Journal of Institute of Control, Robotics and Systems (2012) 18(4):337-342
http://dx.doi.org/10.5302/J.ICROS.2012.18.4.337 ISSN:1976-5622 eISSN:2233-4335

실시간 임베디드 리눅스의 실시간 메커니즘 성능 분석

Performance Evaluation of Real-time Mechanisms for Real-time Embedded Linux

고 재 환, 최 병 욱*
(Jae-Hwan Koh[1] and Byoung-Wook Choi[1])
[1]Seoul National University of Science and Technology

Abstract: This paper presents performance evaluation of real-time mechanisms for real-time embedded linux. First, we presents process for implementing open-source real-time embedded linux namely RTAI and Xenomai. These are real-time extensions to linux kernel and we implemented real-time embedded linux over the latest linux kernel. Measurements of executions of real-time mechanisms for each distribution are performed to give a quantitative comparison. Performance evaluations are conducted in kernel space about repeatability of periodic task, response time of Semaphore, FIFO, Mailbox and Message queue in terms of inter-task communication for each distribution. These rules can be helpful for deciding which real-time linux extension should be used with respect to the requirements of the real-time applications.

Keywords: real-time embedded linux, RTAI, Xenomai, real-time mechanism, ITC

I. 서론

컴퓨터와 하드웨어의 발전은 일상생활에서 사용 가능한 새로운 제어장치와 정보기기의 발전을 가져왔다. 그러나 사용자 요구가 증가하고 시스템이 복잡해짐에 따라 운영체제의 도움을 필요로 하고 있다. 마찬가지로 지능형 서비스 로봇도 외부 환경을 인식하고 지능적 판단을 하여 원하는 동작을 수행하기 위하여 복잡한 소프트웨어 계층을 필요로 한다. 따라서 운영체제를 기반으로 제어장치의 소프트웨어를 개발하는 것은 일반화되고 있다[1].

지능형 서비스 로봇은 인간과 환경의 유기적인 처리를 위하여 실시간 제어가 필요하다. 이와 같은 이유로 범용 운영체제인 윈도우즈와 리눅스 보다는 실시간 운영체제인 RTOS (Real-time Operating System)가 많이 사용되어진다. 상용 RTOS인 VxWorks, Nucleus PLUS, QNX 및 LynxOS 등은 실시간성에는 유용하나 특정 운영체제로의 기술 종속, 고가의 초기 개발 비용이라는 단점을 갖는다[2].

상용 RTOS의 단점을 보완하기 위하여 범용의 운영체제를 이용하여 실시간성을 보장하기 위해 리눅스 진영에서 다양한 노력이 진행되었다. 실시간 임베디드 리눅스 구현 방법은 리눅스 커널의 스케줄러를 수정하여 정밀한 타이머와 추가적인 쓰레드 동기화 메커니즘을 제공하는 방법과 다중 운영체제 지원 방법을 활용하여 실시간성을 보장하는 방법의 두 가지가 있다[1,3-5]. 전자는 주로 상용화된 제품으로 출시되고 있으며 후자의 경우는 오픈소스 프로젝트(open-source project)로 진행되고 있다[6]. 대표적인 오픈소스 프로젝트는 RTLinux에서 시작되었으나 상용화가 이루어지면서 오픈 소스 진영에서 탈퇴하였다. 현재는 RTAI (the Real-time Application Interface for Linux)와 Xenomai가 대표적인 오픈 소스 프로젝트로서 리눅스의 실시간 확장 커널이다[7,8].

RTAI는 1996년 이탈리아의 DIAPM (Dipartimento di Ingegneria Aerospaziale Politecnico di Milano) 연구소의 Paolo Mantegazza에 의해서 RTLinux 개념과 유사하게 시작되었다. 현재는 ADEOS (Adaptive Domain Environment for Operating System)를 이용하고 있다[9]. 지원 프로세서로는 ARM, m68k, i386계열, x86, PowerPC 등이 있다. 그러나 최근에는 프로젝트 진행이 활발히 이루어지지 않고 있고 기술 지원이 부족하여 사용에 어려움이 존재한다.

Xenomai 역시 RTAI와 같이 개발되었으나 2005년 분리되어 ADEOS를 이용하여 진행되었고 RTAI와는 다르게 다양한 RTOS 포팅과 유지관리 측면에 목표를 두고 있다. 오픈소스의 장점과 성능의 우수성으로 인하여 지능형 서비스 로봇의 실시간 운영체제로도 많이 활용되어 왔다[10-12]. 지원 프로세서로는 ARM, Blackfin, Nios II, i64, PowerPC, x86, i386 등이 있으며, RTAI에 비하여 지속적인 패치가 이뤄지고 있다[8].

