https://doi.org/10.5392/JKCA.2020.20.08.054

리눅스 기반 실시간 성능 제공 RTiK의 이식성 향상을 위한 방법

Methods for Improving Portability of RTiK to Real-time Performance on Linux-based Systems

이상길, 이정국, 이철훈
충남대학교 컴퓨터공학과

Sang-Gil Lee(sk0137@cnu.ac.kr), Jeong-Guk Lee(jeongguk.lee.k@o.cnu.ac.kr), Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

RTiK-Linux는 리눅스에 실시간 성능을 제공하기 위한 방법으로, 타임 틱 인터럽트 구현을 위해 하드웨어 레지스터에 직접 접근하여 제어한다. x86 Intel 및 ARM 기반의 AP 인 Exynoss 5422에서 동작하도록 구현되었으나 파편화된 ARM 환경에 모두 이식할 수 없는 단점이 있었다. 본 논문에서는 다양한 플랫폼에서 동작할 수 있도록 이식성을 개선하기 위해 타임 틱 인터럽트의 구조를 변경한다. 하드웨어와 독립적인 고해상도 타이머를 적용하고, 이를 적용하여 시간 결정성을 만족할 수 있도록 태스크와 이벤트 동작 구조를 변경한다. 개선된 RTiK-Linux가 x86 및 다양한 ARM AP 환경에서 잘 동작하는 것을 확인하였다.

■ 중심어 : | 실시간 운영체제 | RTiK | 실시간 시스템 | 임베디드 시스템 | 범용 운영체제 |

Abstract

RTiK-Linux is a method to provide real-time performance to Linux, it is controlled by directly accessing hardware registers to implement time tick interrupts. It implemented on x86 Intel and ARM based Exynoss 5422, but it had a disadvantage that it could not be ported to both fragmented other platform environments. In this paper, We change structure of time tick interrupt for improve po rtability so that it can operate on other platforms. We apply high-resolution timers that are independent of hardware, and modify operating structure to task and event to satisfy time determinism. It was confirmed that the improved RTiK-Linux works well in x86 and various ARM AP environments.

■ keyword : | RTOS | RTiK | Real-Time System | Embedded System | GPOS |

I. 서 론

임베디드 환경은 기존의 큰 크기의 컴퓨터를 대체하는 다양한 환경에서의 사용을 위해 다양한 제품 개발과 연구가 진행되었다. 최근 하드웨어의 발전으로 성능이 비약적으로 상승하여 높은 성능을 요구하는 제어 시스템이나 감지 시스템이 모바일 환경에서 구현되며, 이에 따라 전용 단말의 소형화가 이루어지고 있으며, 이를 위한 운영체제 및 시스템 소프트웨어의 중요성이 높아지고 있다[1][2].

특히 마이크로프로세서의 성능증가로 인해 사물인터넷(IoT: Internet of Things)이 가능해졌고, 다양한 서비스를 제공하는 제품이 등장하게 되었다[3][4]. 단순한 센서를 사용하여 주변의 정보를 모으는 단계에서 벗어나 최근 스마트홈이라는 개념이 생겨나면서 홈 CCTV, 가스 조절기, 전등 스위치, 멀티탭 및 플러그와 냉장고

접수일자 : 2020년 04월 24일
수정일자 : 2020년 08월 03일
심사완료일 : 2020년 08월 20일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

를 포함한 가전제품 등 사람에게 친숙한 기기를 통해서 더욱 다가가고 있으며, 판매 시장을 넓히고 있다[5-8].

2000년대까지는 현대의 기기보다 연산 성능과 저장 메모리가 매우 낮은 한계를 가지고 있어, 소형 단말에서는 한정된 기능만을 정확하게 수행하기 위한 RTOS(Real Time Operating System, 실시간 운영체제)를 사용하여 기능을 구현했다[9][10]. 실시간 운영체제는 요청된 작업의 정확성과 정확한 수행시간을 만족하게 해야 하는 실시간 성능이 필요한 시스템에서 사용하는 전용 운영체제로 수행 마감 시한을 만족시키지 못할 때, 문제가 발생하는 시스템에서 사용한다[11].

그러나 최근 기술의 발달로 하드웨어의 성능증가로 인해 일부 민감한 시스템을 제외하고 발전된 하드웨어 자원을 바탕으로 다양한 기능을 수행할 수 있는 범용 운영체제인 리눅스로 대체하고 있으며, 이는 사용의 편의성과 다양한 부가 기능을 지원할 수 있는 환경을 통해 개발자의 편의를 안겨주게 되었다. 하지만 다양한 프로그램이 공평하게 수행될 수 있기 위한 범용 운영체제인 리눅스는 실시간 성능을 만족하지 않아 이를 위한 부가적인 방법이 필요하다.

리눅스에 실시간 성능을 제공하는 방법으로 기존 개발된 RTiK(Real-time implant Kernel)은 범용 운영체제인 윈도우와 리눅스 x86 시스템 및 모바일 ARM 프로세서 환경에서의 실시간 성능 제공을 위해 개발되었다[14-16]. 실시간 성능을 위해 CPU 내장 하드웨어 타이머를 사용하여 높은 수준의 실시간 성능을 제공하였으나, 해당 CPU 아키텍처에 종속적이게 되어 다양한 성능 및 환경을 사용하는 모바일 및 임베디드 환경에서 사용을 위해서는 모든 CPU 아키텍처에 대한 최적화가 이루어져야 하는 한계가 있으며 실시간 타이머 및 내부 구조의 개선이 필요하다.

