https://doi.org/10.5392/JKCA.2017.17.10.587

ARM 프로세서 기반의 리눅스를 위한 실시간 확장 커널 (RTiKA, Real-Time implant Kernel for ARMLinux)

Real-Time Kernel for Linux based on ARM Processor, RTiKA (Real-Time Implant Kernel For ARMLinux)

이승율, 이상길, 이철훈
충남대학교 컴퓨터공학과

Seung-Yul Lee(winrate92@cnu.ac.kr), Sang-Gil Lee(sk0137@cnu.ac.kr),
Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

최근 하드웨어의 발전으로 모바일 환경에서 리눅스나 안드로이드 같은 범용 운영체제 환경에서 실시간성의 요구가 증가하고 있으나, 범용운영체제의 경우 실시간성을 제공하지 못하는 단점이 있다. 이를 해결하기 위해 리눅스에 부가적으로 설계된 RTiK(Real-Time implanted Kernel)을 통해 실시간성을 제공할 수 있으나, 기존 RTiK의 경우 x86 아키텍처만을 제공하는 단점이 있으며, 실시간성 지원을 위해서는 CPU 플랫폼에 종속되는 한계가 있다. 본 논문에서는 CPU 플랫폼 이식을 위해 ARM 아키텍처를 위한 실시간 확장 커널인 RTiKA(Real-Time implant Kernel for ARMLinux)을 설계 및 구현한다. 실시간성 제공을 위해 독립적인 Local APIC Timer를 대체하는 MCT 타이머를 이용하였으며, 성능 검증 및 평가를 위해 생성된 실시간 태스크의 주기를 측정하였고, 1ms 단위의 주기를 바탕으로 여러 개의 실시간 태스크에 대한 동작을 보장할 수 있었다.

■ 중심어 : | 실시간 운영체제 | 리눅스 | ARM | 모바일 | 커널 |

Abstract

Recently, the demand for real-time performance in mobile environment is increasing due to the improvement of hardware performance, however a GPOS(General-Purpose Operating System) such as Android and Linux do not provide real-time performance. We developed RTiK(Real-Time implant Kernel) for this problem, but it has the disadvantage of supporting only x86 Architecture. In this paper, we designed and implemented a RTiKA(Real-Time implanted Kernel for ARM) to support real-time in ARM Linux. We used MCT(Multi-Core Timer) timer which replaces Local APIC Timer for real-time support, and we measured the period of generated real-time task for performance verification and evaluation. As the recent the RTiKA can guarantee the operating of several real-time tasks based on the cycle of 1ms.

■ keyword : | RTOS | Linux | ARM | Mobile | Kernel |

* 이 연구는 충남대학교 학술연구비에 의해 지원되었음
접수일자 : 2017년 08월 04일
수정일자 : 2017년 10월 17일
심사완료일 : 2017년 10월 17일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

I. 서 론

최근 하드웨어의 발전으로 과거와 비교하여 성능이 비약적으로 상승하면서, 고성능 시스템에서 수행할 수 있었던 제어 시스템이나 감지 시스템을 모바일 환경에서도 구현할 수 있는 성능에 도달하였다. 이에 따라서 최근 전용 단말의 소형화가 이루어지고 있으며, 이를 위한 운영체제 및 시스템 소프트웨어의 중요성이 높아지고 있다[1][2].

이런 모바일 단말에서는 과거 한정된 기능만을 정확하게 수행하기 위해 RTOS(Real-Time Operating System)를 사용하여 서비스를 제공하였다[3][4]. 실시간 시스템이란 실시간성을 보장하는 시스템을 말하는 것으로 실시간성은 요청된 작업을 논리적 정확성뿐만 아니라 시간적 제약을 만족시킬 수 있게 처리하는 것을 보장하는 시스템을 말한다. 시간적 제약은 특정 작업을 정해진 제한시간 내에 수행시키는 조건을 말하며, 그것이 만족되지 않을 경우에는 오작동이 일어날 수 있다[5]. RTOS는 미사일 제어 시스템이나 발전소와 같은 시간적 제약 조건을 만족시키지 못하는 경우 큰 문제가 생기는 상황에서 정확한 기능의 수행을 위해 사용된다. 이처럼 군용장비들은 수행 시간의 정확성인 실시간성이 보장되는 것이 필수적이며, 실시간성을 보장해주기 위해 RTOS를 사용한다.

그러나 최근 기술의 발전으로 하드웨어의 성능의 증가로 메모리 크기의 증가 및 동작 클럭이 높아짐에 따라 시스템에서 사용할 수 있는 리소스가 증가하여 민감한 시스템을 제외하고는 여유로운 시스템 자원을 바탕으로 다양한 기능을 수행할 수 있는 범용운영체제인 MS 윈도우즈나 공개 소프트웨어인 리눅스로 대체하고 있으며, 이는 사용의 편의성과 다양한 부가 기능을 지원할 수 있는 환경을 제공하게 되었다[6-8].

하지만 범용 운영체제는 다양한 서비스 및 프로그램이 균등하게 수행될 수 있도록 하는 스케줄링 알고리즘을 사용하기 때문에 요청된 작업에 대한 논리적 정확성뿐만 아니라 시간적 제약을 만족시킬 수 있도록 하는 실시간성을 만족하지 못한다. 실시간성이 보장되기 위해서는 높은 우선순위 태스크가 항상 선점 가능해야 하고 요청된 시간을 정확하게 만족해야 하는 것을 요구하기 때문에 범용 운영체제에서 실시간성을 만족하기 위한 부가적인 방법을 필요로 하게 된다.