본 논문에서는 공개된 실시간 임베디드 리눅스인 RTAI와 Xenomai의 특징을 커널의 아키텍처 입장에서 분석하고 실시간 시스템 설계에 있어서 가장 중요한 실시간 메커니즘의 시간적인 성능 분석을 목표로 한다. 이를 위하여 먼저 최신의 리눅스 커널을 이용하여 RTAI와 Xenomai 커널 구현 방법을 단계별로 소개하도록 한다. 그리고 구현된 RTAI와 Xenomai 환경에서 커널 프로그램을 이용하여 실시간 시스템 구현을 실시간 메커니즘인 API의 성능을 분석한다.

* 책임저자(Corresponding Author)
논문접수: 2011. 12. 9., 수정: 2012. 1. 9., 채택확정: 2012. 2. 8.
고재환, 최병욱: 서울과학기술대학교 전기정보시스템공학과
(go3167@gmail.com/bwchoi@seoultech.ac.kr)
※ 본 연구는 서울과학기술대학교 교내 학술연구 지원비로 수행되었음.

실시간 시스템 구현의 기본이 되는 실시간 타스크의 주기성을 분석하고, 실시간 메커니즘으로 타스크간 동기화와 통신에 가장 많이 사용되는 세마포어(semaphore), 실시간 FIFO (First-In First Out), 메일박스(mailbox) 및 메시지큐(message queue)의 시간적 성능을 분석한다.

실시간 메커니즘의 시간적 성능 분석은 실시간 시스템 설계에 있어서 타스크의 실시간성을 보장하기 위하여 RTOS 분야에서는 많은 연구가 진행되어왔다[13]. 그러나 RTAI나 Xenomai는 오픈 소스 프로젝트로 진행되면서 체계적인 성능 분석 보다는 시스템의 확장성과 범용성 측면에서 진행되어왔다. 성능 분석에 대한 연구로 시스템의 제어 성능과 인터럽트 처리에 대한 지연의 관점에서 진행된 연구가 발표되었다[14-16]. 그러나 실시간 시스템은 다중 타스크로 구현되며 실시간 메커니즘을 이용하여 타스크간 통신과 동기화 그리고 자원의 관리가 이루어진다[1]. 따라서 실시간 메커니즘의 시간적 성능에 대한 연구는 실시간성을 보장하기 위하여 매우 중요하다. 또한 최신 커널을 이용한 오픈 소스 프로젝트인 RTAI와 Xenomai에 대한 실시간 메커니즘의 시간적 성능 비교는 최신 리눅스 커널을 이용하여 실시간 시스템을 구현하고자 하는 개발자에게 다중 타스크에서 어떠한 메커니즘을 이용하는가를 결정하는 매우 유용한 연구 결과이다.

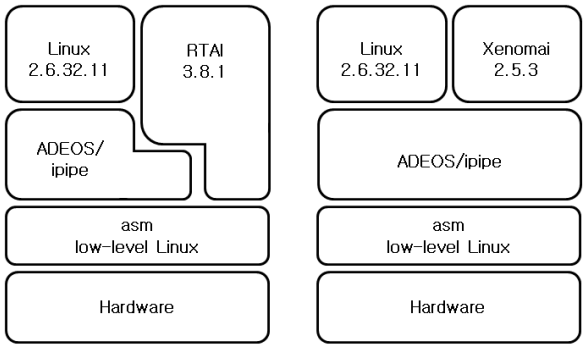
본 논문에서는 II 장에서는 실시간 임베디드 리눅스 커널 아키텍처와 실시간 API를 비교한다. 그리고 III 장에서 최신 리눅스 커널을 이용하여 실시간 임베디드 리눅스 구현 과정을 자세히 설명하며, IV 장에서는 II장에서 비교된 실시간 메커니즘의 실시간성을 커널 공간에서 측정하고 비교한 결과를 기술한다. 그리고 결론을 맺도록 한다.

II. RTAI와 Xenomai 아키텍처 및 API 비교 분석

1. 아키텍처 비교

RTAI에서 실시간 타스크를 위한 운영체제로는 리눅스가 사용된다. RTAI 최신 버전은 3.8.1로서 리눅스 커널 2.6.32를 지원하고 있다. 또한 개발 툴로서 gcc-4.4.3을 지원하고 있다. Xenomai는 여러 종류의 실시간 인터페이스(real-time interface)를 구축할 수 있는 추상적인 RTOS 코어를 기반으로 하는 실시간 개발 프레임워크(real-time development framework)이다. 최신 버전은 2.5.3이며 리눅스 커널 2.6.32를 지원하고 있다.