본 논문에서는 특정 플랫폼에 종속되어 있는 단점을 개선하여 다양한 리눅스 플랫폼에 적용 가능하도록 하기 위해 RTiK-Linux를 개선한다. 기존 H/W 타이머에 의존한 실시간 성능 제공 타이머를 최신 리눅스에서 제공하는 HRTimer(High Resolution Timer)를 사용하여 리눅스가 탑재된 모든 임베디드 단말에서 사용할 수 있도록 이식성을 확장한다. 또한 변경한 타이머 구조를 적용하여 RTiK-Linux의 태스크 관리 구조와 실시간 커널 내부 구조를 개선하고 실시간 태스크를 활용하는 방법에 대해 기술한다.

본 논문에서는 제 2장에서 관련 연구로 실시간 리눅스를 설명한다. 제 3장에서는 실시간 타이머의 구조 개선 방안을 설명하고, 제 4장에서는 실시간 타이머가 적용된 RTiK-Linux의 동작 구조를 설명한다. 제 5장에서는 개선된 타이머의 이식성을 평가하기 위해 x86 환경과 모바일 ARM 환경에서 실시간 성능 제공을 확인한 다음 제 6장에서 결론을 맺는다.

II. 관련 연구

1. RTiK-Linux

범용 운영체제인 리눅스에 실시간 성능을 제공하기 위한 연구로 RTiK-Linux가 개발되었다. RTiK-Linux는 실시간 성능을 제공하기 위해 타임 틱 인터럽트를 발생시켜 실시간 태스크의 수행 성능을 만족시킨다[14-16].

RTiK-Linux는 x86 프로세서에서 제공되는 Local APIC Timer를 사용하여 실시간 성능을 제공하는 타이머를 구현하였다[14]. 그러나 해당 하드웨어 인터럽트는 리눅스에서 시스템 자원을 위해 사용 중이므로 리눅스 커널과 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 처리 함수를 수행하도록 하여

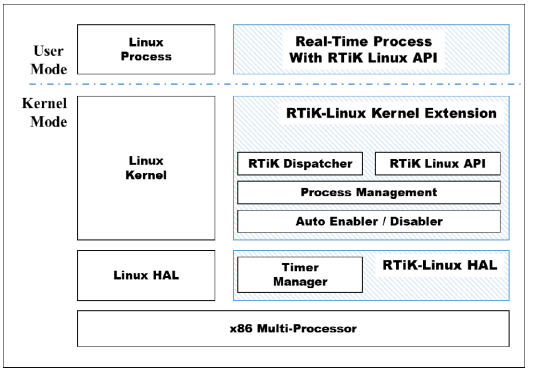

그림 1. RTiK-Linux의 구조

상호 충돌을 방지할 수 있게 구현되어 있다. 이러한 구조는 Local APIC Timer가 제공되는 인텔 프로세서만 사용 가능한 구조로 적용된다[14].

ARM 프로세서 기반의 모바일에서 사용하지 못하기 때문에 이를 개선하여 ARM 칩셋 기반의 Exynoss5422에서 동작하는 RTiKA이 연구되어 모바일 환경에서도 실시간 성능을 제공할 수 있도록 하였으나[15], 다양한 아키텍처를 사용하는 모바일 환경에서는 타겟 플랫폼 마다 타이머를 새로 분석해야 하는 단점이 있기 때문에 다양한 기기에서 사용이 불가능하여 이식성이 떨어진다는 단점이 있다.

이에 따라 다양한 모바일 환경에서 적용할 수 있도록 이식성 향상을 위한 연구가 필요하다.

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 실시간 성능을 제공한다. 하드웨어와 리눅스 커널 사이에 실시간 커널이 추가된 구조로 기존 리눅스 커널을 우선순위가 낮은 실시간 태스크로 관리하며, 우선순위가 높게 설정된 실시간 태스크들이 시간적 제약을 만족시켜 실시간성을 보장받게 된다[21].

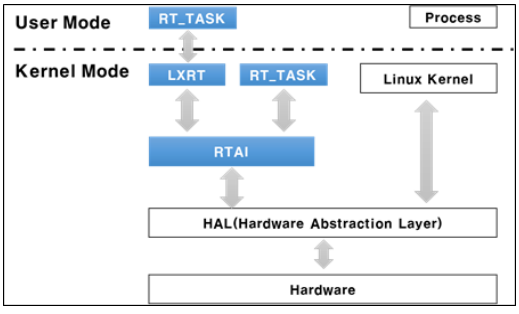

그림 2. RTAI의 동작 구조

RTAI는 x86, x86_64, PowerPC, ARM(StrongARM, ARM7, clps711x-family, Cirrus Logic EP7xxx, CS89712, PXA25x), m68k을 지원하고 있다. ARMv7 및 ARMv8 아키텍처는 지원하지 않는 단점이 있으며, 최근에는 Xenomai가 대체되어 사용되고 있다[26-28].

3. Xenomai

Xenomai는 리눅스 커널을 위한 실시간 개발용 프레임워크로, 유저 영역에 응용 프로그램에 대한 경성 실시간 성능을 제공한다[19]. Xenomai는 2001년 시작되어 2003년 RTAI와 함께 상용 등급의 실시간 성능을 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[20].

Xenomai는 ADEOS(Adaptive Domain Environment for Operating Systems)를 이용한 I-pipe를 통해 인터럽트 가상화를 제공한다. 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[21].

Xenomai는 I-pipe를 통해 ARM[22], ARM64[23], PPC32[24], x86[25] 계열의 아키텍처를 지원하고 있으며 현재 커널 4.14 버전에서 동작이 확인되고 있으며, 지속적인 패치를 통해 다양한 플랫폼에 이식될 수 있도록 제공하고 있다.

RTiK-Linux에서 Xenomai 와 유사한 이식성 지원을 제공하기 위한 이식성 향상을 위한 연구가 필요하다.