범용 운영체제인 리눅스에 실시간성을 제공하기 위한 방법으로 오픈 소스로 공개되고 있는 PREEMPTIVE RT[9]와 RTAI(Real-Time Application Interface)[10]가 있으나 이는 2010년 이전에 나온 제한적인 모바일 환경만을 지원하기 때문에 최근에 출시되고 있는 고사양 모바일 전말에서 사용할 수 없는 단점이 있다. 또한 국내에서 연구되고 있는 RTiK(Real-Time implanted Kernel)은 범용 운영체제인 윈도우즈와 리눅스에서 x86 환경을 지원하고 있으며 사용자 영역에서 1ms 주기의 타이머를 통해 실시간성을 만족하고, 커널 영역에서 0.1ms 주기 단위로 실시간 태스크를 제공하고 있다[11-13]. 그러나 x86 시스템에 한정적으로 연구 개발되어 모바일 환경에 이식을 위해서는 ARM 아키텍처에 맞는 실시간 타이머 및 내부 구조 개선이 필요하다.

본 논문에서는 모바일 단말에서 사용할 수 없는 단점을 개선하여 ARM 프로세서를 위한 실시간 이식 커널을 위해 RTiK을 개선한다. 기존 RTiK-Linux에서 사용된 인텔 사의 프로세서에서 제공되는 Local APIC Timer를 대체하여 ARM 프로세서에서 제공되는 MCT 글로벌 타이머를 사용하여 실시간 타이머를 개선한다. 이를 통해 MCT 글로벌 타이머를 사용하는 실시간 이식 커널 구조를 설계하고 MCT 글로벌 타이머를 사용하기 위한 구조 분석과 설정 방법에 대해 기술한다.

본 논문은 제 2장에서 관련연구인 RTiK와 RTAI, PREEMPTIVE RT 및 최신 ARM 프로세서를 소개한다. 제3장에서는 ARM 프로세서를 위한 RTiKA의 구조 및 동작과정에 대해 소개하고 MCT 타이머를 사용한 실시간성 제공 방안을 기술한다. 제 4장에서는 실험 및 평가를 소개하고, 제 5장에서 결론을 맺는다.

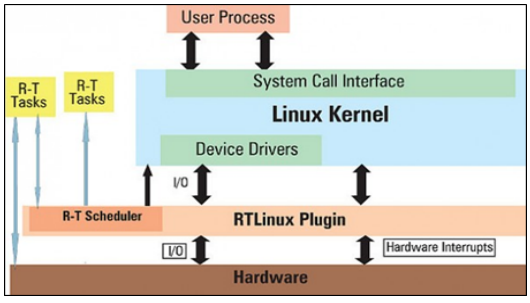

그림 1. RTLinux의 동작 구조

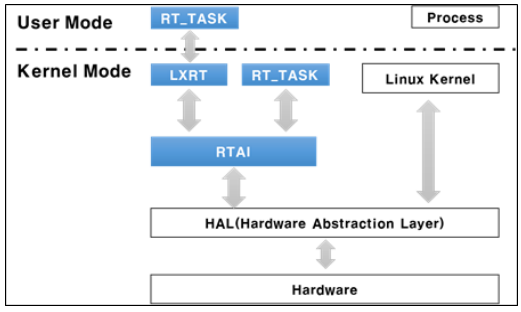

그림 2. RTAI의 동작 구조

II. 관련연구

1. PREEMPTIVE RT

리눅스는 기본적인 시간 처리를 위한 타이머가 제공되지만 연성 실시간성 만을 보장해준다. 경성 실시간성이 보장되기 위해서는 선점형 커널이여야 하지만, 리눅스는 완전한 선점형 커널이 아니다. 선점형 커널은 높은 우선순위 태스크가 준비 상태가 되었을 때 기존에 동작되던 태스크가 멈추고 높은 우선순위의 태스크를 동작시키는 커널을 말한다. 이로 인해 우선순위가 높은 태스크가 낮은 우선순위의 태스크를 대기하게 되어 경성 실시간성을 보장해주지 못하게 된다. PREEMPTIVE RT는 리눅스 커널을 완전한 선점형 커널로 변환시켜 리눅스에 경성 실시간성을 보장해 준다[9].

PREEMPTIVE RT는 x86플랫폼에 실시간성을 제공해주고 있으며 4ms/2.9ms 주기의 실시간성을 제공해주고 있다. ARM 프로세서에서는 ARM9, XScale, ARM7TDMI 등을 지원하고 있으며 최근 주로 사용되고 있는 ARMv7 이나 ARMv8이 적용된 프로세서에서는 사용할 수 없는 단점이 있다[14][15].

2. RTAI(Real-Time Application Interface)

RTAI는 1996년 이탈리아의 밀란 대학교 DIAMP (Dipartimeno di Ingegneria Aerospaziable)의 Paolo Mantegazza 교수에 의해서 RTLinux의 개념을 기본으로 개발되었으며 패치를 통해 리눅스에 실시간성을 제공한다. 하드웨어와 리눅스 커널 사이에 실시간 커널이 추가된 구조로 기존 리눅스 커널을 우선순위가 낮은 실시간 태스크로 관리한다. 이러한 구조에 의해 우선순위가 높게 설정된 실시간 태스크들이 시간적 제약을 만족시켜 실시간성을 보장받게 되고 이후에 리눅스 커널을 동작시키게 된다. 또한 RTAI는 LXRT(Linux Real-Time) 모듈을 통해 사용자 영역에 실시간성을 보장받을 수 있게 하여 리눅스에서 제공되는 라이브러리를 사용할 수 있는 방안도 제공되고 있다[10].

RTAI는 여러 가지 아키텍처를 지원하고 있다. x86, x86_64, PowerPC, ARM(StrongARM; ARM7: clps711x-family, Cirrus Logic EP7xxx, CS89712, PXA25x), m68k을 지원하고 있다. 하지만 PREEMPTIVE RT와 마찬가지로 한정된 모바일 플랫폼을 지원하고 있으며, 최근 사용되고 있는 ARMv7 및 ARMv8 아키텍처는 지원하지 않는 단점이 있다[10].