그림 1은 본 논문에서 구현한 아키텍처이다. 그림과 같이 일반적으로 RTAI와 Xenomai는 범용 리눅스 커널과 실시간 API를 동시에 사용한다는 점에서 개념적으로 공유된 내용을 이용하고 있으나 하드웨어로 부터 인터럽트를 처리하는 방법에 있어서 양간의 차이가 있다. 두 커널 모두 ADEOS를 이용한다[9]. 하드웨어 인터럽트는 ADEOS에서 처리된 후 논리적으로는 파이프 구조에 의하여 다른 컴포넌트 즉 Linux, RTAI 또는 Xenomai로 전달된다. 그러나 이 부분에서 RTAI와 Xenomai의 차이가 존재하는데 그림 1(a)와 같은 RTAI 구조에서는 모든 인터럽트를 ADEOS에서 처리하는 것이 아니라 RTAI가 가로채서 처리하는 것이 가능하다. 이와 같은 이유는 인터럽트가 실시간 타스크를 동작



(a) RTAI. (b) Xenomai.

그림 1. 실시간 임베디드 리눅스 아키텍처.
Fig. 1. Real-time Embedded Linux Architecture.

시키는 경우 ADEOS에 의한 인터럽트 처리에 따른 오버헤드를 제거할 수 있기 때문이다. 반면에 그림 1(b)와 같이 인터럽트 구조가 단순화된 Xenomai는 일관성 있는 처리를 할 수 있는 장점이 있다. 이러한 이유로 인터럽트 지연에 대한 시간적 성능의 경우 RTAI가 Xenomai에 비하여 우수한 결과를 나타내고 있다[15].

2. API 비교

RTAI는 API 인터페이스로 모듈이 존재한다. 각 모듈에 존재하는 IPC (Inter-Process Communication)를 이용하여 실시간 타스크를 구현하여 실시간 제어가 가능하다. 특히 LXRT 즉 사용자 레벨에서 실시간 프로그램이 가능한 특징을 가지고 있다.

- RTAI services functions.
- LXRT module.
- RTAI FIFO module
- Inter-process communications.
- Semaphores.
- Mailbox functions
- Message handling functions
- Remote procedure call functions
- Semaphore functions
- Unified RTAI real-time memory management.
- mini RTAI LXRT tasklets module

반면에 Xenomai에 있어서 Native API는 다른 스킨용 RTOS와 동일하게 사용이 가능하다. 표준 Xenomai에 제공되는 커널 모듈 xeno_native에서 지원되며 커널 공간이나 사용자 공간에서 모두 사용이 가능하다. 다음과 같은 기능이 제공되며 실시간 시스템 구현에 사용되어 진다[17].

- 타이밍 서비스: 워치독 타이머나 알람에 사용된다.
- 동기화 서비스: 세마포어, 뮤텍스, 이벤트 플래그 그룹이 제공되어 타스크간 동기화에 사용된다.
- 메시지 및 통신 서비스: 메시지 큐, 메모리 힙, 메시지 파이프가 존재하며 실시간 타스크 간이나 실시간 타스크

표 1. RTAI 및 Xenomai 메커니즘 비교표.
Table 1. Mechanisms of RTAI and Xenomai.

RTAI API package	*Xenomai native API package*
Semaphores	Semaphores
Condition variables	Condition variables
Mutexes	Mutexes
Bits	Event flag groups
FIFOs	Message pipes
Message queues(PIOSIX)	Message queues
Messages	Message passing support
Shared Memory	(shared) Heaps
Extended Mailboxes	Message queues

와 비 실시간 타스크 간의 정보 교환에 사용된다.

- 디바이스 I/O 처리: 실시간 I/O 처리를 위하여 RTDM (Real- Time Device Module)이 제공된다.
- 레지스트리 지원: 사용자 공간에서 실시간 메커니즘을 위한 객체 생성에 활용된다.

표 1은 RTAI와 Xenomai를 이용할 경우 프로그래밍 인터페이스 입장에서 가장 중요한 API를 비교하였다[17]. 약간의 차이는 있으나 일반적인 실시간 메커니즘을 모두 지원함을 알 수 있다.

