Journal of Korea Robotics Society (2012) 7(4):292-298
http://dx.doi.org/10.7746/jkros.2012.7.4.292 ISSN: 1975-6291 / eISSN: 2287-3961

실시간 임베디드 리눅스에서 다양한 주기적 타스크의 실시간 메커니즘 성능 분석

On Benchmarking of Real-time Mechanisms in Various Periodic Tasks for Real-time Embedded Linux

고 재 환[1], 최 병 욱†

Koh Jae-Hwan[1], Choi Byoung-Wook†

Abstract It is a real-time system that the system correctness depends not only on the correctness of the logical result of the computation but also on the result delivery time. Real-time Operating System (RTOS) is a software that manages the time of a microprocessor to ensure that the most important code runs first so that it is a good building block to design the real-time system. The real-time performance is achieved by using real-time mechanisms through data communication and synchronization of inter-task communication (ITC) between tasks. Therefore, test on the response time of real-time mechanisms is a good measure to predict the performance of real-time systems. This paper aims to analysis the response characteristics of real-time mechanisms in kernel space for real-time embedded Linux: RTAI and Xenomai. The performance evaluations of real-time mechanism depending on the changes of task periods are conducted. Test metrics are jitter of periodic tasks and response time of real-time mechanisms including semaphore, real-time FIFO, Mailbox and Message queue. The periodicity of tasks is relatively consistent for Xenomai but RTAI reveals smaller jitter as an average result. As for real-time mechanisms, semaphore and message transfer mechanism of Xenomai has a superior response to estimate deterministic real-time task execution. But real-time FIFO in RTAI shows faster response. The results are promising to estimate deterministic real-time task execution in implementing real-time systems using real-time embedded Linux.

Keywords: Real-time embedded linux, IPC, Xenomai, RTAI, Real-time mechanism

1. 서 론

실시간 시스템은 시스템의 정확성과 함께 외부 입력에 대하여 원하는 시간적 제한 내에 반응을 위한 출력을 만들어내야 하는 시스템이다. 즉 예측되지 않는 외부 자극에 대하여 시간적으로 예측 가능한 출력을 만들어 내야 한다. 임베디드 시스템은 마이크로프로세서를 이용하여 특정한 목적의 하드웨어를 제어하는 시스템으로 다양한 분야에서 활용되고 있다. 실시간 시스템은 시간적 관점에서의 정의이고, 임베디드 시스템은 구현의 관점에서 정의이다. 최근에 와서 대부분의 실시간 시스템은 임베디드 시스템으로 구현되고 있으며, 이러한 이유로 임베디드 시스템의 운영체제로서 시간적 제약을 만족하여야 하는 시스템을 구현하기 위하여 실시간 운영체제인 RTOS(Real-Time Operating System)가 이용되고 있다.

일반적으로 임베디드 시스템에 사용되는 범용 운영체제는 실시간 시스템이 요구하는 성능을 보장하지 못하여 제어에 어려움을 겪게 된다. 따라서 실시간 시스템이 요구하는 비동기적인 반응에 대하여 미리 정해진 시간에 예측 가능한 방법으로 신뢰성 있는 출력을 보장하기 위해 RTOS를 사용한다. 실시간 운영체제는 임

Received : Aug. 10. 2012; Reviewed : Oct. 8. 2012; Accepted : Nov. 8. 2012
※ This study was financially supported by Seoul National University of Science and Technology
† Corresponding author: Electrical and Information Engineering, Seoul National Univ. of Science and Technology, Gongneung-Dong, Nowon-Gu, Seoul, Korea (bwchoi@seoultech.ac.kr)
1 Electrical Engineering, Seoul National Univ. of Science and Technology (go3167@gmail.com)