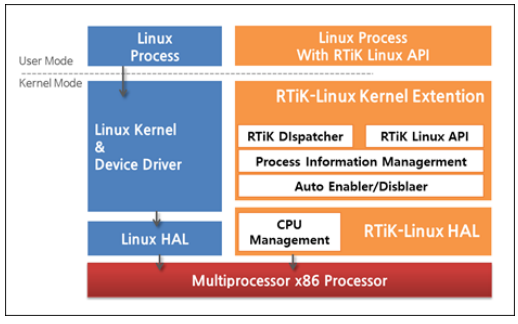

그림 3. RTiK-Linux의 구조

3. RTiK-Linux

범용운영체제인 리눅스에 실시간성을 제공하기 위한 연구로 RTiK-Linux가 개발되었다. RTiK-Linux는 실시간성을 제공하기 위해 Tick 인터럽트를 발생시켜 여러 실시간 태스크의 실시간 수행 성능을 만족시킨다[11-13].

RTiK-Linux는 x86 프로세서에서 제공되는 Local APIC Timer를 사용하여 실시간성을 제공하기 위한 Tick Timer로 사용한다. 그러나 Local APIC Timer는 이미 리눅스에서 사용 중이므로 리눅스와 RTiK-Linux가 충돌하지 않고 공유하며 사용할 수 있도록 해야 한다. 이를 위해 Local APIC Timer에서 발생되는 인터럽트를 처리하기 위한 수행함수를 RTiK-Linux의 처리함수로 대체하고 휴지시간(idle- time)일 때 기존의 처리함수를 수행하도록 하여 상호 충돌을 방지할 수 있게 구현되어 있다. 이러한 구조는 Local APIC Timer가 제공되는 인텔 프로세서에서만 사용가능한 구조로 ARM 프로세서 기반의 모바일 리눅스 환경에서는 사용하지 못한다는 단점이 있다. 본 논문에서는 이런 단점을 개선하고 모바일에서 실시간성을 제공하기 위한 연구를 기술한다.

4. ADEOS(Adaptive Domain Environment for Operating Systems)

ADEOS는 나노커널 기반의 HAL을 제공하는 방법으로 하드웨어와 운영체제 사이에서 동작한다. 다른 나노커널과 달리 외부커널에 대한 저수준의 계층이 아닌 여러 운영체제 간의 가상화 수준의 기능을 제공하는 장점이 있으며, 여러 운영체제간의 하드웨어 자원 공유를 위한 유연한 환경을 제공한다[17].

ADEOS는 리눅스 커널 아래에서 동작하며, SMP 클러스터링, 효율적인 가상화, 패치없는 커널 디버깅 및 리눅스용 실시간 시스템 등을 위한 가능성을 제시하였다. ADEOS는 다른 OS가 함께 수행할 수 있도록 리눅스의 로딩 가능한 커널 모듈의 형태로 제공된다. 이는 ADEOS가 RTAI의 컨텍스트를 위해 개발되었으며, 이를 통해 RTOS로부터 HAL을 분리하여 모듈화 할 수 있도록 ADEOS I-pipe를 통해 인터럽트 가상화를 제공한다.

본 연구에서는 ADEOS와 유사하게 로드 가능한 커널 모듈의 형태로 사용되는 실시간 확장 커널에 대한 연구를 기술한다.

5. Xenomai

Xenomai는 리눅스 커널을 위한 실시간 개발용 프레임워크로, 유저 영역에 응용 프로그램에 대한 경성 실시간 성능을 제공한다[18]. Xenomai는 2001년 시작되어 2003년 RTAI와 함께 상용 등급의 실시간성을 제공하는 리눅스를 위한 자유 소프트웨어 플랫폼인 RTAI/fusion을 위해 통합되었으나, 2005년 분리되었다. Xenomai는 추상화된 RTOS 코어를 바탕으로, 실시간 인터페이스를 사용할 수 있도록 해주며, 대부분의 RTOS 서비스를 사용할 수 있도록 해준다[19]. 본 논문에서 제공하는 RTiKA는 Xenomai와 비슷하게 유저 영역에 실시간성을 제공할 수 있도록 한다.

III. ARM 프로세서 상의 리눅스에 실시간성 제공 방법

본 장에서는 ARM 프로세서 기반의 리눅스 환경에 정확한 주기에 따른 실시간성을 제공하기 위한 구조 및 방법을 제시한다. 먼저 RTiKA의 전체 구조와 MCT에 대해 설명한다. 다음으로 RTiKA를 동작시키는 요소들에 대해 설명하며, 각 요소들 간의 상호 동작에 대한 과정을 설명한다.

1. RTiKA 전체 구조

RTiKA는 [그림 4]과 같은 이중 커널 구조로 하드웨어 접근의 용이성과 간편한 이식성을 위해 모듈 형태의 디바이스 드라이버로 제작되어 있고 별도의 Time Tick 인터럽트를 사용하여 기존 커널과 독립적으로 동작된다. MCT는 RTiKA가 동작하는데 사용되는 인터럽트를 제공하기 위해 기존의 커널과 구분되어 동작할 수 있는 환경을 제공한다. 커널 영역과 유저 영역에 실시

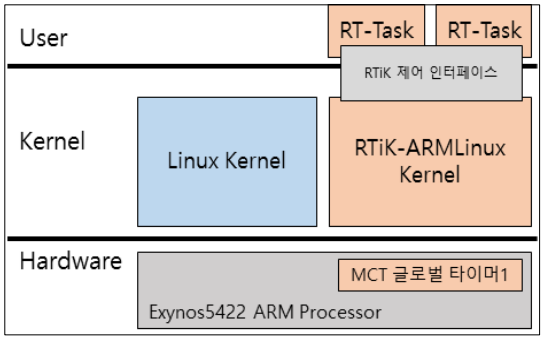

그림 4. RTiKA 구조

간성을 제공할 수 있으며 유저 영역에서는 RTiKA에서 제공되는 인터페이스를 사용하여 유저 영역의 응용 프로그램이 실시간성을 제공받는다.