III. 실시간 임베디드 리눅스 구현

RTAI와 Xenomai는 실시간 운영체제가 아닌 실시간 타스크를 위한 인터페이스이다. 따라서 RTAI나 Xenomai를 사용하기 위해서는 운영체제가 필요하다. 이러한 최신의 리눅스를 이용하여 실시간 임베디드 리눅스를 구현하는 것은 사용되는 도구(tool chain)의 호환성 유지와 단계별 처리 과정의 복잡함으로 일반 사용자가 구현하기가 매우 어렵다.

본 논문에서는 호환성 검토와 실험을 통하여 아래와 같은 개발 환경에서 최신 리눅스 커널을 활용하여 RTAI와 Xenomai를 구현하였다.

- Ubuntu 10.04 LTS
- Intel Core2 Duo CPU E8400 @ 3.00GHz
- kernel 2.6.32.11
- gcc-4.4.3

1. RTAI

RTAI를 패치하기 전에 커널 컴파일을 위하여 다음과 같은 패키지 설치가 필요하다.

- build-essential
- kernel-package
- libncurses5-dev

패키지 설치 후 RTAI에서 지원하는 커널 중 가장 최신 커널인 2.6.32.11과 RTAI-3.8.1을 다운 받아 설치하고 버전에 독립적인 도구의 호환성을 위하여 소프트 링크를 지정한다.

```
# cd /usr/src
# tar xvfj linux-2.6.32.11.tar.bz2
# ln -s linux-2.6.32.11 linux
# tar xvfj rtai-3.8.1.tar.bz2
# ln -s rtai-3.8.1 rtai
```

RTAI는 표준 리눅스 커널에서 실행되도록 설계되었기 때문에 현재 개발 환경으로 사용하는 우분투(ubuntu)의 커널 사용할 경우 예기치 않은 오류를 발생시킬 수 있으므로 Vanilla 커널을 사용한다.

```
# cd /usr/src/linux
# patch -p1 < /usr/src/rtai/base/arch/x86/patches/hal-linux-2.6.32.11-x86-2.6-03.patch
```

커널에 RTAI 패치를 적용 후 필요에 맞게 커널 설정을 한다. 일반적인 커널 설정이 매우 복잡함으로 아래와 같이 현재 개발 환경인 우분투의 설정을 가져와 필요한 부분을 수정하여야 쉽게 구현이 가능하다.

```
# cp /boot/config-2.6.32-28-generic /usr/src/linux
# make oldconfig
# make menuconfig
```

RTAI를 사용하기 위해서는 메뉴에서 설정이 필요한데 연구를 통하여 각 메뉴에서 필요한 설정을 나타내었다.

- General setup
  - Prompt for development and/or incomplete code/ drivers ⇒ "y"
- Enable loadable module support
  - Enable module support ⇒ "y"
  - Module unloading ⇒ "y"
  - Module versioning support ⇒ "n"
- Processor type and features
  - Preemption Model ⇒ Voluntary Kernel Preemption (Desktop)
  - Processor family
- Power Management options
  - CPU Frequency scaling ⇒ "n"

그 외의 설정은 개발 환경에 따라 설정해 주면된다. 그리고 다음과 같이 커널 컴파일과 모듈을 설치한다.

```
# make
# make modules_install
# make install
# update-grub
```

커널 컴파일이 끝나면 부트로더를 업데이트를 하고, 그 후 부팅메뉴가 항상 보이도록 다음과 같은 설정을 하여야 한다. 이후 부트로더를 업데이트 하고 리부팅하여 RTAI로 패치된 커널로 부팅하여야 한다.

```
# vi /etc/default/grub
GRUB_HIDDEN_TIMEOUT=0을 주석처리
# update-grub
# shutdown -r now
```

RTAI가 패치 된 커널로 부팅 후 그래픽 화면에서 실시간 메커니즘 모듈을 설치하는 과정을 진행한다.

```
# cd /usr/src/rtai
# make menuconfig
```

본 논문에서는 성능 분석에 이용할 FIFO, 세마포어, 메일박스를 모듈 형태로 설정하였다. 다른 필요한 설정을 완료 후 RTAI를 설치한다.