베디드 운영체제 중에서 시간을 제어할 수 있는 기능이 포함된 운영체제로 상용 RTOS와 오픈 소스로 진행중인 실시간 임베디드 리눅스로 구분 지을 수 있다[1-4]. 상용 RTOS는 검증된 성능, 체계화된 개발 환경 그리고 체계적인 기술 지원 등의 장점이 있지만 특정 운영체제로의 기술 종속이 될 수 있으며 고가의 초기 개발 비용 및 개발의 유연성이 떨어진다는 단점을 갖는다[5]. 이에 비하여 오픈 소스로 개발된 운영체제는 다양한 하드웨어 지원과 함께 성능도 우수하여 많은 시스템에 적용되고 있다[6-9]. 이러한 RTOS를 이용한 실시간 시스템은 다중 타스크로 구현되며 실시간 메커니즘을 이용하여 타스크 간 통신과 동기화 그리고 자원관리가 이루어진다[10]. 따라서 실시간 시스템의 성능은 실시간 메커니즘의 시간적 성능으로 나타낼 수 있기 때문에 시스템의 성능을 보장하기 위하여 실시간 메커니즘의 시간적 성능 분석은 매우 중요한 연구 분야이다. 상용 RTOS는 실시간 메커니즘의 시간적 성능 분석이 체계적으로 이루어지고 있지만 오픈 소스인 실시간 임베디드 리눅스는 이와 같은 성능 분석이 미비한 실정이다.

현재 실시간 임베디드 리눅스의 성능 분석에 대한 연구로 시스템의 제어 성능과 인터럽트 처리에 대한 지연의 관점에서 진행된 연구가 발표되었고, 또한 선행 연구로써 오픈 소스 프로젝트인 RTAI와 Xenomai를 이용하여 동일 주기 타스크에서의 실시간 메커니즘에 대하여 시간적 성능 분석을 수행하였다[11-13]. 본 논문에서는 선행 연구결과를 바탕으로 타스크의 주기가 변화할 경우에 실시간 메커니즘의 시간적 성능을 분석한다. 대부분의 실시간 시스템에서 사용되는 실시간 타스크가 주기적인 특성을 갖는다는 점에서 다양한 주기에서의 실시간 메커니즘의 성능 변화에 대한 연구가 필요하다. 본 연구는 실시간 시스템을 구현하기 위해 다양한 주기의 실시간 타스크를 구현하여야 한다는 점에서 처음 시도 되는 매우 유용한 결과이다. 추후, 결과에 따른 오픈 소스 분석을 수행한다면 최적의 실시간 메커니즘을 가지는 실시간 임베디드 리눅스 구현이 가능할 것으로 기대된다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 본 논문에서 구현한 실시간 임베디드 리눅스의 실험 환경 및 방법을 간략하게 서술하고, 3장에서는 실시간 시스템의 기본이 되는 실시간 타스크의 주기성을 분석하며, 4장에서는 실시간 메커니즘의 시간적 성능이 타스크의 주기에 따라 어떻게 변화되는지를 살펴본다. 그리고 5장에서 결론을 맺는다.

2. 실시간 임베디드 리눅스 구현

RTAI와 Xenomai는 실시간 운영체제가 아닌 실시간 타스크를 위한 인터페이스다. 따라서 RTAI나 Xenomai를 사용하기 위해서는 운영체제가 필요하다. 본 논문에서는 호환성 검토와 실험을 통하여 아래와 같은 개발 환경에서 최신 리눅스 커널을 활용하여 RTAI와 Xenomai를 구현하였다. 자세한 구현 과정은 선행 연구에서 다뤘다[13].

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

본 실험은 커널 공간에서 프로그램을 작성하였고, 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. 실시간 메커니즘들의 수행시간 측정은 각각 RTOS들이 제공해주는 API를 이용하여 시간을 측정하였다.