이를 통해 리눅스와는 독립적인 실시간 태스크를 사용하여 높은 실시간 성능 주기를 보장받을 수 있으며, 기존의 어플리케이션에서 필요한 부분에서 실시간 태스크를 생성하고 사용할 수 있도록 하는 인터페이스를 통해 기존 사용자가 프로그램을 크게 변경하지 않도록 하는 편의성을 보장해 준다.

2. 실시간성 제공을 위한 타이머 구조

RTiKA는 기존 인텔 프로세서에서 사용되는 Local APIC 타이머를 대체하여 MCT를 사용한다. 본 절에서는 MCT에 대한 구조 및 MCT를 이용한 타이머 생성 방법과 실시간성을 제공하는 과정을 기술한다.

2.1. MCT의 구조

MCT는 ARM 프로세서에서 제공하는 하드웨어 레벨의 타이머로 기존의 타이머를 대체하여 실시간성을 보장해줄 수 있는 환경을 제공해준다. MCT는 4개의 글로벌 타이머와 8개의 로컬 타이머로 구성되어 있으며 각 타이머는 TCLK(Tiemr Reference ClocK)에 의해 동작된다. 타이머는 설정된 시간에 GIC (Generic Interrupt Controller)로 인터럽트를 전달하고, 전달된 인터럽트는 해당 인터럽트를 처리할 코어에 전달되어 처리한다. 클로벌 타이머는 대부분의 인터럽트를 처리하는 주 코어인 #0 코어에서 처리하도록 설정되어 있으

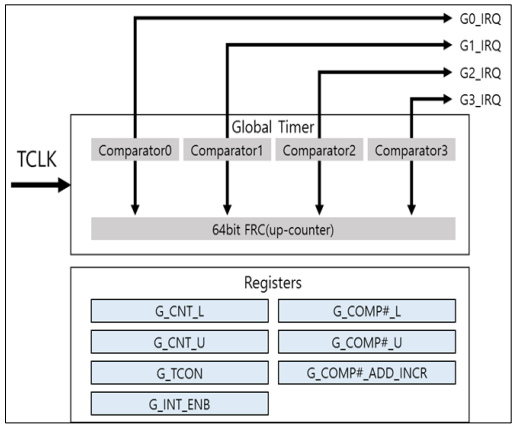

그림 5. MCT 글로벌 타이머 구조

며, 로컬 타이머들은 각 코어마다 하나씩 인터럽트를 담당하여 처리한다. 리눅스에서는 8개의 로컬 타이머와 글로벌 타이머 #0을 사용하고 있어 RTiKA에서는 이를 회피하여 글로벌 타이머 #1을 사용한다.

MCT 글로벌 타이머는 [그림 5]과 같은 구조로 TCLK FRC(Free-Running Counter), Comparator로 구성되어 있으며 레지스터를 사용하여 제어한다. TCLK는 메인 OSC(Oscilloscope)의 클럭을 Prescaler와 Divider로 가공한 클럭으로 24Mhz로 제공된다. FRC는 TCLK에 의해 1/24000ms 마다 1씩 증가하는 카운터로 4개의 글로벌 타이머가 공용으로 사용한다. 각각의 글로벌 타이머마다 Comparator가 존재하며 Comparator의 값은 FRC 값과 비교되어 동일한 값이 되었을 경우 인터럽트를 발생시킨다.

MCT 글로벌 타이머를 제어하기 위한 레지스터들의 역할은 [표 1]와 같다. FRC와 Comparator는 64비트 범위의 값을 가지고 있어 32비트 범위의 레지스터 두 개를 사용하여 값을 설정한다. FRC는 하위 비트 값을 설정하기 위한 G_CNT_L 레지스터와 상위 비트 값을 설정하기 위한 G_CNT_U 레지스터를 사용하여 Comparator들을 각 G_COMP#_L 레지스터와 G_COMP#_U 레지스터를 사용하여 값을 설정한다.

MCT 글로벌 타이머를 주기적으로 동작하기 위해서는 주기 간격을 G_COM#_ADD_INCR 레지스터에 설정해주어야 한다.

표 1. MCT 글로벌 타이머 레지스터

Register	Offset	Description
G_CNT_L	0x0100	FRC 하위 비트 값
G_CNT_U	0x0104	FRC 상위 비트 값
G_COMP1_L	0x0200	Comparator1 하위 비트 값
G_COMP1_U	0x0204	Comparator1 상위 비트 값
G_COMP1_ADD_INCR	0x0208	Comparator1 자동 증가 값
G_TCON	0x0240	글로벌 타이머 제어
G_INT_ENB	0x0248	인터럽트 활성화

다음으로 G_TCON 레지스터는 다음 [표 2]과 같은 구조로 각 비트마다 타이머의 동작 여부나 동작 방식을 설정하는 데 사용된다.

표 2. G_TCON 레지스터

Name	Bit	Type	Description
RSVD	[31:9]	R	Reserved
Timer Enable	[8]	RW	FRC 카운트 증가
Auto-Increment3	[7]	RW	Comparator3 자동증가 활성화
Comp3 Enable	[6]	RW	FRC 와 Comparator3 비교 활성화
Auto-Increment2	[5]	RW	Comparator2 자동증가 활성화
Comp2 Enable	[4]	RW	FRC 와 Comparator2 비교 활성화
Auto-Increment1	[3]	RW	Comparator1 자동증가 활성화
Comp1 Enable	[2]	RW	FRC 와 Comparator1 비교 활성화
Auto-Increment0	[1]	RW	Comparator0 자동증가 활성화
Comp0 Enable	[0]	RW	FRC 와 Comparator0 비교 활성화

Timer Enabler는 TCLK가 발생할 때마다 FRC 값을 증가시키는 카운트 여부를 설정하는 비트이다. Auto-Increment#은 타이머 동작 시 자동으로 다시 동작하기 위해 Comparator 값을 증가 시킬지에 대한 여부를 설정하는 비트이다. Comp# Enable 비트는 Comparator의 값과 FRC의 값을 비교하여 타이머 동작 여부를 설정하는 비트이다. RTiKA에서는 MCT 글러벌 타이머 #1을 주기적으로 사용하기 위해 Comp#1 Enable 비트와 Auto-Increment#1 비트를 1로 설정하여 타이머를 동작시킨다.