- Supported services
  - Fifo ⇒ "m"
  - Semaphores ⇒ "m"
  - Mailboxes ⇒ "m"

2. Xenomai

RTAI와 동일한 커널 사용을 위해 리눅스 커널 2.6.32를 지원하는 xenomai-2.5.3을 다운 받아 이용한다. 기본적인 구현 과정은 RTAI와 동일하다.

```
# cd /usr/src
# tar xvfj linux-2.6.32.11.tar.bz2
# ln -s linux-2.6.32.11 linux
# tar xvfj xenomai-2.5.3.tar.bz2
# ln -s xenomai-2.5.3 xenomai
```

Xenomai도 RTAI와 동일한 환경에서 성능 분석 실험을 진행하기 위해 Vanilla 커널을 사용한다. 이에 대한 패치 쉘 스크립트는 다음과 같다.

```
# cd /usr/src/xenomai
# scripts/prepare-kernel.sh --arch=x86 --adeos=/usr/src/xenomai/ksrc/arch/x86/patches/adeos-ipipe-2.6.32.11-x86-2.6-03.patch --linux=/usr/src/linux
```

패치 후 RTAI와 같이 현재 사용하고 있는 우분투 리눅스 설정을 이용하여 커널을 설정한다.

```
# cp /boot/config-2.6.32-28-generic /usr/src/linux
# make oldconfig
# make menuconfig
```

Xenomai는 x86에 패치 시 충돌을 일으킬 수 있다고 알려진 다음과 같은 설정을 disable 시켜야 한다.

- CONFIG_CPU_FREQ
- CONFIG_APM
- CONFIG_ACPI_PROCESSOR
- CONFIG_INTEL_IDLE
- CONFIG_INPUT_PCSPKR
- CONFIG_PCI_MSI
- CONFIG_CC_STACKPROTECTOR

위의 설정이 끝난 후 본 논문에서 사용하는 실시간 메커니즘을 사용하기 위하여 다음과 같이 설정을 한다.

- Real-time sub-system → interfaces → Native API
  - Message pipes ⇒ "y"
  - Counting semaphores ⇒ "y"
  - Message queues ⇒ "y"

컴파일 완료 후 부트로더를 업데이트 한다. 그 뒤 부팅 메뉴에서 패치 된 Xenomai 커널로 리부팅을 하여 다음 경로로 이동한 후 설치를 마무리 한다.

```
# cd /usr/src/xenomai
# ./configure
# make
# make install
```

IV. 성능 분석

실시간 시스템 구현은 표 1에 나타낸 실시간 메커니즘을 이용하며 실시간 타스크 간에 데이터 전달이나 동기화 등을 구현한다. 따라서 실시간 메커니즘에 대한 응답시간은 실시간 시스템의 성능을 예측하는데 매우 중요한 평가 기준이 된다. 이와 같은 실시간 메커니즘의 응답시간에 대한 연구는 상용 RTOS에서 많이 진행되었다[13]. 그러나 실시간 임베디드 리눅스에서는 인터럽트 지연시간에 대한 연구는 진행되었으나 실시간 메커니즘의 시간적 성능에 대하여서는 보고된 바가 없다[15,16]. 실시간 타스크의 최악의 수행 시간 분석과 시간의 측면에서 예측 가능한 시스템 성능을 보장하여야 하는 실시간 시스템에서는 시간적 응답 특성은 매우 중요한 결과이다.

본 연구를 위하여 커널 공간에서 프로그램을 작성하였다. 또한 실험의 용이성을 위하여 모듈 구조를 이용하여 구현하였다. 그리고 모든 성능 분석은 커널 메시지 함수를 이용하여 수행 후 확인하는 절차를 거쳤다. 따라서 데이터 처리에 따른 지연 시간을 배제하였다. 또한 스케줄링에 의한 지연을 방지하기 위하여 항상 하나의 모듈을 커널에 구동하여 실험하였다. 그러나 이와 같이 진행하여도 리눅스에 기본적으로 동작하는 프로세스가 존재하기 때문에 성능의 비교를 파악할 수 있도록 50번의 실험을 반복하였다. 이 이상을 진행하는 것은 비슷한 결과를 보였으며, 실제 최악의 실행 시간 분석은 실시간 시스템 연구 분야에서도 매우 어려운 분야이다. 따라서 본 연구에서는 지능형 서비스 로봇의 운영체제로서 두 개의 실시간 임베디드 리눅스의 성능 분석을 목표로 하여 진행하였으며 본 결과를 이용하여 실시간 시스템에서 다중 타스크를 구현할 경우 가장 적절한 실시간 메커니즘에 대한 기준을 제공할 수 있다.

1. 주기적 타스크의 주기성

커널 공간에서 그림 2와 같은 50 ms 주기를 갖는 함수의 주기성에 대한 성능분석을 위하여 총 50번 반복 수행하여 결과를 표 2에 나타내었다. 일반적인 RTOS의 경우 10 ms를 기본으로 하여 이에 정수배의 주기로 실시간 타스크를 구현하고 있다. 일반적인 타스크의 경우 50 ms가 많이 활용되기 때문에 이를 중심으로 성능 분석을 진행하였다.