3. 주기적 타스크의 주기성 연구

본 논문에서는 대표적인 실시간 임베디드 리눅스인 RTAI와 Xenomai를 IPC(Inter Process Communication) 관점에서 시간적 성능 분석 한다. 성능 분석은 실시간 시스템 구현의 기본이 되는 실시간 타스크의 주기성을 분석하고, 실시간 메커니즘으로 타스크 간 동기화와 통신에 가장 많이 사용되는 세마포어(Semaphore), 실시간 FIFO(First-In First Out), 메일박스(Mailbox) 및 메시지큐(Message Queue)의 시간적 성능을 분석한다. 본 논문에서 수행한 실시간 메커니즘의 시간적 성능 분석

은 상용 RTOS 및 최근에 개발된 uC/OS-III의 성능 분석 방식과 동일한 방식으로 수행되었다[14].

실시간 시스템은 다양한 주기로 구현되며 일반적인 RTOS의 경우 10[ms]를 기본으로 하여 이에 정수 배의 주기로 실시간 타스크를 구현한다. 따라서 본 논문에서는 타스크의 주기에 따른 실시간 메커니즘의 성능 차이를 보기 위해 타스크의 주기를 10[ms], 30[ms], 50[ms]로 변경하며 실시간 메커니즘의 응답 시간을 측정하였다. 타스크의 주기가 변경되어도 실험 구조는 동일하다. Fig. 1은 전체 실험 구조를 간략화 하여 그림으로 나타낸 것이다.

Fig. 2는 Periodic Task의 실험 구조를 의사코드로 표현하였다. 타스크가 처음 실행되면 원하는 주기를 설정하고 타스크가 주기적으로 실행되게 만든다. 그 다음 각각의 실시간 임베디드 리눅스가 제공하는 API를 이용하여 타스크가 처음 설정한 주기에 정확히 실행되는지 보기 위해 시간을 측정한다. 각각의 주기에 대한 실험 결과는 Fig. 3~5와 같다.

Fig. 3~5에 나타난 결과는 각각의 주기에서 실제 타스크의 주기성의 오차에 대한 결과이다. 결과는 [ns]로 나타내었다. 50[ms]주기에서 Xenomai는 평균적으로 ±8269[ns]의 오차가 났고, RTAI의 경우 ±15878[ns]의

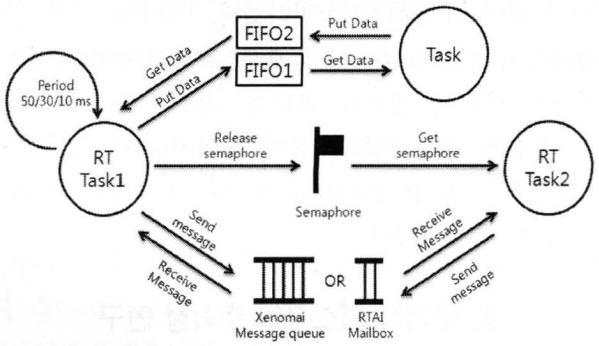

Fig. 1. Task diagram of testing real-time performance

```
void Task1()
{
    set period and make periodic;
    ...
    while(1)
    {
      start time measurement;
      wait period;
      stop time measurement;
    }
}
```