III. 이식성 향상을 위한 실시간 타이머의 설계

본 장에서는 기존 하드웨어 타이머에 종속되어 사용하던 RTiK-Linux의 이식성 향상을 위해 리눅스 커널에서 제공하는 고성능 타이머를 이용하여 개선된 실시간 타이머 모듈을 설계 및 구현하고, RTiK-Linux에 등록된 실시간 태스크가 유기적으로 동작할 수 있도록

하는 스케줄링 알고리즘을 설명한다.

1. 실시간 타이머 개선을 위한 방법

과거 리눅스 시스템에서 제공하는 타이머의 해상도가 낮아서 ms 단위 혹은 그 이하의 매우 높은 정확도를 요구하는 실시간 성능을 만족할 수 없었다. 리눅스 커널이 지속적으로 발전함에 따라서 추가된 자료구조인 HRTimer는 리눅스 커널 v2.6.21에서 메인 커널 버전에 추가된 커널 타이머로 이론상으로 1ns 단위의 고해상도 타이머를 관리할 수 있도록 제공하고 있다.

이는 기존의 jiffies 기반의 loweres 타이머를 사용하여 구현되어 Hz 기반 Tick Time에 의해 해상도가 수 ms ~ 수십 ms의 낮은 타이머 해상도만을 관리할 수 있었던 것을 보완한다.

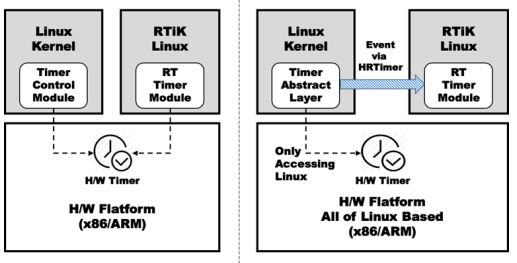

그림 3. H/W 타이머를 각각 직접 접근하여 제어하는 구조(좌)와 리눅스를 통해 타이머를 간접 접근하는 구조(우)

[그림 3]는 기존의 실시간 타이머를 통해 H/W 타이머에 접근하는 모습(좌)와 변경된 구조를 통해서 리눅스에서 제공하는 HRTimer에 등록하여 사용하는 실시간 타이머를 비교한 모습이다.

리눅스에서 제공하는 HRTimer를 통해 실시간 성능을 제공하는 개선된 구조의 RTiK-Linux 시스템에서는 고정밀 커널 타이머를 사용하기 때문에 사용자가 원하는 주기에 대응할 수 있도록 100us ~ 500us 단위 혹은 그 이상의 실시간 타이머를 위한 타임 틱 인터럽트를 발생시키기 위한 HRTimer에 대응하는 실시간 타이머용 핸들러를 등록시킨다. 또한 RTiK-Linux에서는 시스템의 하드웨어 타이머의 접근을 직접 하지 않기 때문에 리눅스와 타이머 혼선으로 인한 시스템 크러시를 방지할 수 있으며, Linux가 동작하는 모든 임베디드 시스템 플랫폼에 이를 바로 이식하여 적용할 수 있게 된다.

2. 개선된 실시간 타이머를 통한 타임 틱 인터럽트 동작

앞선 과정을 통해 실시간 타이머로 사용될 HRTimer 핸들러인 “Time Tick ISR”를 등록하는 과정을 마치게 되면, 실시간 성능이 제공되는 타이머로 사용될 준비를 마친다. 실시간 타이머가 동작하게 되는 과정은 [그림 4]로 보여줄 수 있다.

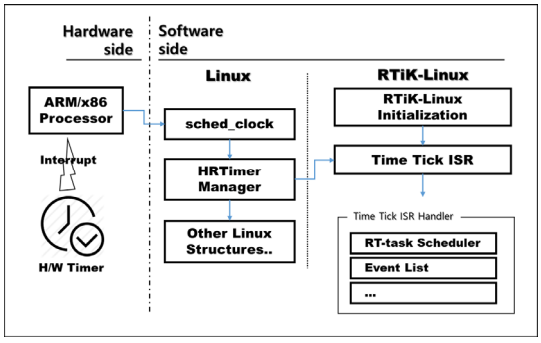

그림 4. 이식성이 개선된 구조에서 RTiK-Linux의 타이머 호출 과정

기존의 하드웨어 타이머를 직접 제어하던 구조를 변경하여 타이머를 HRTimer에 접근하는 모듈이 사용된다. 개선된 실시간 타이머 모듈에서는 사용자나 프로그램 설정 값을 통해서 입력받은 주기의 실시간 타이머를 등록한다. HRTimer를 통해서 등록한 실시간 타이머는 하드웨어 타이머 인터럽트를 통해서 Linux OS Level에서 표준화된 Timer Subsystem을 통해서 처리되며, 처리 후에 “sched_clock” 과정을 통해 시스템에 등록된 리눅스 타임 이벤트를 발생시킨다.

리눅스 타임 이벤트 중 하나로 등록되어 있는 HRTimer 관리자가 호출되면, RTiK-Linux의 주기 제어를 위해 앞서 등록된 RTiK-Linux의 “Time Tick ISR Handler”가 수행된다. 본 과정은 RTOS의 동작 기준 주기가 되는 타임 틱 인터럽트로 동작함으로써 RTiK-Linux에 정확한 실시간 성능을 제공할 수 있도

록 한다.

이런 과정을 통해 RTiK-Linux에서 사용되는 실시간 태스크 스케줄러를 위한 기본적인 시스템 함수들이 실행된다. “Event List”는 실시간 시스템에서 태스크간의 동기화를 위해 필요한 세마포어와 같은 자료구조나 태스크 간의 데이터 전달을 위해 사용되는 메시지 메일박스나 메시지 큐 등의 자료구조에 시간 결정성을 제공하기 위해 관리되는 리스트로 다음 장에서 자세히 설명한다.

