http://dx.doi.org/10.5392/JKCA.2011.11.9.045

RTiK-Linux: 리눅스용 실시간 이식 커널의 설계

RTiK-Linux: The Design of Real-Time implemented Kernel for Linux

김주만*, 송창인**, 이철훈**
부산대학교 IT응용공학과*, 충남대학교 컴퓨터공학과**

Joo-Man Kim(joomkim@pusan.ac.kr)*, Chang-In Song(ckddls1234@cnu.ac.kr)**,
Cheol-Hoon Lee(clee@cnu.ac.kr)**

요약

첨단 군사 체계를 위한 측정 장치의 필요성에 따라 낮은 지연을 추구하는 실시간 특성인 시간 결정성과 수행의 정확성은 매우 중요해 졌으며, 이러한 이유로 리눅스와 같은 범용 운영체제에 실시간 기능을 추가하는 시장 요구가 증대하게 되었다. 따라서 RTLinux와 RTAI가 리눅스기반의 이중 커널로 개발되었다. RT-Linux의 경우 경성 실시간성을 제공하지만 어셈블러를 사용해야함으로 개발자가 다루기 어려운 단점이 존재한다. 또한 RTAI의 경우 연성 실시간성만을 제공하는 단점이 있다. 이러한 단점을 해결하기위해 RTiK-Linux를 개발하였다. 본 논문에서는 리눅스에 경성 실시간 특징을 가지며 새로운 이중 커널 구조인 실시간 이식 커널인 RTiK-Linux를 제안한다. 먼저 관련 연구에 대한 소개를 하고, 타이머 설정 값과 거의 일치하는 분해능을 보장하는 설계 방법론을 기술한다. 그리고 경험적 측정으로 얻어진 결과를 보이고, 제안하는 RTiK-Linux를 검증 및 평가하기 위하여 그 결과를 분석한다.

■ 중심어 : | 실시간 운영체제 | 타이머 | 리눅스 | 인터럽트 지연 | 분해능 |

Abstract

According to the necessity of measuring equipments for advanced military systems, real-time characteristics such as time determinism and execution accuracy pursuing low-latencies have become very important. With this reason, the market demand for real-time features in the general purpose operating system such as Linux has been enlarging. To meet these requirements, RTLinux and RTAI has been developed as dual-kernels based on Linux. However, developers should use assembler languages to facilitate real-time in RT-Linux, it is very difficult to deal with it. RTAI has disadvantage that it only provides soft real-time. To solve these problems, RTiK-Linux was developed. In this paper, we propose a new dual-kernel with hard real-time capabilities in Linux, called RTiK-Linux(Real-Time implemented Kernel for Linux). We first introduce related researches and then describe the design methodologies to guarantee the resolution which almost accords with the timer settings. Finally, we present the results of experimental measurements and analyze them in order to validate and evaluate the proposed RTiK-Linux.

■ keyword : | Real-Time OS | Timer | Linux | Interrupt Latency | Resolution |

I. 서 론

최근 군사용 장비의 첨단화로 다양한 환경에서의 데이터 수집의 정확성과 시간 기반의 실시간성을 요구하고 있다[1]. 즉, 데이터 수집과 분석과 처리 과정이 임의의 정해진 만료시간을 반드시 지키도록 요구하고 있으

* 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.2011-0007310)
접수번호 : #110707-004
접수일자 : 2011년 07월 07일
심사완료일 : 2011년 09월 07일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

며, 그러한 요구조건에 따라 오늘날 휴대형 장비는 실시간성을 제공하기 위해 VxWorks와 같은 실시간 운영체제를 사용하거나, 윈도우 기반의 RTX, INtime와 같은 상용 솔루션 운영체제를 사용하고 있다. 이들 시스템은 고가의 구입비용과 경상 기술료의 문제로 제품화하기에 걸림돌이 되고 있다. 따라서 개발 비용을 절감할 목적과 개발 환경의 다양성을 위한 윈도우시스템에서의 실시간 지원 점검 장비를 위한 연구가 진행되었으며[1-2], 고 신뢰성을 요구하는 유도 무기 발사 통제 시스템과 같이 정확한 데이터 수집과 예측을 요구하는 실시간 시스템의 필요성이 대두되었다[3].

실시간 시스템이란 제한된 응답시간이나 오동작을 포함한 결과의 위험성에 관한 명확한 규정을 만족하는 시스템으로 정의되며, 또한 외부의 사건에 의하여 임의의 연산이 시작된 이후 그 연산의 결과가 주어진 연산 시간이나 외부의 특정 기준 시간에 의존하는 시스템으로 정의한다. 실시간 시스템은 시간적 제한 조건을 만족시키지 못한 경우, 실시간 시스템의 사용 특성에 따라 작게는 시스템의 잘못된 동작, 크게는 큰 위험에 직면하는 사건을 유발하게 된다. 이와 같이 시간 제약을 반드시 지키기 위한 시스템적 요구 조건으로 시간 결정성과 실행 예측성이 보장되어야 할 것이다[5].