Fig. 2. Pseudocode for testing periodic task

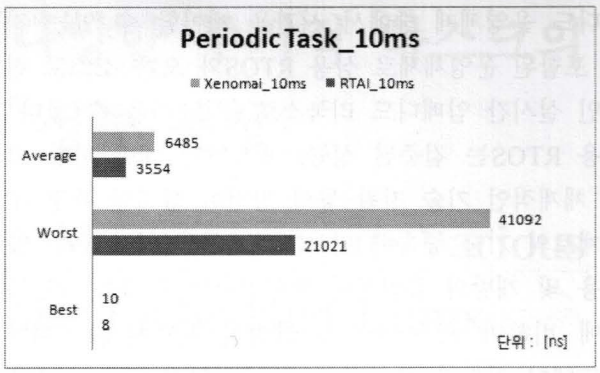

Fig. 3. Experimental result of task of period 10ms

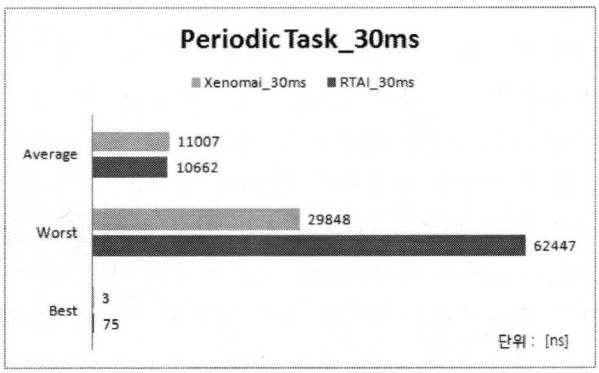

Fig. 4. Experimental result of task of period 30ms

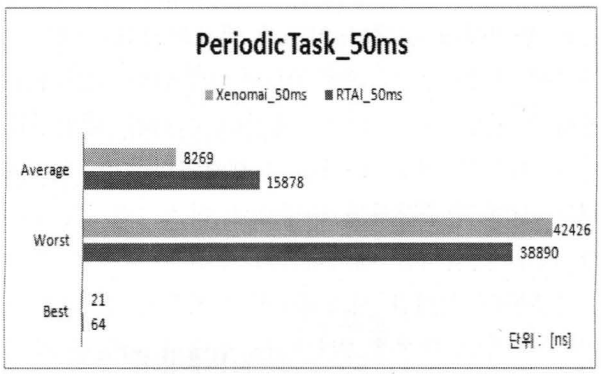

Fig. 5. Experimental result of task of period 50ms

오차가 발생하였다. 30[ms] 주기에서는 RTAI와 Xenomai의 오차의 평균이 비슷하게 나타났고 10[ms] 주기에서는 50[ms]와는 달리 RTAI가 Xenomai에 비해 오차의 평균값이 2배 가까이 줄어드는 모습을 볼 수 있다. 즉 타스크의 주기가 짧을수록 RTAI가 Xenomai에 비해 정확한 시간에 타스크가 실행되었고 주기가 길수록 Xenomai가 RTAI에 비해 정확한 시간에 타스크가 실행되었다. 또한 RTAI의 경우 타스크의 주기가 감소 할 수록 오차가 현저하게 줄어드는 모습을 볼 수 있다.

4. 주기성에 따른 실시간 메커니즘 성능 분석

4.1 세마포어(Semaphore)

Fig. 6은 세마포어의 시간적 성능 분석을 위한 의사코드이다. Task1이 세마포어를 양도한 시점부터 Task2가 세마포어를 획득한 시점까지 걸린 시간을 측정하였다. 즉, Task1이 세마포어를 양도하고 Task2로 Context Switch가 일어나고 Task2가 세마포어를 획득하기까지 걸린 시간을 측정하여 결과를 Fig. 7~9에 나타내었다.

실험 결과를 살펴보면 Xenomai가 세마포어를 양도하고 다른 타스크가 이를 획득하는데 걸리는 시간이

```
void Task1()
{
  set period and make
  periodic;
  ...
 while(1)
 {
  release semaphore;
  start time measurement;
 }
}
```

```
void Task2()
{
  ...
  while(1)
  {
   get semaphore;
   stop time measurement;
  }
}
```