“RT-task Scheduler”는 실행되며 사용자가 등록한 실시간 태스크를 주기에 맞게 호출해 주는 역할을 수행한다. 이때, 시간 결정성을 만족하기 위해서 모든 실시간 태스크는 선점 가능해야 하며, 각 태스크 간의 우선순위로 구분되어야 한다. 이와 같은 태스크의 스케줄링 정책은 다음과 같다.

3. 실시간 태스크 관리를 위한 스케줄러의 우선순위 정책

실시간 타이머를 통해 호출되는 RTiK-Linux의 실시간 태스크 스케줄러는 리눅스와 함께 동작하는 RTiK-Linux의 실시간 성능을 만족하기 위해 우선순위에 따른 태스크 선점이 필요하다. 이에 따라서 태스크의 우선순위를 정할 필요가 있으며, 본 논문에서는 [그림 5]로 나타낼 수 있다.

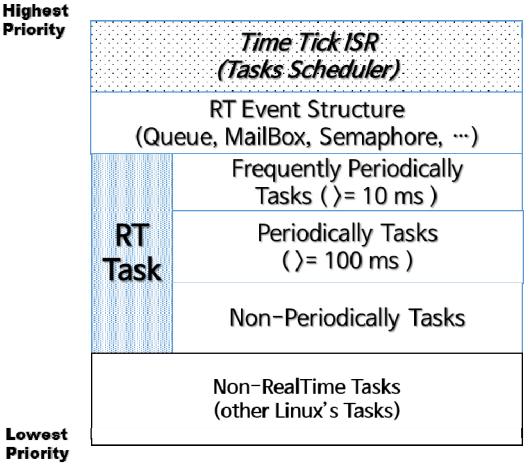

그림 5. RTiK-Linux의 태스크/이벤트 우선순위

실시간 태스크 스케줄러를 통해 관리되는 우선순위는 가장 낮은 “Non Real-Time Task”부터 “Time Tick ISR”로 호출되는 스케줄러로 구분할 수 있다. 하드웨어 인터럽트와 리눅스에서 사용되는 외부 인터럽트는 우선순위 산정에서 제외하도록 한다.

가장 낮은 우선순위는 리눅스에서 실행하고 있는 태스크로 실시간 성능이 필요치 않은 태스크를 말한다. 대부분의 리눅스 태스크는 본 레벨에서 동작하며, 실시간 성능이 필요한 태스크의 실행을 선점하지 못하도록 가장 낮은 우선순위를 사용한다. 이런 비실시간 태스크는 일반 리눅스의 우선순위를 통해서 리눅스의 태스크 스케줄링에 영향을 받게 된다. RTiK-Linux의 스케줄러에서는 모든 실시간 태스크가 수행된 이후에 이런 비실시간 태스크가 동작하도록 한다.

다음으로 실시간 태스크로 등록되었으나, 태스크의 호출 주기가 없는 “Non-Periodically RT Task”가 있다. 이 태스크는 특정하게 정한 실행 주기가 없으므로 주기가 있는 태스크에 선점당하지만, 일반 비실시간 태스크보다 실행 우선순위가 높으므로 실시간 태스크가 동작하지 않은 Slack Time에 수행되도록 스케줄링 되고 있다.

RTiK-Linux 에서는 실시간 태스크 스케줄리 알고리즘 중 잦은 호출 주기를 가지는 태스크가 높은 우선순위를 갖도록 하는 RM(Rate-Monotonic) 스케줄링 기반의 알고리즘을 사용한다. 가령, 주기를 가지는 태스크 셋 T={T1=2ms, T2=5ms, T3=100ms}의 호출 주기를 가지는 태스크가 있을 때, 태스크 T3가 1회 호출될 때, 태스크 T1은 50회 호출된다. 이에 따라서 실행 빈도가 높은 것은 T1이기 때문에 더 높은 우선순위를 갖도록 태스크 스케줄링 된다.

이에 따라서 태스크를 리스트로 관리하고 있으며, 빠른 접근을 위해 “Frequently Periodically RT Tasks”와 그보다 낮은 호출 주기를 갖는 “Periodically RT Tasks”로 구분하여 태스크를 관리하고 있다.

마지막으로 실시간 타이머를 이용하는 스케줄러가 가장 높은 우선순위를 갖는다. 스캐줄러는 어떤 태스크가 수행되고 있다고 하더라도, 더 높은 태스크를 수행하도록 항상 더 높은 우선순위를 가진다.

RTiK-Linux에서는 위와 같은 실시간 태스크 스케줄

러를 사용하여 원하는 기능을 수행할 수 있도록 한다. 다음으로 실시간 태스크의 호출 구조를 설명한다.

Ⅳ. 실시간 타이머가 적용된 RTiK-Linux 동작 과정

RTiK-Linux를 통해 리눅스 기반 시스템에서 실시간 성능을 제공하기 위해 실시간 태스크 및 태스크 동기화와 태스크 간의 통신(Inter-Task Communication, ITC)을 관리하기 위한 실시간 커널의 스케줄러가 호출된다. 본 장에서는 개선된 실시간 타이머가 호출되며 발생하는 실시간 태스크의 수행 과정을 설명한다.

1. 태스크 동기화 및 태스크 간의 통신의 타임아웃 처리

실시간 운영체제에서는 태스크 간의 데이터 전송이나 태스크 수행 순서의 동기화를 위해 세마포어, 메시지 큐, 메지시 메일박스와 같은 자료구조를 이용하고 있다[29].