이러한 실시간 기능은 기존의 Linux와 같은 범용 운영체제의 추가적 기능으로 많이 개발하게 되었다. 그 결과 RTLinux[6]와 RTAI[7] 및 Preemted-RT[8] 등이 Linux상에서 이중 커널 구조로 실시간성을 지원하기 위해 개발되었다. 본 논문에서 제안하는 RTiK-Linux 커널은 원래 윈도우용 실시간 커널인 RTiK를 Linux에 맞게 개발한 것으로서, 참고문헌[2]에서 RTiK은 윈도우에 일정한 부하를 부과하여 테스트한 결과 실시간 주기성이 오차 범위에 속하는 분해능을 입증하였다.

본 논문에서는 시간 제약에 민감한 응용에 적합하고, 또한 데이터 측정의 결과를 표시하기 위한 기존 운영체제의 다양한 기능을 활용하도록 윈도우 기반의 경성 실시간 커널인 RTiK를 리눅스용으로 개발이 가능하도록 리눅스 드라이브 모듈로 설계하였다.

기존의 리눅스에 실시간성을 부여한 이중 커널들은 타이머의 높은 분해능을 기반으로 인터럽트 지연 시간 및 스케줄링 지연 시간을 줄여 시간 제약 실시간 태스크의 예측성과 시간 결정성을 확보하는데 목적을 두었다. 본 연구에서 제안하는 RTiK-Linux 또한 이들 리눅스 기반의 실시간 태스크 지원하는 관점에서 유사한 기능을 지원하면서, 리눅스 동적 모듈 형태로 개발되어 용이한 이식성과 높은 분해능을 지원한다.

본 논문은 리눅스 커널에 드라이브 모듈로 동적으로 탑재되는 실시간 커널인 RTiK-Linux의 설계 및 검증에 관한 것으로 제 2장에서 관련 연구에 대하여 소개한다. 제 3장에서는 RTiK-Linux의 설계 방법론을 기술하고, 제 4장은 실험 및 환경을 소개하고 제 5장에서 결론을 맺는다.

II. 관련연구

1. RTLinux(Real-Time Linunx)

리눅스는 시분할 라운드 로빈 정책과 우선순위 기반의 공정성과 응답성을 중요시하는 범용 운영체제이다. 태스크 실행 권한은 우선순위 기반 비 선점이며, 시간 정밀도는 250Hz(즉, 4ms) 주기성의 연성 실시간 시스템으로 분류된다. 이러한 연성 실시간 리눅스 커널에 실시간 기능을 패치하여 이중 커널 구조로서 약 100us의 주기성을 보장하는 경성 실시간 리눅스인 RTLinux가 도입되었다[4][6][9].

RTLinux 스케줄러는 리눅스 커널을 휴지(idle) 태스크로 취급하여 실시간 태스크 혹은 실시간 커널이 비활성화 되었을 때 실행한다. 리눅스는 어떠한 경우라도 인터럽트를 중지 시키지 못하며, 선점에 대한 권한이 없다. 이러한 두 운영체제가 동시에 수행될 수 있는 것은 인터럽트 제어 하드웨어에 대한 소프트웨어적 에뮬레이션에 의한다.

실시간 태스크는 커널 공간에서 수행된다. 즉, 실시간 태스크(쓰레드)는 커널 메모리 공간 내에 실행되므로 스왑 아웃되지 않고 또한 TLB(Translation Lookaside Buffer) 미스 횟수를 줄여준다. 실시간 쓰레드는 프로세서 슈퍼바이저 모드로 동작하기 때문에, 하드웨어 전반적 제어가 가능하다.

어떤 임의의 자원을 리눅스의 프로세스가 점유함으로써 실시간 태스크가 그 자원을 기다리는 현상을 허용하지 않는다. 따라서 메모리 요청이나, 어떤 자료 구조에 대한 동기화나 스핀 락을 공유하지 않아야 한다.

RTLinux에서 모든 인터럽트 제어권은 RTLinux가 가진다. 기존 리눅스의 인터럽트는 RTLinux에서 소프트웨어적인 에뮬레이션으로 처리한다. 실시간 인터럽트 처리를 위하여 request_RTirq()와 free_RTirq()함수의 등록으로 RTLinux를 활성화한다.

실시간 태스크의 응용 범위가 극히 제한적이기 때문에 필요에 따라 리눅스 프로세스에 의한 확장 기능의 수행이 필요할 것이다. 이를 위하여 RTLinux에서는 커널 주소공간에 공유 메모리를 이용하여 실시간 태스크와 기존 리눅스 프로세스간의 데이터 전달을 위하여 실시간 FIFO를 두고 있다.