Fig. 6. Pseudocode for testing semaphore

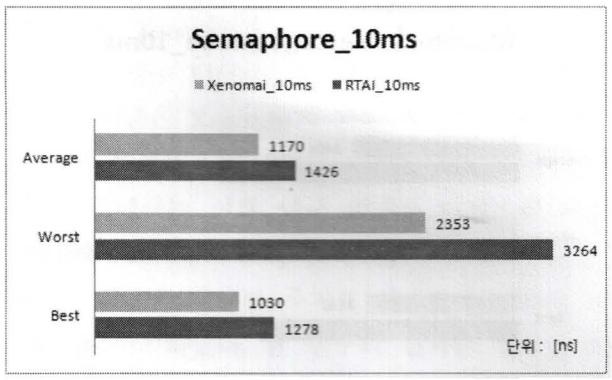

Fig. 7. Result of semaphore of period 10ms

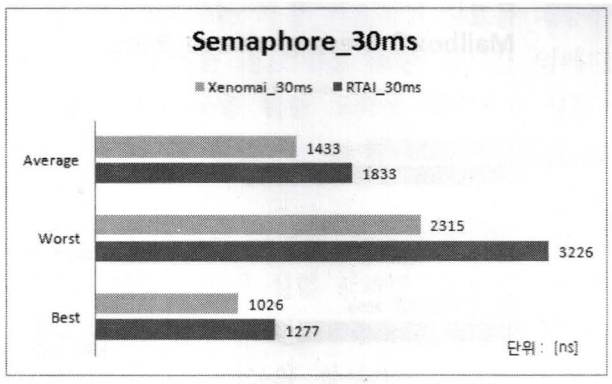

Fig. 8. Result of semaphore of period 30ms

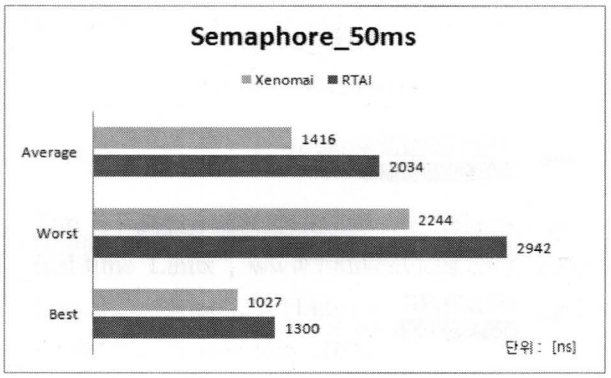

Fig. 9. Result of semaphore of period 50ms

모든 주기에서 RTAI보다 짧게 소요되는 모습을 볼 수 있다. 또한 RTAI의 경우 주기가 짧을수록 Fig. 6의 과정을 수행하는데 걸리는 시간이 감소하였다. 실시간 시스템을 구현 할 경우 Xenomai의 Semaphore를 이용할 경우 모든 주기에서 RTAI보다 우수한 시간적 응답 특성을 얻을 수 있다.

4.2 실시간 FIFO

Fig. 10은 커널공간의 Task1이 실시간 FIFO를 통하여 사용자 공간의 타스크로 데이터를 전달하고 다시 사용자 공간에서 커널 공간의 타스크로 데이터를 전달하는 구조를 의사코드로 나타내었다. 실험은 위의 일련의 과정을 수행하는데 걸린 시간을 측정하였다. FIFO는 읽기와 쓰기를 모두 지원하는 모드로 사용이 불가능하여 양방향 자료 전송을 위해 두 개의 FIFO를 사용하였다. Xenomai의 경우 FIFO가 메시지 파이프(Message Pipe)로 정의 되어있다. 실험 결과는 Fig. 11~13과 같다.

실험 결과를 보면 타스크의 주기가 짧아질수록 RTAI와 Xenomai 모두 수행시간이 감소하였다. 또한 모든 주기에서 RTAI가 제공하는 실시간 FIFO가

Kernel Space

```
void Task1()
{
  set period and make
  periodic;
  ...
 while(1)
 {
  put data to FIFO1;
  start time measurement;
  get data from FIFO2;
  stop time measurement;
 }
}
```

User Space

```
void main()
{
  ...
 while(1)
 {
  get data from FIFO1;
  put data to FIFO2;
 }
}
```