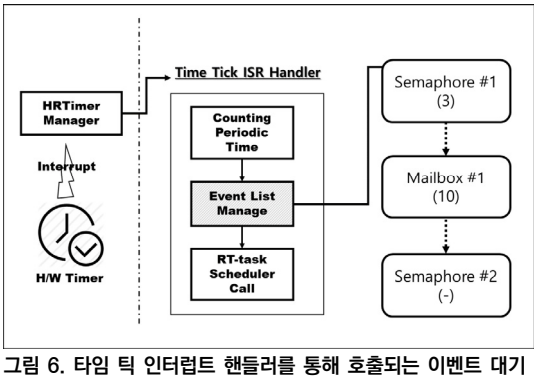

그림 6. 타임 틱 인터럽트 핸들러를 통해 호출되는 이벤트 대기 리스트 관리 구조

변경된 타이머를 통해 호출되는 타임 틱 인터럽트는 실시간 시스템에서 심장박동의 역할을 하며, 타임 틱 인터럽트를 통해 호출되는 핸들러에서는 태스크의 수행을 결정하기 이전에 각 자료구조에서 대기하고 있는 마감 시한을 관리한다. [그림 6]은 RTiK-Linux의 스케줄러에서 세마포어를 포함한 이벤트 자료구조의 마감 시한 관리를 위한 리스트를 구성하며 이를 델타 프로세싱을 통해 이벤트를 기다리고 있는 태스크의 마감 시한을 관리한다[30].

[그림 6]에서의 “Semaphore #1 (3)”을 보면, RTiK-Linux에서 실시간 태스크가 세마포어를 통해 특정 이벤트를 기다리게 되면, 태스크의 마감시한을 지켜야 하기 때문에 타임아웃 시간을 가진다. 이때 [그림 6]에서 해당 세마포어를 3의 클럭 틱 시간 동안 기다린다는 의미로, 타임 틱 인터럽트가 세 번 호출될 때까지를 마감시한으로 지정한다. 타임 틱 인터럽트 핸들러가 호출되면, 리스트의 첫 번째 자료구조의 대기 중인 시간을 차감하고, 해당 값이 0이 되었을 때 이벤트에 대기중인 태스크에 신호를 보내서 다시 수행 가능한 상태로 변경해 준다. 이는 정상적으로 이벤트를 받지 못했기 때문에 실시간 태스크는 타임아웃 상태의 처리를 해주게 된다.

이때 델타 프로세싱을 사용하여, 리스트 내부의 모든 이벤트의 대기 시간을 줄이는 것이 아닌, 리스트의 선두의 카운트 값만 차감시켜, 타임 틱 인터럽트 핸들러의 수행 시간 오버헤드를 감소해준다. 그 외에도 “(-)”로 표기된 이벤트의 경우, 특별하게 관리되는 제한 시간이 없는 이벤트 대기 상태를 나타내고 있다.

2. 태스크 수행을 위한 방법

앞 절에서 설명한 이벤트 자료구조에 대한 대기 시간 관리를 통해서 태스크의 동기화나 태스크 간 통신이 원활하게 될 수 있도록 하며, 위와 같은 과정이 실행된 이후에 실시간 태스크가 수행될 수 있도록 실시간 태스크 스케줄러를 호출한다. 실시간 태스크 스케줄러가 호출되면, RM 알고리즘을 사용하여 호출되는 태스크를 선택한다. 주기를 바탕으로 태스크를 선점하여 호출하기 위해, 주기를 바탕으로 태스크를 호출되는 주기에 맞추어 선택될 수 있도록 하는 자료구조를 활용한다[31].

[그림 7]은 RTiK-Linux에 등록된 실시간 태스크와 Slack Time을 계산하기 위해 태스크의 호출 주기에 맞는 테이블 인덱스를 가지는 구조를 사용하여, 타임 틱 인터럽트가 수행될 때 마다 오버헤드를 줄이기 위해 테이블 형태로 구성된 모습을 보이고 있다.

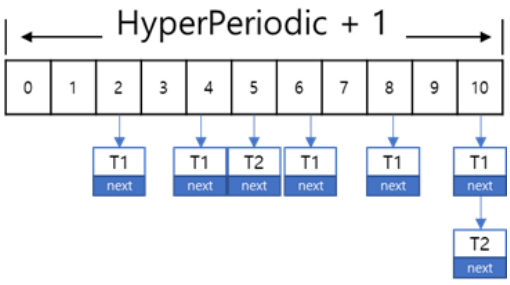

그림 7. 호출 주기에 맞게 등록된 태스크의 모습

[그림 7]의 HyperPeriodic은 등록된 태스크의 주기를 최소 공배수로 계산하여 주기의 최대 값으로 제한하고 테이블이 무한히 커지는 것을 방지한다. [그림 7]은 2의 주기를 갖는 태스크 T1과 5의 주기를 갖는 태스크 T2를 전개한 모습이다. HyperPeriodic은 두 태스크의 최소 공배수인 10으로 정하고, 0의 인덱스를 비우기 위해 HyperPeriodic +1의 크기로 테이블을 정하여 구성한다. 태스크의 호출 과정은 다음과 같은 흐름으로 표기할 수 있다.

Algorithm. Task Scheduling via Periodic Table

```
Time_Tick_ISR Occur
Tick_Count Increase
Check Pariodic Table by Tick_Count
Ti = Pt[Tick_Count]
IF Ti = NULL then Goto (6)
while Ti ≠ empty do
    Ti.task request execution
    Ti = Ti.next
end
IF Tick Count ≥ Hyperperiod then
Tick_Count = 0;
```

(1) 타임 틱 인터럽트가 호출되면 테이블을 조회하기 위한 과정으로 진입하여, (2) 시스템의 글로벌 Tick Count를 증가시켜서 (3) 호출되는 태스크를 조회한다. (4) 이때, 해당 테이블 인덱스에 등록된 태스크가 없을 경우, 과정을 건너뛰게 된다. (5) 태스크가 등록되어 있는 경우 테이블에 연결된 태스크의 정보를 통해 해당 태스크를 수행한다. (6) 마지막으로 글로벌 Tick Count를 제한하기 위해 값을 초기화 한다. 이를 통해서 타임 틱 인터럽트 핸들러 호출될 때의 지연 시간이 감소되며 태스크를 호출할 수 있게 해준다.