2. RTAI(Real-Time Appkication Interface)

RTAI(Real-Time Application Interface)은 RTLinux 개발초기에 함께 파생된 표준 리눅스 커널로서, 리눅스 커널에 기반을 둔 개방형 실시간 운영체제이다[7][10]. RTAI는 기본적으로 인터럽트 디스패처로 구성되는데, RTHAL(Real-Time Hardware Abastraction Layer)의 개념을 사용하여 하드웨어 인터럽트 가로채기를 통한 리눅스에 실시간성을 지원하고, 기존 리눅스로 인터럽트를 재 경로 설정하도록 한다. RTLinux와 마찬가지로 기존 리눅스 커널은 유휴 태스크로 취급되어 RTAI 스케줄러에 의해 관리되는 실시간 태스크보다 항상 낮은 우선순위를 갖게 된다.

RTAI 스케줄러에 의한 실시간 태스크는 커널 수준에서 수행되어 경성 실시간을 지원한다. 그러나 기존 리눅스의 라이브러리를 활용하는 다양한 기능성을 지원하기 위하여 LXRT(Linux Real-Time) 모듈을 통한 사용자 수준의 실시간 태스크의 수행도 가능하게 한다. LXRT 모듈은 커널 수준에서 실행되며 RTAI 스케줄러에 관리되어 리눅스 프로세스보다 높은 우선순위로 실시간성을 보장받는다.

3. PREEMPT_RT(Preempt Real-Time)

리눅스는 우선순위 설정과 POSIX primitive로서 연성 실시간을 제공한다. 그 커널은 두 가지 스케줄링 정책인 SCHED_FIFO와 SCHED_RR를 제공하고 그리고 SCHED_NORMAL에서 “nice”를 사용하여 우선순위 조정할 수 있는 기능으로서 실시간성을 제공한다. 그런데 커널에서 실시간 의미를 살리기 위하여 반드시 선점 가능한 커널이 되어야 한다. 리눅스는 완전한 선점 커널이 아니다. 따라서 리눅스 커널을 완전한 선점성을 부여하기 위하여 기존 리눅스에 PREEMPT_RT 패치를 통하여 경성 실시간성을 지원한다[8][11]. 이러한 패치는 높은 분해능을 가진 범용 클럭 이벤트 계층으로 완전한 선점을 가능하게 하여 리눅스 커널이 경성 실시간을 갖도록 하였다. PREEMPT_RT 패치는 필수적인 비 선점과 태스크 스케줄러 등 극히 일부분을 제외한 커널의 대부분에 적용되었다. 스핀 락과 같은 커널 내부 모든 Locking primitive들은 실시간 뮤텍스로 재 구현되었다. 결과적으로 spinlock_t 혹은 rwlock_t로써 보호되는 임계영역들이 선점 가능하게 되었다. 우선순위 상속도 커널 내부 스핀 락과 세마포를 위해 구현되었다. 모든 인터럽트 핸들러는 커널 쓰레드로 변환되었다. 모든 소프트 인터럽트 핸들러는 task_struct 구조체를 가지고서 커널 쓰레드 문맥으로 쓰레드화 되어 우선순위를 가질 수 있고, 적당하게 스케줄 되는 쓰레드로서 다루어진다. 그러나 현재는 커널 문맥으로 IRQ를 등록하는 것도 가능하다. 또한 기존의 리눅스 타이머 API가 높은 분해능을 가진 커널 타이머를 위하여 별개의 구조로 변환되어 높은 분해능으로 사용자 공간에서 POSIX 타이머를 사용 가능하게 하였다. PREEMPT_RT 패치는 인터럽트 지연을 개선하였고, 완벽한 선점 가능 커널을 제공하게 되었다.

Ⅲ. RTiK-Linux 설계 방법론

1. 윈도우 RTiK

RTiK는 기존 범용 운영체제의 실시간성을 지원하기 위하여 개발된 실시간 이식 커널이다. RTiK는 먼저 윈

도우 기반의 군사용 점검 장비의 실시간성을 지원하기 위하여 개발되었다[1].

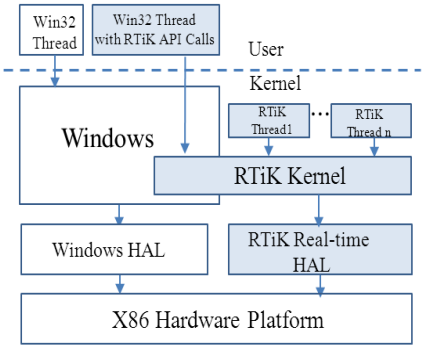

그림 1. 윈도우 RTiK 커널의 구조

[그림 1]과 같이 윈도우 RTiK는 윈도우와 이중 커널 구조로서 커널 수준의 실시간 쓰레드와 기존 윈도우 운영체제 기능을 동시에 지원하다. RTiK 실시간 이식 커널은 Local APIC Timer를 통해 윈도우 독립적인 타이머 인터럽트를 통해 실시간 쓰레드의 주기적 수행을 가능하게 하였다.