Fig. 10. Pseudocode for testing real-time FIFO

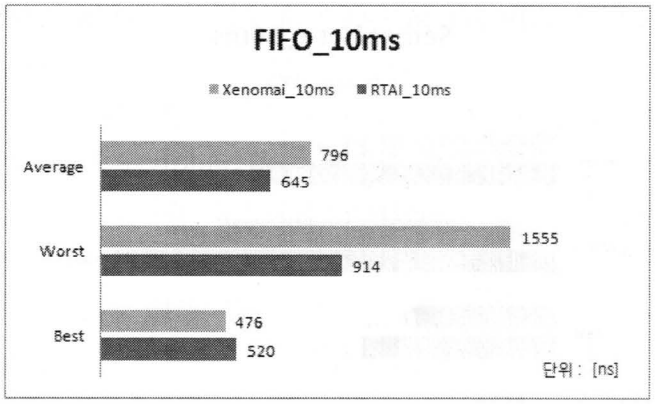

Fig. 11. Result of FIFO of period 10ms

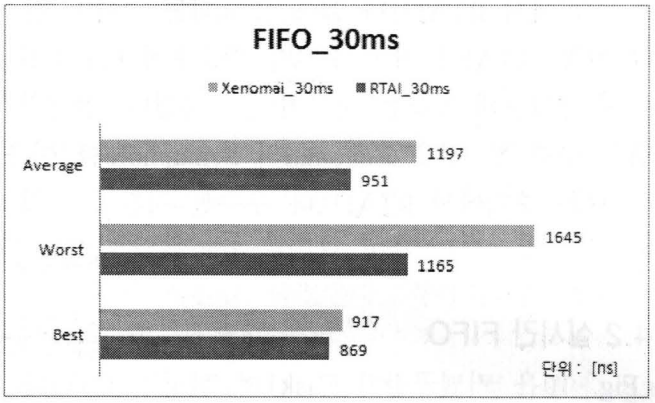

Fig. 12. Result of FIFO of period 30ms

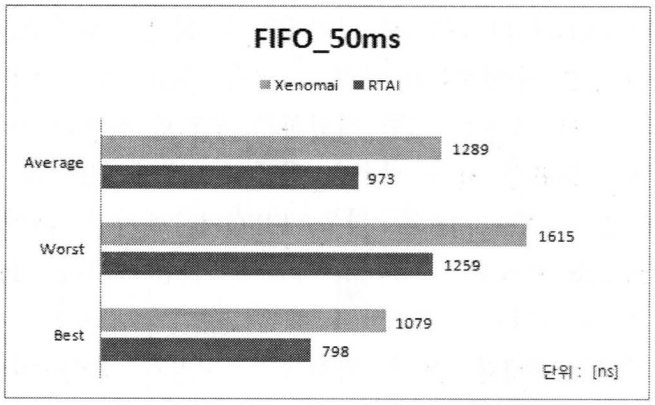

Fig. 13. Result of FIFO of period 50ms

Xenomai에서 제공하는 메시지 파이프에 비해 모든 주기에서 짧은 응답 시간 특성을 나타내었고, 더 낮은 최악의 응답 시간을 보장하고 있다.

4.3 메일박스와 메시지큐

데이터 전송에 사용되는 메일박스와 메시지큐의 경우 RTAI는 메일박스, Xenomai는 메시지큐만 API를 제공하고 있다. 따라서 본 논문에서는 성능 분석을 RTAI에서는 메일박스로 그리고 Xenomai에서는 메시지큐를 이용하여 Fig. 14와 같은 구조의 데이터 전송

```
void Task1()
{
  set period and make periodic;
  ...
 while(1)
 {
  send message to Task2;
  start time measurement;
  wait for receive message from Task2 ;
  stop time measurement;
 }
}
```

```
void Task2()
{
  ...
 while(1)
 {
  wait for receive message from Task1;
  send message to Task1;
 }
}
```

Fig. 14. Pseudocode for testing Mailbox and Message queue

에 사용되는 실시간 메커니즘의 성능을 분석하였다.

실험은 Fig. 14에 나타낸 의사코드와 같은 구조로 Task1이 Task2에 메시지를 전달하고, 다시 Task2로부터 메시지를 받는 데까지 소요된 시간을 측정하였다. 또한 주기를 변경하며 주기에 따른 시간적 성능 차이를 분석하였다. 실험 결과는 Fig. 15~17과 같다.