3. 태스크 실행의 예

```
// user handle proto type
int userHandle ( int, int );

#define  rtik_errorstring(...) "Undefined Error"

int main ( void )
{
   int tid = -1;
   int result = -1;

   printf("userHandle's addr:::: %p₩n", userHandle);

   result = rtik_taskCreate ( userHandle, 1, 0, 0 );
   if ( result < 0 ) {
      printf(" ERROR: rtik_taskCreate Fail with %d, %s₩n"
      , result, rtik_errorstring(result));
      exit (1);
   }
   tid = result;

   result = rtik_taskExitWait   ( tid ) ;
   if ( result < 0 ) {
      printf(" Error: rtik_taskExitWait failed with %d, %s₩n"
      , result, rtik_errorstring(result));
   }
   else {
      printf("Successfully Execute via rtik-api module₩n");
   }

   return 0;
}


int userHandle(int arg1, int arg2)
{
   printf("user Handle example ₩n");

   return 0;
}
```

그림 8. RTiK-Linux를 사용하여 태스크를 생성한 후, 호출하는 예시 코드

[그림 8]은 RTiK-Linux을 사용하여 태스크를 생성하는 예시 코드로 사용자가 원하는 수행 함수인 “userHandle(int arg1, int arg2)”를 작성하여 “rtik_taskCreate()” 함수를 호출할 때 인자로 넘겨주면서 태스크를 생성할 수 있다. 등록된 주기마다 사용자가 지정한 함수를 반복해서 수행할 수 있게 해주며, 제공된 RTiK-Linux을 제어할 수 있는 API를 사용하는 것으로 실시간 성능이 제공되는 사용자 영역에 태스크를 통해 원하는 기능을 수행할 수 있게 된다.

V. 성능 측정

1. 실험 환경

본 논문에서 개선된 RTiK-Linux의 여러 플랫폼으로서의 대응을 위한 이식성을 평가하기 위해, 개선된 실시간 타이머가 적용된 RTiK-Linux을 x86 플랫폼과 ARM 플랫폼에 적용한 후 실험을 진행하였다.

본 실험에서는 두 개의 시스템에서 동일한 커널 버전으로 대상으로 진행하였으며, 동일한 리눅스 코어인 우분투 운영체제를 사용하였으나, 세부 버전이 약간 다르게 적용되었다.

표 1. 이식성 평가를 위한 대상 하드웨어 사양

x86_64	대상 아키텍처	ARM Open-Q 820
3.10.105	Kernel Version	3.10.83
i7-2600	Processor	Snapdragon 820 APQ8086
Personal Computer	Type	Embedded Board
ubuntu 14.04	OS Version	ubuntu 16.04

[표 1]은 이식성 평가를 위한 하드웨어의 성능을 나타낸 표로, 두 개의 아키텍처에서 사용하는 프로세서와 타입을 나타낸다. PC 플랫폼에서 사용하는 x86_64 아키텍처는 인텔 프로세서를 탑재한 시스템에서 측정하였으며, 임베디드 프로세서는 스냅드래곤 820을 사용하는 Open-Q 820 보드에서 측정을 진행하였다.

실시간 커널의 실시간 성능에 대한 중요한 지표로 지정된 주기에 맞추어 태스크가 호출되어야 한다. 실시간 시스템은 여러 주기를 가질 수 있으며, 고정밀 시스템의 경우 1ms 이하의 주기를 갖기도 하며, 일반적인 응용 프로그램은 20ms 혹은 그 이상의 주기를 바탕으로 실시간 태스크를 구현한다.

2. 실험 결과

본 논문에서는 RTiK-Linux를 통한 실험은 사용자 레벨의 태스크와 커널 레벨 태스크에서 1ms 주기의 태스크를 작성하여 실험을 진행하였다. 정확한 시간 측정을 위한 clock count api를 사용하여 시간을 측정하였으며, 각 실험은 10,000 회 반복을 통해 다수의 데이터를 통해 측정하였다.

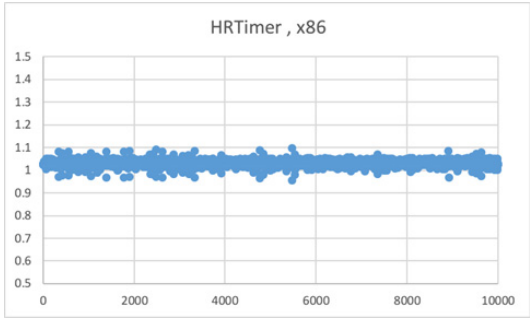

그림 9. 개선된 실시간 타이머가 적용된 RTiK을 x86 플랫폼에서 측정한 결과

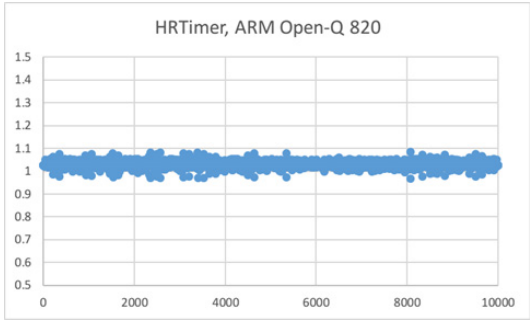

그림 10. 개선된 실시간 타이머가 적용된 RTiK을 ARM Open-Q 820 플랫폼에서 측정한 결과

[그림 9]와 [그림 10]은 개선된 실시간 타이머가 적용된 RTiK-Linux을 x86 플랫폼과 ARM 플랫폼에서 측정한 결과를 나타낸 그래프이다. 그래프에서 x 축은 주기 측정 횟수를 나타내며, y축은 주기 값을 나타낸다. 그래프가 기준 값에 가깝게 나타날수록 오차가 적다는 것을 보일 수 있다.