표 2에 나타난 결과는 설정한 주기 50 ms에서 실제 타스크의 주기성의 오차에 대한 결과로 RTAI의 경우 오차의 평균이 15878 ns 만큼 발생하였고, Xenomai의 경우에는 8269 ns 만큼의 평균 오차가 발생하였다. 위의 결과로 보면 Xenomai가 RTAI보다 약 2배 가까이 정확한 성능을 보인다.

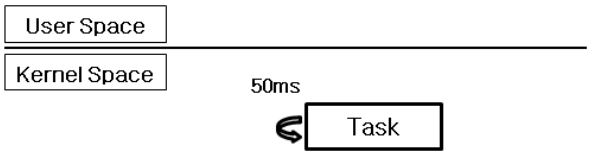


그림 2. 주기적 타스크 구조.
Fig. 2. Periodic task structure.

표 2. Periodic Task의 실험 결과.
Table 2. Experimental results of the Periodic Task.

	최상 [ns]	최악 [ns]	평균 [ns]
RTAI	64	38890	15878
Xenomai	21	42426	8269

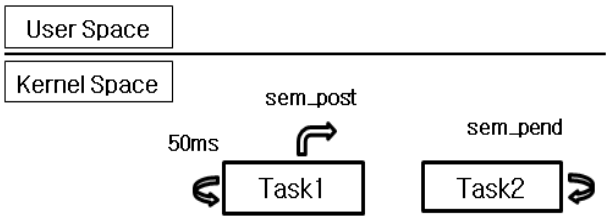


그림 3. 세마포어 분석을 위한 타스크 구조.
Fig. 3. Task structure for Semaphore evaluation.

표 3. Semaphore의 실험 결과.
Table 3. Experimental results of the Semaphore.

	최상 [ns]	최악 [ns]	평균 [ns]
RTAI	1300	2942	2034
Xenomai	1027	2244	1416

2. 세마포어(semaphore)

세마포어의 성능 분석을 위하여 그림 3과 같이 실시간 타스크를 구현하였다. 커널 공간에서 Task1과 Task2가 동작하며 Task1은 50 ms 주기적 타스크이며 세마포어를 양도한다. Task2는 세마포어를 획득하기 위하여 대기하고 있다가 Task1이 세마포어가 양도되면 다음 명령을 실행한다. 즉 Task2는 세마포어를 통하여 Task1과 동일하게 50 ms로 동기화 되고 있다. 성능 분석은 Task1이 세마포어를 양도한 시점부터 Task2가 세마포어를 획득한 시점까지 걸린 시간을 체크하였다. 이번 연구 역시 총 50번 반복 수행하였고 결과는 표 3과 같다.

세마포어의 경우에도 Xenomai가 RTAI에 비하여 비교적 더 빠른 타스크 전달 결과를 보여주고 있다.

3. FIFO

그림 4는 커널 공간의 50 ms의 주기적인 RT_Task에서 FIFO를 통하여 사용자 공간의 타스크로 데이터를 전달하고 다시 사용자 공간에서 커널 공간의 타스크로 데이터를 전달하는 구조이다.

FIFO는 읽기와 쓰기를 모두 지원하는 모드(O_RDWR)로 사용이 불가능하여 양방향 자료 전송을 위해 두 개의 FIFO를 사용하였다. 또한 성능 분석을 위해 RT_Task에서 데이터를 전달하고 다시 받는데 까지 걸린 시간을 측정하였다. Xenomai의 경우 FIFO가 표 1에 나타낸 바와 같이 메시지 파이프(message pipe)로 정의 되어 있다.

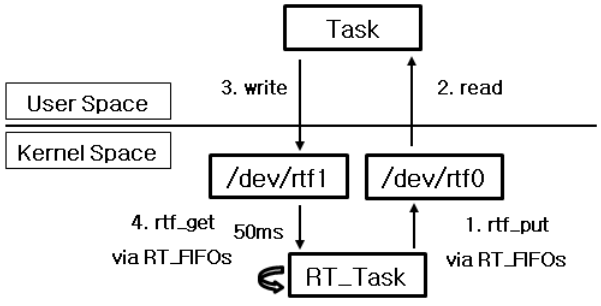


그림 4. FIFO 분석을 위한 타스크 구조.
Fig. 4. Task structure for FIFO evaluation.

표 4. FIFO의 실험 결과.
Table 4. Experimental results of the FIFO.

