ~ 현재 : 한국전자통신연구원 책임연구원
- 2005년 9월 ~ 현재 : 한양대학교 전자컴퓨터공학 (박사과정)

<관심분야> : 로봇 지능, 로봇 소프트웨어 공학, 군사용 로봇

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 객원 연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙 연구원

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