G_INT_ENB 레지스터는 다음 [표 3]와 같은 구조로 #0 ~ #3 비트가 사용되며 글로벌 타이머 동작 시 인터럽트의 발생 여부를 설정하는 비트이다. RTiKA에서는 C_INT#1_ENABLE 비트를 1로 설정하여 MCT 글로벌 타이머 1이 동작할 때마다 인터럽트를 발생시킨다.

표 3. G_INT_ENT 레지스터

Name	Bit	Type	Description
RSVD	[31:4]	R	Reserved
C_INT3_ENABLE	[3]	RW	글로벌 타이머3 인터럽트 활성화
C_INT2_ENABLE	[2]	RW	글로벌 타이머2 인터럽트 활성화
C_INT1_ENABLE	[1]	RW	글로벌 타이머1 인터럽트 활성화
C_INT0_ENABLE	[0]	RW	글로벌 타이머0 인터럽트 활성화

2.2 MCT 글로벌 타이머 동작 과정

MCT 글로벌 타이머는 [그림 6]과 같은 동작과정을 통해 타이머 인터럽트를 제공한다. 이런 동작과정을 번호순으로 나열하면 다음과 같다.

① MCT 글로벌 타이머는 TCLK에 의해 동작이 시작된다.

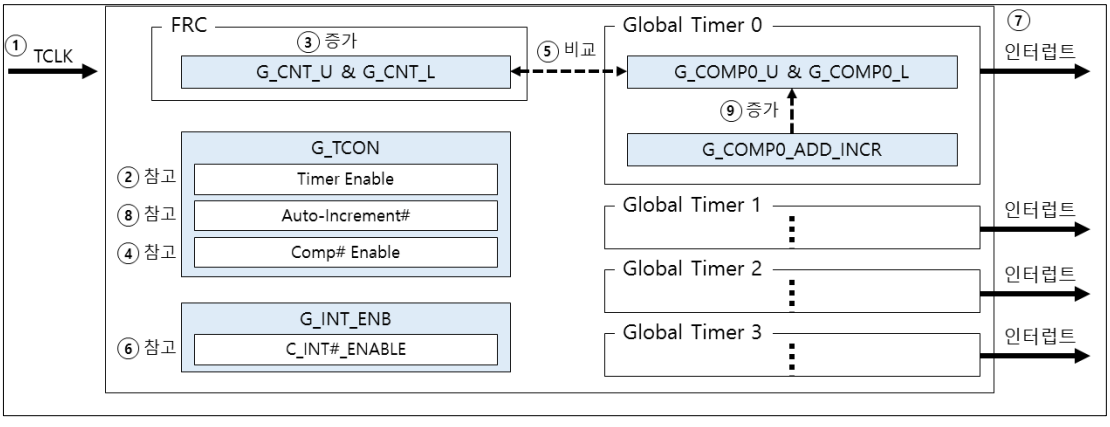

그림 6. MCT 글로벌 타이머 동작과정

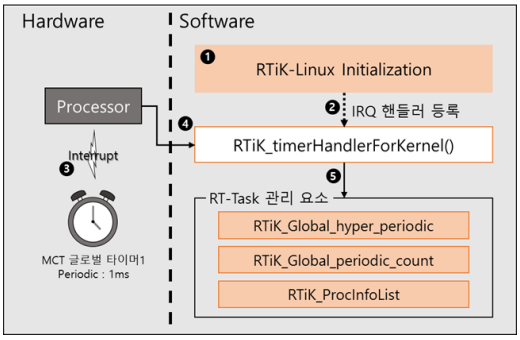

그림 7. RTiK-ARMLinux 동작 구조

② TCLK 신호가 발생하면, G_TCON 레지스터의 “Timer Enable” 비트 값을 통해 활성화 여부를 확인한다.

③ “Timer Enable” 비트 값이 활성화 되어 있으면 FRC 값을 증가시켜준다.

④-⑤ 증가된 FRC 값을 통해 각 타이머의 “Com# Enable” 비트의 활성화 여부에 따라 Comparator 의 값과 비교한다.

⑥-⑦ 비교된 값이 같고 G_INT_FENB의 비트가 활성화 되어있다면, 인터럽트를 발생하게 된다.

⑧-⑨ “Auto Increment” 값이 설정되어 있는 경우, Comparator 값을 G_COMP#_ADD_INCR 레지스터에 저장되어 있는 값만큼 증가시킨다.

이와 같은 과정을 통해 글로벌 타이머가 동작하며 인터럽트를 발생시키고, 이를 통해 RTiKA에서 실시간성을 제공받기 위한 실시간 타이머로 동작할 수 있는 인터럽트가 된다.

2.3 1ms 주기 제공을 위한 MCT 설정 방법

RTiKA는 1ms 주기의 Tick 타이머를 MCT의 동작으로부터 제공받아서 동작한다. RTOS에서 사용되는 Tick은 시스템의 주요 기능 수행을 위해 동작하므로 이를 이용하여 시스템에 대한 기초 타이머를 사용하게 된다. 기본적으로 MCT 글로벌 타이머에서 공통으로 사용되는 FRC를 G_CNT_U와 G_CNT_L 레지스터의 값을 0으로 초기화해 주고 카운트 값이 증가하도록 G_TCON 레지스터의 “Timer Enable” 비트를 설정하여 활성화 해준다. 다음으로 MCT 글로벌 타이머#1을 1ms 주기로 동작할 수 있도록 G_COMP#1_ADD_INCR의 값을 24,000으로 설정해준다. 그리고 G_TCON의 “Comp#1 Enable” 비트의 값을 활성화 해주며 G_INT_ENB 레지스터의 “C_INT#1_ENABLE” 비트의 값을 활성화 시켜주는 것으로 실시간 서비스를 위한 타이머 인터럽트를 사용할 수 있도록 준비된다.