RTiK의 실시간 쓰레드는 Export Driver 형태로서 개발자가 정의하는 함수로 구현되며, RTiK에서 제공되는 실시간 커널의 API를 통하여 실시간 쓰레드로 등록하면, 독립 타이머에 의하여 RTiK 커널이 주기적으로 이 함수를 호출함으로써 실시간 기능을 수행하도록 한다.

RTiK는 윈도우와는 독립적인 타이머를 사용함으로써 Window HAL(Hardware Abstraction Layer)과 나란히 RTiK Real-Time HAL을 두고 있다. 상위에 RTiK 커널을 두고, 커널 모드의 실시간 쓰레드를 관리한다. RTiK는 윈도우 쓰레드의 수행한 결과, 0.1ms~15ms까지의 분해능에서 오차 범위 내에 정확하게 수행됨을 보였다.

2. RTiK-Linux 시스템 구조

윈도우 환경에서는 일반적으로 Local APIC Timer(Advanced Programmable Interrupt Controller)를 사용하지 않기 때문에 윈도우 버전의 RTiK에서는 이를 이용하여 독립적인 타이머를 발생시켰다. 하지만 리눅스의 경우 Local APIC Timer를 사용하고 있기 때문에, RTiK-Linux가 이를 이용하여 실시간 주기 분해능을 가지기 위해서는 기존에 수행되던 기능들과 상호 충돌이 없도록 구현해야 한다. 이를 위해서 리눅스 시스템 초기화 시 RTiK-Linux를 위한 인터럽트 핸들러를 IDT(Interrupt Descriptor Table)에 등록시켜 RTiK-Linux의 핸들러를 수행하도록 하였다. 즉, RTiK-Linux 모듈이 등록될 때 기존의 리눅스에서 수행되던 리눅스의 Local APIC Timer 핸들러를 대신해 RTiK-Linux에서 등록 시킨 Local APIC Timer 핸들러를 수행하게 된다. 따라서 리눅스는 RTiK-Linux 모듈이 적재되면 RTiK-Linux 디스패처에 의해 실행권을 부여받는데, RTiK-Linux의 태스크가 휴지시간(idle time)에 리눅스의 태스크가 수행 된다.

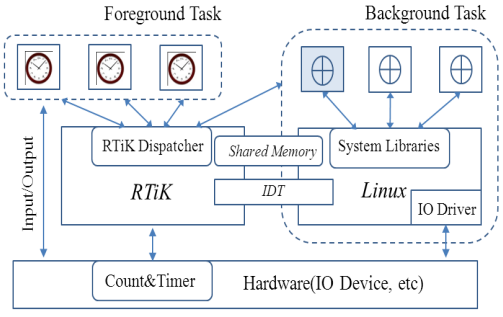

그림 2. RTiK-Linux 시스템 구조

[그림 2]는 RTiK-Linux의 전체적인 시스템 구조를 도식화한 것이다. 생성된 RTiK-Linux의 실시간 태스크는 커널 모드에서 동작하며, 하드웨어 자원을 직접 제어하는 권한을 가지고 있다. 리눅스 프로세스의 경우, 공유메모리를 통해 실시간 태스크의 데이터에 접근하여 리눅스 자원을 분석 및 모니터링이 가능하다.

3. RTiK-Linux 태스크의 동작 환경

RTiK-Linux의 실시간 태스크는 RTiK-Linux 디스패처에 의해 활성화된다. [그림 3]에서 보는 바와 같이 RTiK-Linux 태스크의 수행 레벨은 실시간 동작 레벨에 속하며, 리눅스는 RTiK-Linux 유휴 시간에 각 우선순위별로 동작 모드를 갖는다.

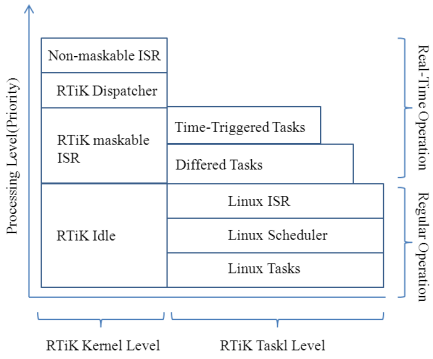

그림 3. RTiK-Linux Task 수행 레벨

4. 타이머 관리

Local APIC은 x86의 프로그램 가능한 인터럽트 제어기이다. 타이머의 구동은 “Initial Count”에 의해 설정된 카운터 값이 “Current Count” 레지스터에 복사되고, “Current Count” 레지스터의 카운트 값이 버스 클럭에 의해 감소되어 ‘0’이 되면 Local APIC Timer 인터럽트가 발생한다. 이때 프로세서는 인터럽트를 인지하여 IDT 벡터를 가지는 LVT(Local Vector Table)의 Timer 레지스터를 가지고 인터럽트 핸들러를 호출하게 된다.