	최상 [ns]	최악 [ns]	평균 [ns]
RTAI	798	1259	973
Xenomai	1079	1615	1289

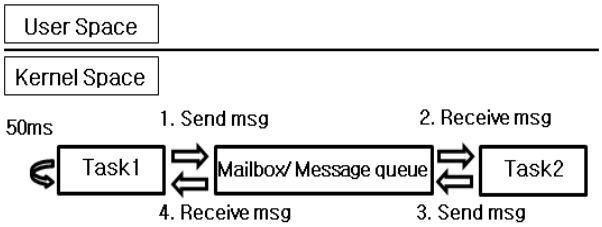


그림 5. Mailbox & Message queue 분석을 위한 타스크 구조.
Fig. 5. Task structure for Mailbox & Message queue evaluation.

표 5. 메일박스와 메시지큐의 실험 결과.
Table 5. Experimental results of the Mailbox and Message queue.

	최상 [ns]	최악 [ns]	평균 [ns]
RTAI Mailbox	5430	10747	7090
Xenomai Message queue	3189	5707	3737

표 4의 실험 결과를 보면 50번의 실험에 대하여 RTAI에서 제공하는 실시간 FIFO가 Xenomai에서 제공하는 메시지 파이프에 비하여 우수한 성능을 보인다. 따라서 커널 공간과 사용자 공간의 타스크 간의 데이터 전송에서는 RTAI가 우수한 성능을 보이고 있다.

4. 메일박스와 메시지큐(mailbox & message queue)

데이터 전송에 사용되는 메일박스와 메시지큐의 경우 표 1과 같이 RTAI에는 메일박스가 제공되나 Xenomai는 메시지큐만 존재한다. 따라서 본 논문에서는 성능 분석을 RTAI에서는 메일박스로 그리고 Xenomai에서는 메시지큐를 이용하여 그림 5와 같은 구조를 이용하여 데이터 전송에 사용되는 실시간 메커니즘의 성능을 분석하였다.

실험은 50 ms의 주기를 갖는 Task1이 메일박스나 메시지큐에 메시지를 보내면 데이터를 기다리던 Task2가 메일박스나 메시지큐에서 메시지를 읽어온다. 따라서 데이터 전송에 의하여 Task2는 50 ms의 동기화가 이루어지며 데이터 전송이 이루어진다. 그리고 Task2가 다시 메일박스나 메시지큐에 메시지를 전달하고 Task1이 데이터를 받아온다. 실험은 Task1이 메시지를 보낸 시점부터 Task2가 받은 메시지를 다시 Task1에 보내서 다시 받아올 때까지 걸린 시간을 측정하였다. 결과는 표 5와 같다. 그림에 나타낸 바와 같이 메일박스에 비하여 메시지큐가 데이터 전송에 우수한 시간적 반응 특성을 나타내었다. 일반적으로 메일박스는 1:1의 타스크에 전역 변수를 전달하는 메커니즘으로 사용되며 메시지큐는 N:1의 타스크 구조의 데이터 전달이 가능한 메커니즘으로 직접적인 비교는 어려우나 시간적 성능의 분석은 실시간 시스템 구현에 매우 유용한 정보이다.

V. 결론

실시간 임베디드 리눅스의 성능은 많은 연구에서 발표된 바와 같이 상용의 RTOS에 비하여 뒤지지 않는 결과를 보이고 있다. 그러나 각각의 실시간 메커니즘의 경우 실제적인 API 수행 시간에 대한 연구가 진행되지 아니하였다. 일반적으로 상용 RTOS는 모든 API에 대하여 최악의 성능 시간을 제공함으로써 사용자가 시스템을 설계할 경우 최악 조건에서 실시간 실행 시간을 예측할 수 있는 자료를 제공한다. 그러나 실시간 임베디드 리눅스의 경우 오픈 소스로 개발됨에 따라 계속적인 성능 개선이 이루어지고는 있지만 이와 같이 사용자에게 필요한 연구가 미비한 실정이다.

본 논문에서는 실시간 임베디드 리눅스인 RTAI와 Xenomai의 활용을 위하여 최신 커널을 이용한 실시간 리눅스 확장 과정을 소개하였다. 또한 RTAI와 Xenomai에 대하여 타스크 관리의 기본적인 성능인 주기성에 대한 성능 분석 그리고 타스크 동기화를 위한 세마포어의 실시간 성능 분석과 사용자 공간과 커널 공간의 데이터 전송을 위한 FIFO에 대한 성능 분석 그리고 마지막으로 실시간 타스크 간의 정보 교환에 사용된 메일박스, 메시지큐의 실시간 성능 분석을 수행하였다.