3. 실시간 태스크 동작 과정

3.1 RTiKA 동작 구조 및 구성 요소

RTiKA는 리눅스에 실시간성을 제공하기 위한 이중 커널 구조를 가지고 있으며, 해당 커널 내부에서의 동작 구조는 다음 [그림 7]과 같다. 리눅스에 디바이스 드라이버 형태로 설치되는 RTiKA는 리눅스에 설치되면 기본적인 초기화 과정을 거치게 된다. 초기화 과정에서는 실시간성 제공을 위한 타이머 인터럽트 처리를 위해 RTiKA의 핸들러 함수를 등록한다. MCT 글로벌 타이머 #1에 의해 인터럽트가 발생되면 핸들러 함수를 호출하고, 핸들러 함수는 RTiKA를 동작시키는 구성 요소를 통해 실시간 태스크가 수행될 수 있도록 신호를 전송한다. RTiK_ProcInfoList는 실시간성을 제공하기 위한 실시간 태스크의 정보를 저장하는 리스트로, 리스트의 노드에는 실시간 태스크의 리눅스 프로세스 정보와 동작 주기를 저장한다. RTiK_Global_periodic_count 는 경과 시간을 기록하기 위한 것으로 실시간 태스크들의 동작 주기 정보와의 비교를 통해 실시간 태스크를 동작시키는데 사용된다. RTiK_Global_hyper_periodic은 시간 경과에 의해 RTiK_Global_periodic_count 값이 오버플로우 되는 것을 방지하기 위한 프레임의 최대 주기(HyperFrame)으로 사용된다.

3.2 실시간 태스크의 동작 과정

RTiKA는 MCT 글로벌 타이머에 의해 수행되는 핸들러 함수를 통해 실시간성을 제공한다. [그림 8]은 실시간 타이머 핸들러를 통한 실시간 태스크 호출 과정을 나타내는 것이다. MCT 글로벌 타이머 #1에서 1ms 주기로 인터럽트가 발생되면 인터럽트를 처리하기 위한 핸들러 함수가 호출된다. 핸들러 함수는 전역변수인

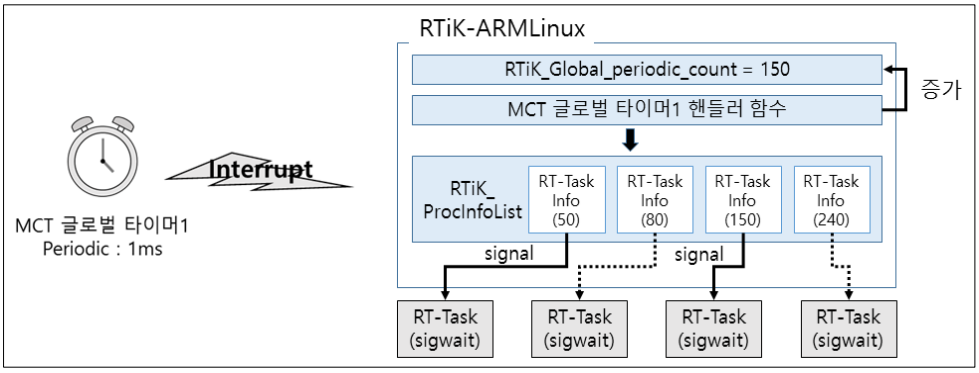

그림 8. 타이머 핸들러를 통한 실시간 태스크 동작 구조

RTiK_Global_periodic_count의 값을 증가시켜 1ms 단위로 경과 시간을 기록한다. 이후 RTiK_ProcInfoList에 등록되어 있는 모든 실시간 태스크의 정보를 확인한 다음, 현재 Time Tick값과의 나머지 연산을 통해 실시간 태스크를 확인하여 시그널을 보내는 것으로 대기 상태에 있던 실시간 태스크를 실행 상태로 변경한다. 이때, 컨텍스트 스위칭을 통해 일반 태스크가 중지되고 실시간 태스크가 CPU를 선점하게 되어 항상 실시간 태스크가 우선 실행될 수 있도록 하여 주기에 맞는 수행을 보장한다. 이런 과정을 통해 ARM 아키텍처를 사용하는 모바일 플랫폼에 정확한 주기를 만족하는 실시간성을 제공할 수 있도록 하는 구조를 설계 및 구현하였으며, 이를 통한 성능 평가를 진행한다.

IV. 실험 및 평가

본 장에서는 RTiKA를 적용한 환경에서 실시간 태스크가 항상 요청한 주기에 정확히 수행되는 실시간성 보장 여부 확인을 위한 실험 방법 및 결과에 대해 서술한다.

1. 실험 환경 및 방법

성능 평가를 위한 실험 환경은 모바일 단말에서 사용하는 ARM 프로세서인 Exynos5422 Octa Core (Cortex A15 2.1GHz Quad/A7 1.5 GHz Quad)[16]를 탑재한 개발 보드와 PC를 시리얼 통신으로 연결한 후, 개발 보드에서 성능을 측정하였다.

실험 방법은 개발 보드에 RTiKA를 설치하고 실시간 태스크를 등록하여 동작 시킨다. 실시간 태스크의 주기를 기록하여 주기와의 실제 호출 시간의 오차를 확인하는 것으로 요청한 시간에 정확하게 실행되는 실시간성 보장 여부를 판단한다.