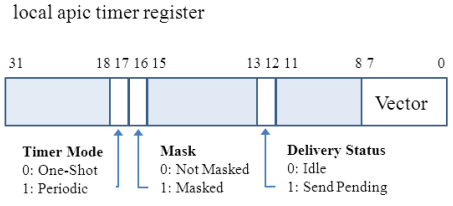

그림 4. x86 Local APIC Timer 레지스터

[그림 4]는 LVT의 Timer 레지스터의 구조를 나타낸 그림이다. 그림에서 보듯이 레지스터의 bit[7:0]을 통해 IDT에 접근하여 해당 핸들러를 수행 시킨다. 인터럽트 핸들러가 호출되고 난 뒤 다시 LVT의 “Initial Count” 레지스터의 값은 “Current Count” 레지스터에 복사되어 주기적인 인터럽트 동작을 한다[12-13].

또한 RTiK-Linux는 각 실시간 태스크를 위해 하나 이상의 소프트 타이머를 디스패처 테이블로 제공하고 있다. 디스패처 테이블은 리눅스의 콜 아웃 테이블과 유사한 형태로서 디스패처 테이블에 등록된 여러 개의 소프트 타이머는 태스크의 주기와 같은 카운터 값을 통해 마치 하드웨어 카운터와 같이 동작한다.

5. RTiK-Linux 인터럽트 재 경로 설정

IBM 호환 PC는 15개의 IRQ(Interrupt Request)를 제공하는 PIC(Programmable Interrupt Controller)의 계층 구조를 형성하는 2개의 8259A 칩을 사용한다. 각 하드웨어 장치 제어기는 PIC에 인터럽트 요청을 가능하게 하며, PIC의 인터럽트가 발생하게 되면 해당하는 IDT의 벡터로 변환한다. PIC는 인터럽트가 발생하면 프로세서의 인터럽트 핀에 시그널을 보내고, PIC는 CPU의 응답이 올 때 까지 기다린다. 시그널은 받은 CPU는 벡터를 인덱스로 사용하여 IDT로부터 일치하는 핸들러 호출한다.

본 연구에서 적용하는 X86 CPU는 256개의 인터럽트 벡터를 지원하고, IDT의 구조는 256개의 엔트리를 갖는 배열구조 형태를 가진다[12]. 또한 기존의 리눅스 타이머 인터럽트는 IRQ0로 매핑되어 있고, IRQ0에 해당하는 핸들러가 존재한다. 따라서 RTiK-Linux은 기존의 IRQ0에 해당하는 핸들러를 포함한 RTiK-Linux의 핸들러를 IDT의 빈 벡터에 등록해야한다. IDT에 RTiK-Linux 핸들러를 등록하기 위해서 리눅스 커널 API인 set_intr_gate()를 사용한다. 등록시킨 IDT 벡터를 사용함으로써 리눅스 커널과 RTiK-Linux의 커널 동작에 대한 상호 충돌 방지를 보장한다. 즉, IDT의 빈 엔트리에 RTiK-Linux의 인터럽트 게이트를 등록하고, Local APIC Timer 레지스터의 벡터 변경을 통해 리눅스가 가지고 있던 Local APIC Timer에 대한 제어권을 RTiK-Linux에게 양도하는 방식이다.

[그림 5]는 RTiK-Linux의 타이머 인터럽트를 리눅스로부터 가로채기 하는 과정을 도시한 것이다. 먼저 리눅스 커널에 set_intr_gate()함수를 통해 IDT의 0x70 게이트 영역에 RTiK-Linux를 위한 인터럽트 게이트를 등록시킨다. 리눅스가 수행 중인 상태에서 RTiK-Linux 모듈을 등록할 시 init_module()에서 ioremap_nocache()라는 API를 통해 LVT 레지스터들의 실제 물리 메모리