표의 데이터는 측정된 주기의 최솟값과 최댓값, 오차율을 나타낼 수 있도록 하며, 기존의 논문에서의 데이터와 비교하는 것으로 실시간 타이머의 변경이 성능에 영향을 미치지 않고 이식성을 향상되었다는 것을 측정하였다.

표 2. 1ms 주기 측정 결과

구분	변경된 HRTimer 사용된 RTiK-Linux		기존 HW Timer 사용		Xenomai를 사용한 실시간 성능 측정 결과[17]	
종류	x86	ARM	x86 RTiK[15]	ARM RTiKA[16]	Beagle Bone Black User Mode[17]	SABRE Lite User Mode[17]
최댓값	1.096 ms	1.085 ms	1.023 ms	1.070 ms	1.032 ms	1.008 ms
최솟값	0.955 ms	0.967 ms	0.987 ms	0.950 ms	0.893 ms	0.993 ms
평균값	1.026 ms	1.026 ms	1.000 ms	1.000 ms	1.000 ms	1.000 ms

실험으로 1ms 주기 태스크에 대한 성능 검증을 수행하였다. 성능 측정된 결과는 [그림 9]과 [그림 10]로 나타낼 수 있다. [그림 9]은 HRTimer를 적용하여 x86에서 적용한 실험 결과를 나타내고, [그림 10]는 HRTimer가 적용된 ARM 플랫폼에서 적용한 실험 결과를 나타낸다. y축의 눈금 단위는 0.1ms로 기준 주기에서 10%의 성능 단위로 구분할 수 있도록 나타내었다. 측정된 주기 값이 대부분 0.1ms 오차 이내로 측정된 것을 확인할 수 있었으며, 모든 태스크의 수행 주기가 매우 안정적인 것을 볼 수 있었다. 또한 x86 플랫폼과 ARM 플랫폼에서도 모두 값이 안정적으로 측정되는 것을 확인할 수 있는데, 이는 HRTimer를 적용하였을 때, 기존 하드웨어 타이머를 사용하는 것만큼의 안정된 성능을 제공하는 것을 확인할 수 있었으며, 이식성 향상이 이루어졌다는 것을 검증할 수 있었다.

또한 기존의 연구에서 수행한 실시간 성능 측정 결과와 비교를 위해 Xenomai를 이용한 Beagle Bone Black 보드와 SABRE Lite 보드에서 수행한 결과와 비교한다[17]. 해당 연구에서 측정한 사용자 영역의 1ms 주기의 실시간 성능은 1ms를 바탕으로 Beagle Bone Black 보드에서 0.893 ~ 1.032 사이로 측정되고 있으며, SABRE Lite 보드에서 0.993 ~ 1.008 사이로 측정되고 있는 것으로 성능이 측정되었고 이는 본 논문에서 연구한 변경된 타이머를 적용한 RTiK-Linux에서 성능 측정한 x86 환경에서의 0.955 ~ 1.096과 ARM 환경에서의 0.967 ~ 1.085 와 유사하게 측정된 것을 볼 수 있다. 이를 통해서 다른 실시간 리눅스가 적용된 환경과 유사한 성능을 보이는 것을 검증할 수 있었으며, 이식성이 향상된 RTiK-Linux을 통해서도 리눅스 환경에서 실시간 성능을 제공할 수 있는 것을 확인하였다.

VI. 결론 및 향후 연구

본 논문에서는 기존에 ARM 플랫폼과 x86 프로세서 상에서 동작하는 리눅스 환경의 이식성이 한정된 RTiK-Linux에 다양한 플랫폼에서 동작할 수 있도록 기존의 실시간 타이머로 사용하던 H/W 타이머를 리눅스에서 고성능으로 제공하는 HRTimer를 사용하는 방식으로 개선하여 실시간 타이머를 구현하였고, 이를 통해 리눅스에서 실시간 성능을 제공받을 수 있도록 실시간 태스크를 사용할 수 있는 자료구조를 제공하며, 실시간 태스크가 주기에 맞추어 실행될 수 있는 구조를 보였다.

구현된 HRTimer가 적용된 RTiK-Linux의 성능을 검증하기 위해 주기에 발생하는 Tick 값을 측정하여 이전 값과의 비교를 통해 주기에 응답한 시간을 측정하였다. 각 10,000 회의 주기를 측정하여 1ms를 바탕으로 실행하였을 경우, 오차범위 5%이내에서 동작하여 1ms 단위로 실시간 태스크의 주기를 설정할 수 있도록 하는 것을 검증하였다. 또한 두 개 이상의 플랫폼에 적용하였을 때, 성능 만족 여부를 확인하기 위해 ARM 프로세서를 사용하는 임베디드 환경과 x86 시스템을 사용하는 환경에서 각각 측정하여 실시간 성능을 안정적으로 제공하는 것을 검증하였다.

향후 연구과제로는 ARM 프로세서를 사용하고 리눅스 커널을 공유하는 휴대용 모바일 기기의 안드로이드 운영체제에 실시간 성능을 제공하는 연구가 필요하다. 이를 통해 범용 운영체제에서 리눅스 커널을 공유하는 환경에서 실시간 성능을 제공할 수 있는 여부를 확인하는 것이 필요하다.

참 고 문 헌

[1] http://www.epnc.co.kr/news/articleView.html?idxno=47042, 2020.4.13.

[2] https://estimastory.com/2011/08/20/andreessen/, 2020.4.13.