실험결과는 Fig. 15~17에서 볼 수 있듯이 RTAI에서 제공하는 실시간 메커니즘인 메일박스에 비해 Xenomai의 메시지큐가 데이터 전송에 우수한 시간적 반응 특성을 나타내었고, 타스크의 주기에 따른 수행

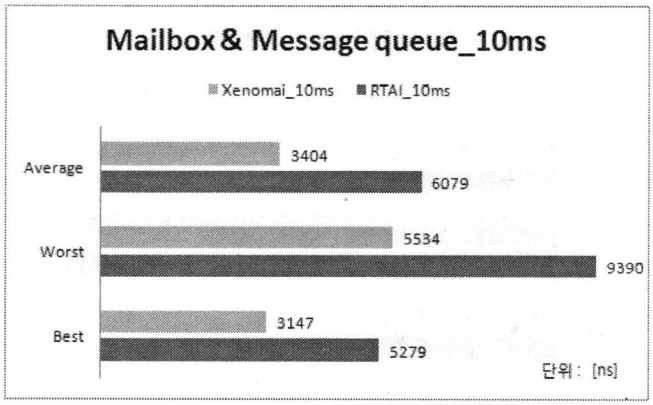

Fig. 15. Result of Mailbox and Message queue of period 10ms

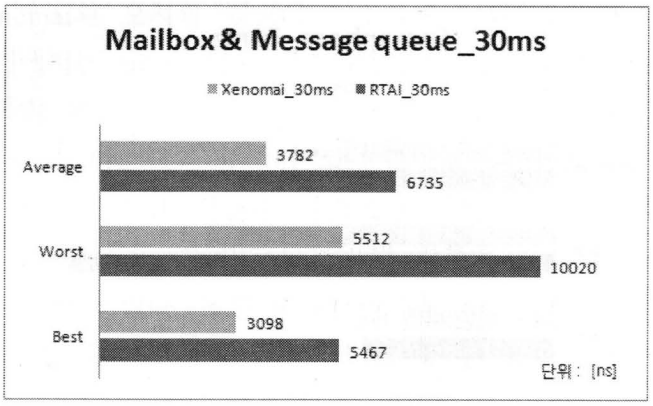

Fig. 16. Result of Mailbox and Message queue of period 30ms

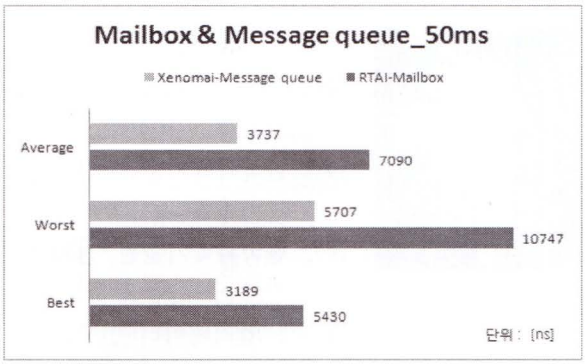

Fig. 17. Result of Mailbox and Message queue of period 50ms

시간의 차이는 거의 나타나지 않았다. 다만 RTAI의 메일박스의 경우 타스크의 주기가 감소할수록 Fig. 14와 같은 과정을 수행하는 시간이 감소하였다. 일반적으로 메일박스의 경우 1:1 타스크 구조 간 데이터 전달에 사용되는 메커니즘이고, 메시지큐는 N:1의 타스크 구조에서 사용되는 메커니즘이므로 직접적인 비교는 어려우나 이러한 실시간 메커니즘의 시간적 성능 분석은 실시간 시스템 구현에 매우 유용한 정보이다.

5. 결 론

실험결과를 요약하면 타스크의 주기성은 주기에 따라 RTAI와 Xenomai가 성능차이를 보였으며, 실시간 메커니즘의 경우 FIFO는 RTAI가 세마포어 및 메시지 전달 메커니즘은 Xenomai가 모든 주기에서 우수한 시간적 응답 특성을 나타내었다.