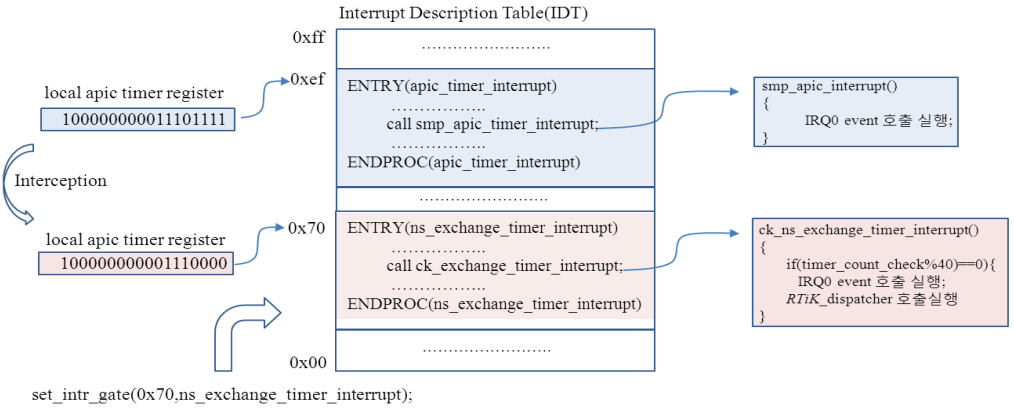

그림 5. RTiK-Linux 인터럽트 가로채기 과정

주소들을 가상 주소로 매핑 시킨다. 이 가상주소를 통해 필요한 주기 설정이나 인터럽트 벡터를 설정해 줄 수 있다.

본 논문에서는 0.1ms 주기로 타이머 인터럽트가 발생하도록 설정하였다. 기존의 리눅스에서 사용하던 벡터는 0xef이지만 RTiK-Linux의 모듈의 등록을 통해 인터럽트 벡터가 0x70으로 바뀌게 된다. 그리고 RTiK-Linux를 위해 등록시킨 타이머 인터럽트 핸들러 안에서 기존 리눅스가 가지고 있던 4ms 주기의 타이머 인터럽트 핸들러가 호출됨으로써 상호 충돌 방지를 보장하였다.

6. RTiK-Linux 동작 과정

리눅스의 다양한 기능성을 활용할 목적으로 이중 커널 형태로 설계된 RTiK-Linux 확장 커널의 동작 과정을 [그림 6]을 통하여 살펴본다. 먼저 RTiK-Linux 모듈이 리눅스 커널에 삽입되고, 삽입과 동시에 필요한 초기화를 수행한다. 초기화는 RTiK-Linux의 관리를 위하여 동작 주기 설정 및 벡터 값 조정을 API를 통하여 수행한다. 이때 LVT의 Timer 레지스터 안의 벡터 값은 IDT의 0x70 엔트리의 인터럽트 게이트를 가리키게 된다. 이 인터럽트 게이트는 이미 리눅스 빌드 시에 set_intr_gate()를 통하여 만들어져 있다. 따라서 등록된 RTiK-Linux의 핸들러는 RTiK-Linux의 확장 커널이 적재되고, Local APIC Timer 인터럽트가 발생 하게 되면 수행된다. 설정된 주기 값에 따라 타이머 루틴이 호출하게 되고, 디스패처를 호출하게 된다. 이 디스패처는 공유 메모리의 디스패처 테이블의 소프트웨어 카운트 값에 따라 주기별로 등록된 RTiK-Linux의 태스크를 호출 실행한다.

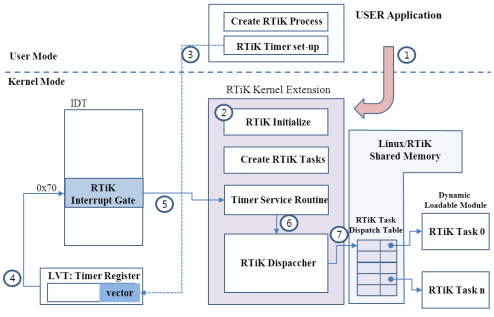

그림 6. RTiK-Linux 동작 과정

Ⅳ. 실험 및 평가

1. 실험 환경 및 방법

본 연구의 실험은 인텔 펜티엄4 2.66GHz PC에 Linux-2.6.9를 올려 실험하였다. 본 연구에서 제안하는 RTiK-Linux의 타이머 분해능에 대한 실효성을 검증하기 위하여 두 가지 방법으로 실험 하였다. 한 가지는 리눅스에 입출력의 부하를 가한 상태에서 RTiK-Linux의 주기를 0.1ms로 설정한 뒤 그 결과를 통해 분해능을 측

정하였고, 다른 실험은 RTiK-Linux를 RTLinux 및 RTAI에 동일한 환경에서 구축하여 각각의 분해능을 상호 비교를 하는 방법으로 실험하였다.

실험 데이터를 얻기 위해서 이중 커널의 공유 메모리를 이용하였다. 타이머 인터럽트의 발생 및 인터럽트 지연 시간을 고려한 Local APIC Timer의 발생 간격을 공유메모리에 기록하면, 리눅스 모니터링 응용 프로세스에서 수집하여 그 결과를 분석하도록 하였다. 실험에서 설정한 부하의 설정은, 임의의 장치에 읽기/쓰기를 수행하는 장치 모듈을 리눅스 커널에 0, 1, 5, 10개를 적재하여 위와 동일한 실험 방법으로 측정하였다.