[3] J. A. Stankovic, “Research directions for the internet of things,” IEEE Internet of Things Journal, Vol.1, No.1, pp.3-9, 2014(2).

[4] SK telecom Smart Home, https://www.sktsmarthome.com, 2020.4.13.

[5] Infineon Smart Home, https://www.infineon.com/cms/kr/discoveries/smart-home-basics/, 2020.4.13.

[6] KT Smart Home, https://product.kt.com/wDic/index.do?CateCode=6018, 2020.4.13.

[7] LG Smart Home, https://social.lge.co.kr/tag/%EC%8A%A4%EB%A7%88%ED%8A%B8%ED%99%88/, 2020.4.13.
[8] Z. He, A. Mok, and C. Peng, "Timed RTOS Modeling for Embedded System Design," Real Time and Embedded Technology and Applcations Symposium(RTAS), 2005.
[9] 박병률, 맹지찬, 이종범, 유민수, 안현식, 정구민, "RTOS기반 임베디드 S/W를 위한 API 정변환/역변환기의 개발," 대한전기학회 학술대회 논문집, pp.187-189, 2007.
[10] 주민규, 이진욱, 김종진, 조한무, 박영수, 이철훈, "x86 기반의 윈도우즈 상에서 실시간성 지원 방법," 한국차세대컴퓨팅학회 논문지, 제7권, 제4호, pp.47-58, 2011.
[12] 조아라, 송창인, 이철훈, "윈도우즈 상에서 실시간 디바이스 드라이버를 위한 통합 미들웨어," 한국콘텐츠학회논문지, 제13권, 제3호, pp.22-31, 2013.
[13] 박지윤, 조아라, 김효중, 최정현, 허용관, 조한무, 이철훈, "태블릿 PC 환경의 실시간 처리 기능 지원," 한국콘텐츠학회논문지, 제13권, 제11호, pp.541-550, 2013.
[14] 김주만, 송창인, 이철훈, "리눅스용 실시간 이식 커널의 설계," 한국콘텐츠학회논문지, 제11권, 제9호, pp.45-53, 2011.
[15] 이상길, 이승율, 이철훈, "리눅스 사용자 영역에 실시간성 제공을 위한 미들웨어", 한국콘텐츠학회논문지, 제16권, 제5호, pp.217-228, 2016.
[16] 이승율, 이상길, 이철훈, "ARM 프로세서 기반의 리눅스를 위한 실시간 확장 커널," 한국콘텐츠학회논문지, 제17권, 제10호, pp.587-597, 2017.
[17] 신욱철, 최병욱, "공개 임베디드 하드웨어의 실시간 메커니즘 성능 분석," 제어로봇시스템학회 논문지, 제23권, 제1호, pp.60-66, 2017.
[18] J. H. Koh and B. W. Choi, "Performance Evaluation of Real-time Mechanisms for Real-time Embedded Linux," Journal of Institute of Control, Robotics and Systems, Vol.18, No.4, pp.337-342, 2012.
[19] https://xenomai.org/, 2020.7.10.
[20] http://www.cs.kun.nl/J.Hooman/DES/XenomaiExercise/Background.html, 2020.7.10.
[21] https://www.rtai.org/, 2020.7.10.
[22] https://gitlab.denx.de/Xenomai/ipipe-arm, 2020.7.10.
[23] https://gitlab.denx.de/Xenomai/ipipe-arm64, 2020.7.10.
[24] https://gitlab.denx.de/Xenomai/ipipe-ppc32, 2020.7.10.
[25] https://gitlab.denx.de/Xenomai/ipipe-x86, 2020.7.10.
[26] L. Muratore, A. Laurenzi, E. Mingo Hoffman, and N. Tsagarakis, *The XBot Real-Time Software Framework for Robotics: From the Developer to the User Perspective*, IEEE Robotics & Automation Magazine, 2020. doi: 10.1109/MRA.2020.2979954.
[27] 손태영, 임성락, "WSN을 위한 Xenomai의 실험적 성능평가," 한국산학기술학회 논문지, 제18권, 제1호, pp.709-714, 2017.
[28] 길진수, 곽대식, 신제호, 최기수, 김근연, "특수목적용 기계를 위한 실시간 제어기 S/W 설계 및 구현에 관한 연구," 한국정밀공학회 학술발표대회 논문집, pp.862-863, 2017.
[29] C. M. Krishna and Kang G. Shin, *Real-Time Systems*, McGraw-Hill, 1997.
[30] 고영환, 박세영, 이철훈, "실시간 운영체제 uC/OSII를 위한 멜타 프로세싱 설계 및 구현," 한국콘텐츠학회 종합학술대회 논문집, pp.25-26, 2012.
[31] 이상길, 이철훈, "실시간 태스크 관리를 위한 주기 기반의 테이블 자료구조 연구," 한국통신학회 학술대회 논문집, pp.1600-1601, 2017.

저 자 소 개

이 상 길(Sang-Gil Lee) 정회원

- 2014년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2016년 2월 : 충남대학교 컴퓨터공학과(공학석사)
- 2018년 2월 : 충남대학교 컴퓨터공학과 박사과정 수료

〈관심분야〉 : 실시간 운영체제, 임베디드 시스템

이 정 국(Jeong-Guk Lee) 정회원

- 2011년 2월 : 인하대학교 정보통신공학과(공학사)
- 2012년 1월 ~ 2015년 12월 : LIG넥스원 연구원
- 2016년 2월 ~ 2018년 4월 : LIG 넥스원 선임연구원
- 2019년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

〈관심분야〉 : 임베디드 시스템, 실시간 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 객원연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙연구원

〈관심분야〉 : 실시간 시스템, 운영체제, 고장허용 컴퓨팅, 로봇 미들웨어