실험은 두 가지로 진행된다. 첫 번째 실험으로는 오차율이 가장 큰 최소 주기인 1ms 주기로 동작하는 실시간 태스크를 10분간 동작시켜 총 60만 번의 동작 주기를 측정하는 실험으로 기존의 RTiK-Linux에서의 실험 결과와 비교하여 오차율의 차이가 없는 것을 확인한다. 두 번째 실험으로는 5ms, 8ms, 15ms, 24ms주기로 동작하는 여러 실시간 태스크를 동시에 동작시켜 여러 개의 실시간 태스크가 있는 환경에서 각 태스크의 정확한 수행 시간을 보장하는 실시간성의 제공 여부를 확인한다.

2. 실험 결과

실험 결과는 표와 그래프로 나타낼 수 있으며, 그래프에서 x축은 주기 측정 횟수를 의미하며, y축은 주기 값을 나타낸다. 그래프가 기준 값에 가깝게 나타낼수록 오차가 적다는 것을 나타낼 수 있다.

표의 데이터는 측정된 주기의 최솟값과 최댓값, 오차율을 나타낼 수 있도록 하며, 기존의 RTiK-Linux의 데이터와 비교하여 표현한다.

2.1 1ms 주기 태스크의 성능 검증

첫 번째 실험으로 1ms 주기 태스크에 대한 성능 검증

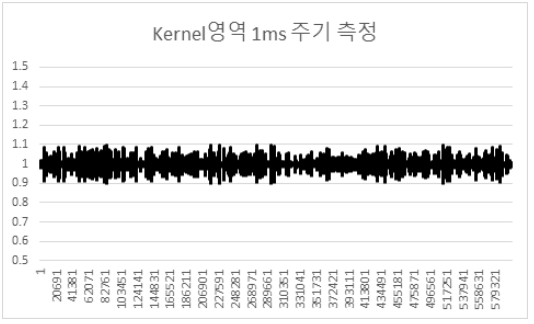

그림 9. 1ms 주기 실시간 태스크 실험 결과

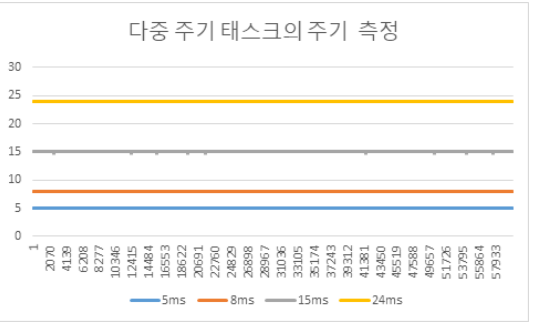

그림 10. 다중 주기 태스트크 실험 결과

을 수행하였다. 성능 측정된 결과는 [그림 9]으로 나타낼 수 있다. y축의 눈금 단위는 0.1ms로 기준 주기에서 10%의 성능 단위로 구분하여 나타내었다. 측정된 주기 값이 대부분 0.1ms 오차 이내로 측정된 것을 확인할 수 있으며, 모든 태스크의 수행 주기가 최대 7% 이내인 0.07ms 의 오차를 보이는 것으로, 주기를 만족하는 실시간성을 제공받는 것을 확인할 수 있었다. 또한 기존 시스템과의 성능 비교를 위해, RTiK-Linux와의 성능 비교는 [표 4]로 나타낼 수 있다. 성능을 측정한 결과 최댓값 1.070ms, 최솟값 0.950ms 로 약 7%의 최대 오차를 보인다. 이는 x86 시스템에서 측정하였던 4%의 오차보다 조금 높을 수 있으나, 40μs와 70μs의 차이를 보이는 것으로 요청한 1ms 주기 성능에 영향을 주지 않는 범위 이내라고 할 수 있으며, 항상 1ms 주기 단위에 만족할 수 있는 안정된 성능을 제공하는 것을 확인하였다.

표 4. 1ms 주기 측정 결과

종류	구분	값
기존 RTiK –Linux (x86)	최댓값	1.023 ms
	최솟값	0.978 ms
	평균값	1.000 ms
	오차	4 %
RTiKA (ARM)	**최댓값**	**1.070 ms**
	최솟값	**0.950 ms**
	평균값	**1.000 ms**
	오차	**7 %**

2.2 다중 주기 태스크의 성능 검증

두 번째 실험으로 각각 다른 주기를 가지는 여러 개의 태스크를 동시에 수행하여 성능을 검증하였다. 다음 그림은 실험을 통해 측정된 데이터이다.

각 태스크의 수행 횟수는 총 6만 회의 주기 태스크 호출에 대한 성능 측정을 수행하였다. 각 태스크가 모두 종료된 후의 데이터를 각각 수집하였으며, 성능 검증을 위해 각 태스크마다 그래프를 분리하여 표현하였다. 각 태스크의 실험 결과는 [그림 10]으로 나타낼 수 있으며, 각 실험을 통해 얻어진 정보는 [표 5]로 요약할 수 있다.

표 5. 여러 태스크의 주기 측정 결과

종류	구분	값
5ms 태스크	최댓값	5.034 ms
	최솟값	4.964 ms
	평균값	5.000 ms
	오차	± 0.7 %
8ms 태스크	최댓값	8.034 ms
	최솟값	7.965 ms
	평균값	8.000 ms
	오차	± 0.44 %
15ms 태스크	최댓값	15.045 ms
	최솟값	14.954 ms
	평균값	15.000 ms
	오차	± 0.30 %
24ms 태스크	최댓값	24.054 ms
	최솟값	23.934 ms
	평균값	24.000 ms
	오차	± 0.27 ±

측정된 데이터의 경우 1ms 주기 측정 데이터보다 오차율이 적은 것을 볼 수 있다. 오차의 경우, 제시된 주기에서 실제 태스크가 호출된 주기의 최댓값과 최솟값을 바탕으로 비교하여 차이를 계산하고 있다. 이 때, 5ms 주기에서는 0.03ms 이내, 8ms 주기에서는 0.04ms 이내, 15ms 주기에서는 0.05ms 이내, 24ms 주기에서는 0.06ms 이내로, 공통적으로 0.1ms 이내에는 해당 태스크가 반드시 호출될 수 있도록 주기가 제공되고 있으

며, 이는 다양한 주기를 가지는 태스크가 있는 환경에서 큰 지연시간 없이 안정된 수행 성능을 제공받을 수 있는 것을 볼 수 있다. 이를 통해서 RTiKA를 사용한 ARM 리눅스에서 사용자 영역 응용 프로그램에서 정확한 주기를 만족하는 실시간성을 제공받는 것을 확인하였다.