2. 실험 결과 및 분석

[그림 7]은 타이머의 주기 설정을 0.1ms로 하여 부하를 가하지 않은 상태의 분해능이다. 그림에서 보듯이 오차 범위가 0.1% 이내로 거의 무시할 수준임을 알 수 있다. 그림 8~10은 타이머 설정 0.1ms에서 부하를(즉, 임의의 장치에 읽기/쓰기를 수행하는 장치 모듈을) 각각 1, 5, 10개 가하여 실험한 결과를 보여주고 있다.

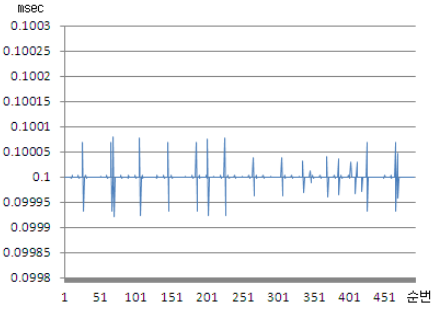

그림 7. 부하0에서 0.1ms 분해능

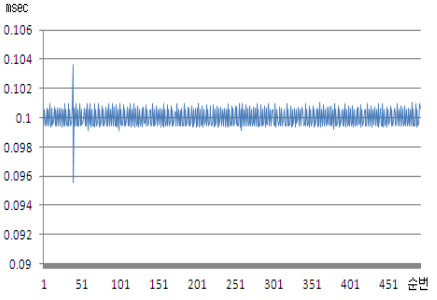

그림 8. 부하1에서 0.1ms 분해능

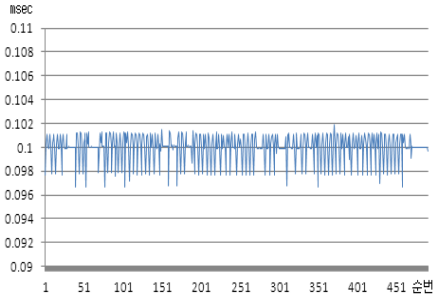

그림 9. 부하5에서 0.1ms 분해능

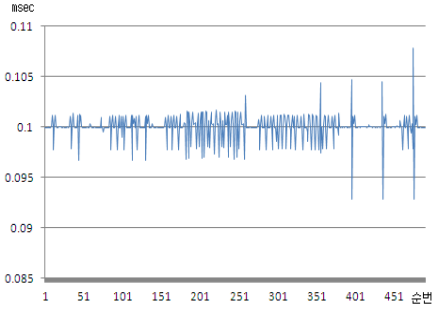

그림 10. 부하10에서 0.1ms 분해능

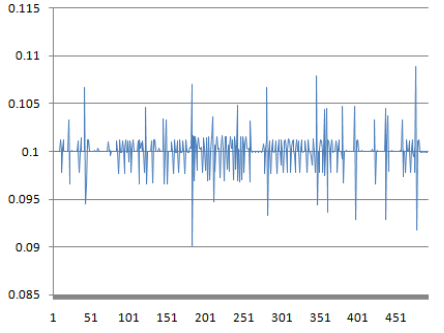

그림 11. 부하27에서 0.1ms 분해능

[그림 11]의 경우 리눅스 시스템이 정상 동작을 보장하면서 최대한의 부하를 가한 실험 결과이다.

결과에서 보듯이 리눅스의 일정 부하가 걸리더라도 RTiK-Linux의 실시간성 분해능 오차가 미미하여 크게 영향을 미치지 않음을 알 수 있다.

다음 실험은 RTiK-Linux와 RTLinux 및 RTAI를 동일한 조건에서 분해능에 대한 실험으로서, 타이머 레지스터에 주기 설정 값을 각각 0.1ms, 1ms 및 10ms로 하

여 각각 10,000번씩 실험했을 때의 분해능의 최대값 및 최소값을 측정하였다.

표 1. 타이머 주기 0.1, 1 및 10에서의 분해능 실험

		0.1ms	1ms	10ms
RTiK-Linux	max	0.10725	1.00017	10.00001
	min	0.09465	0.99899	9.99984
RTLinux	max	0.10402	1.00266	10.00553
	min	0.09702	0.99808	9.99427
RTAI	max	0.25940	1.29597	10.07650
	min	0.00011	0.72962	9.89988

[표 1]에서 보듯이, 동일한 조건에서 0.1ms, 1ms, 그리고 10ms에 대해서 RTiK-Linux의 경우 최대 각각 약 7%, 0.1%, 0.01%의 오차범위를 가지고, RTLinux의 경우 각각 약 4%, 0.2%, 0.5% 1ms의 오차범위를 가지는 것을 알 수 있다. 이를 통해 RTiK-Linux와 RTLinux는 유사한 분해능을 보이고 있음을 알 수 있다. 그러나 RTAI는 각 주기 설정 값에 대한 분해능의 오차 범위가 주기 설정 값을 크게 벗어나는 것을 확인할 수 있다. 즉, RTAI는 타이머 주기 설정 값이 작을수록 분해능의 오차 범위가 크게 벗어나며, 주기 설정 값이 커질수록 분해능이 설정 값에 근접하는 것을 알 수 있다. 이를 통해 RTAI의 경성 실시간성 지원이 부적합함을 알 수 있다. 실험 결과에서 RTiK-Linux는 리눅스의 이중 커널구조로서, RTAI 보다는 실시간성을 위한 분해능이 뛰어남을 알 수 있고, 또한 RTLinux 와 비슷한 분해능을 가지지만, 다양한 범용 운영체제의 이식성이 뛰어나고, 리눅스에서 더욱 쉽게 적용할 수 있다는 장점이 있다.