실시간 메커니즘의 시간적 성능 분석은 실시간 시스템 설계에 있어서 타스크의 실시간성을 보장하기 위하여 매우 중요하다. 따라서 본 연구의 결과는 실시간 임베디드 리눅스를 이용하여 지능형 서비스 로봇 등의 실시간 시스템 설계를 위한 가장 기초적인 자료를 제공하고 있어서 매우 유용한 정보이며, 향후 사용자 공간에서의 실시간성 분석과 함께 본 연구에서 확인된 성능의 차이에 대한 원인 분석을 통하여 최적의 실시간 메커니즘을 구현한다면 최적의 실시간 임베디드 시스템을 구현할 수 있을 것으로 기대된다.

참고문헌

[1] B. W. Choi “A Review and Outlook of Robotic Software Framework,” *Journal of Korean Robotic Society*, vol5, no.2, pp. 169-176, 2010.

[2] W. S. Liu, *Real-Time Systems*, Prentice Hall, 2000.

[3] Tim Bird, *Comparing two approaches to real-time Linux*, www.linuxdevices.com, 2002.

[4] Kevin Dankwardt, *Comparing real-time Linux alternatives*, www.linuxdevices.com, 2000.

[5] D. Abbout, *Linux for Embedded and Real-time Applications*, Elsevier, 2006.

[6] N. Vun, H. F. Hor, and J. W. Chao, “Real-time Enhancements for Embedded Linux,” *4th IEEE Int. Conf. on Parallel and Distributed Systems*, pp. 737-740, 2008.

[7] RTAI - the Real-time Application Interface for Linux from DIAMP, *http://www.rtai.org.*

[8] The Xenomai Project, *http://www.xenomai.org.*

[9] The ADEOS Project, *http://home.gna.org/adeos.*

[10] E. C. Shin and B. W. Choi, “Implementation of a mobile robot control platform using real-time embedded linux,” *Journal of Control, Automation, and Systems (in Korean)*, vol. 12, no. 2, pp. 194-200, Feb. 2006.

[11] B. W. Choi, D. G. Shin, J. H. Park, S. Y. Yi, and S. Gerald, “Real-time control architecture using Xenomai for intelligent service robot in USN environments,” *Journal of Intelligent Service Robotics*, vol. 2, pp. 139-151, 2009.

[12] S. M. Hong, Y. H. Oh, B. J. You, and S. R. Oh, “A walking pattern generation method of humanoid robot MAHRU-R,” *Journal of Intelligent Service Robotics*, vol. 2, pp. 161- 171, 2009.

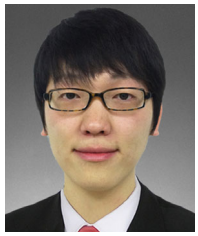
[13] *Comparison between VxWorks, QNX and PSosSystem.* Real Time Magazine,

[14] J. H. Park, S. Y. Yi, and B. W. Choi, “Implementation of dual-kernel based control system and evaluationof real-time control performance for intelligent robots,” *Journal Institute of Control, Robotics and Systems (in Korean)*, vol. 14, no. 11, pp. 1117-1123, Nov. 2008.

[15] A. Barbalace, A. Lunchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, “Performance Comparison of VxWorks, Linux, RTAI and XENOMAI in a Hard Real-time Application,” *Proc. of Real-Time Conference 2007 15th IEEE-NPSS*, pp. 1-5, May 2007.

[16] M. Franke, *A Quantitative Comparison of Realtime Linux Solutions*, Chemnitz University of Technology, 2007.

[17] Xenomai homepage, http://www.xenomai.org/documentation/xenomai-2.3/pdf/Native-API-Tour-rev-C.pdf.



고 재 환

2011년 서울과학기술대학교 전기공학과 졸업. 2011년도~현재 서울과학기술대학교 전기공학과 석사과정. 관심분야는 임베디드 리눅스, 실시간 시스템 설계.

최 병 욱

1988년 1992년 한국과학기술원 전기및 전자공학과 석사 및 박사졸업. 1988년~2000년 LG산전 중앙연구소 책임연구원. 2000년~2005년 선문대학교 제어계측공학과 부교수. 2003년~2005년 임베디드웹 대표이사. 2007년~2008년 Nanyang Technological University, Senior Fellow. 2005년~현재 서울과학기술대학교 전기정보시스템공학과 교수. 관심분야는 실시간 시스템 설계, 임베디드 시스템, 임베디드 리눅스, 지능형 로봇 소프트웨어.