본 논문에서는 선행 연구 결과를 바탕으로 타스크의 주기에 따라 데이터 전송 및 동기화에 사용되는 실시간 메커니즘을 다양한 주기적 타스크에 대하여 시간적 성능을 분석하였다. 본 연구의 결과는 실시간 시스템의 타스크가 주기적 타스크로 구현되며 그 안에서 실시간 메커니즘을 사용한다는 관점에서 실시간 시스템의 성능을 예측하는 가장 기초적인 자료를 제공하고 있어서 매우 유용한 정보이다. 또한 실시간 임베디드 리눅스의 경우 사용을 위한 노력에 집중되고 성능 분석에 대한 연구가 없었다는 점에서 매우 귀중한 결과이다.

향후 오픈 소스로 구현된 실시간 시스템의 메커니즘 분석을 통하여 앞선 실험 결과에 대한 원인을 정확히 파악하고자 한다. 또한 최적의 실시간 메커니즘을 구현한다면 우수한 성능의 실시간 임베디드 리눅스를 구현할 수 있을 것으로 기대된다.

참고문헌

[1] Tim Bird, “Comparing two approaches to real-time Linux”, www.linuxdevices.com, 2002

[2] Ismael Ripoll,“RTLinux versus RTAI”, www.linuxdevices.com, 2002

[3] Kevin Dankwardt, “Comparing real-time Linux alternatives”, www.linuxdevices.com, 2000

[4] W. S. Liu, Real-Time System, Prentice Hall, 2000

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

[6] M.D. Marieska, A.I. Kistijantoro, and M. Subair, “Analysis and Benchmarking Performance of Real Time Patch Linux and Xenomai in Serving a Real Time Application,” Proc. of International Conf. on Electrical Engineering and Informatics, pp.1-6, 2011

[7] P. Kadionik, B. Le Gal, H. Levi, Ben Atitallah, “A. Performances analysis and evaluation of Xenomai with a H.264/AVC decoder,” Proc. of International Conf. on Microeletronics, pp.1-4, 2011

[8] G. Zhang, L. Chen, and A. Yao, “Study and Comparison of the RTHAL-based and ADEOS-based RTAI Real-time Solutions for Linux,” Proc. of International Multi-Symposium on Computer and Computational Sciences, pp.771-775, 2006

[9] M. Liu, D. Liu, Y. Wang, M. Wang, and Z. Shao, “On Improving Real-Time Interrupt Latencies of Hybrid Operating Systems with Two-Level Hardware Interrupts,” IEEE Trans. on Computers, Vol.60, No.7, pp.978-991, 2001

[10] Choi, B.W, “A Review and Outlook of Robotics Software Framework”, J. of Korean Robotic Society, Vol5, No2, pp169-176, 2010

[11] 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.

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

[13] Koh, J.H, Choi, B.W, "Performance Evaluation of Real-time Mechanisms for Real-time Embedded Linux", J. of Institute of Control, Robotics and Systems, Vol18, No.4, pp. 337-342, 2012.

[14] uC/OS-III, http://micrium.com

고 재 환

2011 서울과학기술대학교 전기공학과 (공학사)

2001~현재 서울과학기술대학교 전기공학과 석사과정

관심분야 : 실시간 운영체제, 임베디드 리눅스, 지능형 서비스 로봇.

최 병 욱

1986 한국항공대학교 항공전자공학과 (공학사)

1988 한국과학기술원 전기및 전자공학과 (공학석사)

1992 한국과학기술원 전기및 전자 (공학박사)

1988~2000 LG산전주식회사 엘리베이터 연구실장 및 임베디드 시스템 연구팀장

2000~2005 선문대학교 제어계측공학과 부교수

2007~2008 Nanyang Technological University, Senior Fellow

2005~현재 서울과학기술대학교 전기정보공학과 교수

관심분야 : 임베디드 시스템, 임베디드 리눅스, 실시간 운영체제, 지능형 로봇 응용, 소프트웨어 플랫폼