V. 결 론

본 논문에서는 범용 운영체제에 이중 커널을 통한 실시간성을 부여하는 RTiK-Linux 이식 커널의 설계 방법론을 제안하고, 이를 통해 실험 측정 결과를 분석하여 RTiK-Linux의 분해능을 검증하였다.

지금까지 리눅스에 실시간성을 부여하는 이중 커널 및 리눅스 커널에 패치를 통한 선점형 커널을 지원하는 연구가 진행되었다. 이러한 연구를 토대로 RT-Linux 및 RTAI가 구현 되었지만, RT-Linux의 경우 어셈블리어를 사용해야 되기 때문에 개발자가 사용하기 매우 복잡하다. 또한 리눅스 커널 패치를 통한 불편한 설치 과정을 요구하는 단점이 있다. 그러나 본 논문에서 제안하는 RTiK-Linux는 RTAI 보다 분해능이 실시간성을 충분히 보장하며, RTLinux보다 더 편리한 이식과정을 제공하는 데 본 연구의 기여도가 있다.

본 연구의 결과물은 특정 군사용 측정 장비의 실시간 데이터 수집에 목적을 두었으나, 앞으로 다양한 경성 실시간 기반 환경을 지원하기 위하여 실시간 응용에 적응할 수 있는 확장 기능에 관한 연구가 필요하다.

참 고 문 헌

[1] 충남대산학협력단, “점검장비용 실시간 윈도우즈 운영체제 개발,” (주)LIG 넥스원 최종보고서, 2010(10).

[2] 이진욱, “윈도우 기반의 점검장비에 실시간성을 지원하는 실시간 이식 커널의 설계 및 구현,” 한국콘텐츠학회논문지, 제10권, 제10호, pp.36-44, 2010.

[3] 송대기, 장부철, 이철훈, “고신뢰성 발사통제시스템을 위한 고장허용 통신 미들웨어 설계 및 구현,” 한국콘텐츠학회논문지, 제8권, 제8호, pp.37-46, 2008.

[4] 강민구, “스케줄러 선택기반의 실시간 리눅스의 성능분석,” 한국인터넷정보학회논문지, 제8권, 제1호, pp.71-78, 2007.

[5] C. L. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," Journal of the ACM, Vol.20, No.2, pp.46-61, 1973.

[6] http://www.rtlinuxfree.com

[7] http://www.rtai.org

[8] http://www.osadl.org

[9] Pavel Moryc, "Task jitter measurement under

RTLinux operating system," Proceeding of the IMCSIT, ISSN 1896-7094, pp.849-858, 2007.

[10] M. Bergsma, M. Holenderski, J. B. Reinder, and Johan J. Lukkien, "Extending RT AI/Linux with Fixed-Priority Scheduling with Deferred Preemption," OSPERT, pp.5-14, 2009.

[11] Carsten Emde, "Long-term monitoring of app arent latency in PREEMPT RT Linux real-time systems," OSADL, 2010.

[12] Intel, "Intel 64 and IA-32 Architectures Soft tware Developer's Manual Volume 1: Basic Architecture", 2009(9).

[13] Intel, "Intel 64 and IA-32 Architectures Soft ware Developer's Manual Vol.3: System Programming Guides," 2009(9).

저 자 소 개

김 주 만(Joo-Man Kim) 정회원

- 1984년 2월 : 숭실대학교 전산학과(공학사)
- 1998년 2월 : 충남대학교 컴퓨터공학과(공학석사)
- 2003년 2월 : 충남대학교 컴퓨터공학과 박사졸업(공학박사)
- 1985년 1월 ~ 2000년 2월 : ETRI 책임 연구원 (운영체제연구팀장)
- 1995년 ~ 1996년 : Novell Inc. 방문연구원
- 2001년 3월 ~ 현재 : 부산대 IT응용공학과 교수

<관심분야> : 임베디드 소프트웨어, Real-time OS. 분산병렬처리, 고장허용시스템

송 창 인(Chang-In Song) 준회원

- 2010년 8월 : 충남대학교 컴퓨터 공학과(공학사)
- 2010년 8월 ~ 현재 : 충남대 학교 컴퓨터 공학과(공학석사 과정)

<관심분야> : 실시간 운영체제, 내장형 시스템, 로봇 미들웨어

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