V. 연구 결과 및 향후 연구방향

본 논문에서는 ARM 프로세서 환경에서 동작하는 리눅스에 실시간성을 부여하기 위해 RTiKA를 설계 및 구현하였다. RTiKA를 ARM 프로세서에서 동작할 수 있도록 기존의 실시간 타이머인 Local APIC Timer를 ARM에서 제공하는 MCT를 사용하는 방식으로 개선하여 실시간 타이머를 구현하였고, 이를 통해 리눅스에서 실시간성을 제공받을 수 있도록 실시간 태스크를 사용할 수 있는 자료구조를 제공하며, 이를 통해 실시간 태스크가 주기에 맞추어 실행될 수 있는 구조를 제안하였다.

구현된 RTiKA의 성능을 검증하기 위해 주기에 발생하는 Tick 값을 측정하여 이전 값과의 비교를 통해 주기에 응답한 시간을 측정하였다. 최소 주기인 1ms를 바탕으로 실행하였을 경우 약 7%의 오차 범위 내에서 동작하여 1ms 단위로 실시간 태스크의 주기를 설정할 수 있도록 하는 성능을 확인하였다. 또한 두 개 이상의 실시간 태스크가 있을 때, 다양한 주기에 대한 실시간성을 만족할 수 있도록 하는 것을 확인하였으며 5ms, 8ms, 15ms, 24ms를 측정하여 모두 0.1ms 이내의 응답시간을 보이는 것으로 확인할 수 있어서, ARM 프로세서에 실시간성을 제공하는 것을 증명하였다.

향후 연구과제로는 ARM 프로세서를 사용하는 휴대용 모바일 기기에서 RTiKA를 이용해 실시간성을 제공하는 연구가 필요하다. 이를 통해 다양한 디바이스에 적용할 수 있는 여부를 확인하며, 최근 스마트폰에서 사용하고 있는 안드로이드에 이식하여 실시간성을 제공할 수 있는 여부를 확인하는 연구가 필요하다.

참 고 문 헌

[1] https://estimastory.com/2011/08/20/andreessen/

[2] http://www.epnc.co.kr/news/articleView.html?idxno=47042

[3] Z. He, A. Mok, and C. Peng, “Timed RTOS Modeling for Embedded System Design,” Real Time and Embedded Technology and Applcations Symposium(RTAS), 2005.

[4] 박병률, 맹지찬, 이종범, 유민수, 안현식, 정구민, ″임베디드 S/W 개발을 위한 RTOS API 변환기의 설계 및 구현,″ 대한전기학회 학술대회 논문집, pp.443-445, 2006(10).

[5] C. M. Krishna and Kang G. Shin, *Real-Time Systems*, McGraw-Hill, 1997.

[6] 주민규, 이진욱, 김종진, 조한무, 박영수, 이철훈, “x86 기반의 윈도우 상에서 실시간성 지원 방법,” 한국차세대컴퓨팅학회논문지, 제11권, 제4호, 2011.

[7] 조아라, 송창인, 이철훈, “윈도우즈 상에서 실시간 디바이스 드라이버를 위한 통합 미들웨어,” 한국콘텐츠학회논문지, 제13권, 제3호, 2013.

[8] 박지윤, 조아라, 김효중, 최정현, 허용관, 조한무, 이철훈, “태블릿 PC 환경의 실시간 처리 기능 지원,” 한국콘텐츠학회논문지, 제13권, 제11호, 2013.

[9] https://www.kernel.org/pub/linux/kernel/projects/rt

[10] https://www.rtai.org/

[11] 김주만, 송창인, 이철훈, “RTiK-Linux: 리눅스용 실시간 이식 커널의 설계,” 한국콘텐츠학회논문지, 제11권, 제9호, 2011.

[12] 이상길, 이철훈, “멀티프로세서 기반 리눅스에 실시간성 지원 방안 연구,” 한국콘텐츠학회 종합학술대회 논문집, pp.57-58, 2015(5).

[13] 이상길, 이승율, 이철훈, ″리눅스 사용자 영역에 실시간성 제공을 위한 미들웨어,″ 한국콘텐츠학회논문지, 제16권, 제5호, pp.217-228, 2016(5).

[14] https://rt.wiki.kernel.org/index.php/Main_Page

[15] https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch
[16] http://www.samsung.com/semiconductor/minisite/Exynos/index.html
[17] K. Yaghmour, *Adaptive domain environment for operating systems*, Opersys inc, 2001.
[18] https://xenomai.org/
[19] http://www.cs.kun.nl/J.Hooman/DES/XenomaiExercises/Background.html

저 자 소 개

이 승 율(Seung-Yul Lee) 정회원

- 2015년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2017년 2월 : 충남대학교 컴퓨터공학과 석사과정 졸업

<관심분야> : 실시간 운영체제, 임베디드 시스템

이 상 길(Sang-Gil Lee) 정회원

- 2014년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2016년 2월 : 충남대학교 컴퓨터공학과(공학석사)
- 2016년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 박사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템

이 철 훈(Cheol-Hoon Lee) 정회원

- 1983년 2월 : 서울대학교 전자공학과(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 5월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙 연구원

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