Journal of The Korea Society of Computer and Information
Vol. 22 No. 6, pp. 1-8, June 2017

www.ksci.re.kr
https://doi.org/10.9708/jksci.2017.22.06.001

Error Recovery Technique for Improving Reliability of Embedded Systems

Sunghoon Son*

Abstract

In this paper, we propose a fault tolerance technique which enables embedded systems to run without interruption while its operating system and tasks fail. In order to improve reliability, the proposed scheme makes an embedded system run as a virtual machine on virtual machine monitor. It also prepares a contingency virtual machine at which periodical backups of the embedded system are saved. When an error occurs in the main virtual machine, the corresponding standby virtual machine takes a role of the main virtual machine and continues its operation. Especially such backups and switches of virtual machines are performed with minor performance degradation by manipulating page table entries in virtual machine monitor. By conducting performance evaluation studies, we show that the proposed scheme makes embedded system robust against errors while it does not degrade the performance of the system significantly.

▸Keyword: Embedded System, Reliability, Error Recovery, Fault Tolerance

I. Introduction

임베디드 시스템은 여러 산업 분야에서 다양하게 사용된다. 흔히 임베디드 시스템이 사용되는 환경은 고도의 신뢰성과 가용성을 요구하는 경우가 많다. 따라서 임베디드 시스템이 오랜 시간 오류 없이 동작하도록 하는 것은 중요한 이슈 중의 하나이며, 이는 임베디드 시스템에도 고장 감내 (fault tolerance) 기법의 적용이 필요한 이유이다.

지금까지의 고장 감내 기법들은 전통적으로 금융 분야나 항공 관련 분야 등의 대형 서버에 적용되어 왔으며, 대부분 컴퓨터 시스템을 구성하는 하드웨어나 소프트웨어를 중복 설치하는 이중화를 통해 고장 발생에 대처하고 있다. 최근에는 임베디드 시스템을 위한 고장 감내 기술에 대한 연구도 다양하게 진행되고 있으나, 임베디드 시스템의 경우 서버와 같은 범용 컴퓨터 시스템과는 달리 자원의 제약이 많기 때문에 기존 범용 시스템에서 사용되었던 중복에 의한 고장 감내 기술을 그대로 적용하기는 어렵다.

최근에는 임베디드 시스템을 위한 가상화(virtualization) 기법들도 많이 연구되고 있다. 컴퓨터 시스템을 가상화하면 서버 병합(consolidation), 자유로운 운영체제 업그레이드, 보안 수준의 향상, 유연한 소프트웨어 개발 환경의 구축, 마이그레이션을 통한 부하 조정, 고장 격리(failure isolation)를 통한 서비스 가용성 향상, 파이어월 서비스, 공간 자원의 효율적 활용 등의 다양한 이점을 얻을 수 있다 [1]. 임베디드 시스템을 가상화하면 이러한 가상화의 장점들 중 많은 부분을 취할 수 있다. 특히 유지 보수 없이 오랜 시간 중단 없이 동작해야 하는 임베디드 시스템에서는 고장 격리를 통한 가용성의 향상은 임베디드 시스템의 운용에 큰 도움이 될 수 있다. 초기의 임베디드 시스템들은 간단한 단일 용도의 장치들이 대부분이었고 그 성능에도 한계가 있었기 때문에 가상화 기법들을 적용하기 어려웠다. 그러나 최근에는 임베디드 시스템에서도 고성능의 하드웨어가 사용됨에 따라 다양한 형태의 가상화 기법들을 임베디드 시스템에 적용하는 것이 가능해지고 있다. 특히 임베디드 시스템의 마이크로프로세서가 메모리 관리 장치(Memory Management Unit, MMU)를 가지고 있는 경우 좀 더 효율적인 메모리 가상화가 가능하다 [2, 3].

본 논문은 MMU 기능을 가진 마이크로프로세서를 가정한 가상 머신 모니터를 기반으로 동작하는 임베디드 시스템 상에 고장 감내 기법을 적용하여 높은 신뢰성과 고가용성을 가진 임베디드 시스템을 제안한다. 이 기법에서는 우선 가상화를 통해

• First Author: Sunghoon Son, Corresponding Author: Sunghoon Son
*Sunghoon Son (shson@smu.ac.kr), Dept. of Computer Science, Sangmyung University
• Received: 2017. 03. 29, Revised: 2017. 04. 14, Accepted: 2017. 05. 19.

임베디드 시스템을 가상 머신 모니터 상의 가상 머신으로 수행하게 하고, 동시에 별도의 백업용 가상 머신을 사용하여 임베디드 시스템의 동작을 주기적으로 백업하도록 한다. 만일 동작 중인 임베디드 시스템에서 오류가 발생하는 경우 해당 백업 가상 머신이 주 가상 머신의 역할을 대신하여 동작을 이어나가게 된다. 특히 이러한 가상 머신의 백업과 가상 머신 간의 전환은 가상 머신 모니터 상의 페이지 테이블의 조작을 통해 이루어지게 함으로써 중복으로 인한 성능 저하를 최소화하도록 하였다. 제안된 고장 감내 임베디드 시스템은 페이지 테이블 조작을 통해 오류를 일으킨 가상 머신 상의 게스트 운영체제 또는 그 태스크를 쉽게 복구할 수 있다. 성능 평가를 통해 제안된 기법이 임베디드 시스템의 성능을 크게 저하시키지 않으면서도 시스템의 가용성을 크게 향상시킬 수 있음을 보였다.

본 논문은 다음과 같이 구성된다. 1장에 이어, 2장에서는 임베디드 시스템에 대한 가상화와 고장 감내 기법 등에 대한 관련 기존 연구를 소개한다. 3장에서는 임베디드 시스템의 가상화를 활용한 고장 감내 시스템의 설계 및 구현 내용을 제안하고, 4장에서는 이 고장 감내 시스템에 대한 성능 측정 결과를 제시한다. 마지막으로 5장에서는 결론을 도출하고 향후 연구 방향을 모색한다.

II. Preliminaries

이 장에서는 우선 임베디드 시스템을 위한 고장 감내 기법에 대한 기존 연구들을 소개한다. 이어서 본 논문에서 제안하는 임베디드 시스템을 위한 고장 감내 기법의 기반이 되는 임베디드 시스템용 가상 머신 모니터를 소개한다.

1. Fault tolerance for embedded systems

고장 감내 시스템이란 일부 구성 요소의 고장에도 불구하고 정상적인 동작을 이어나갈 수 있도록 설계된 컴퓨터 시스템을 말한다. 지금까지 고장 감내 시스템에 대한 다양한 연구가 진행되어 왔으나 그 개념이나 관련 용어들이 정확하게 정의되고 사용되지는 않고 있다. [4]는 기존의 고장 감내 시스템과 관련한 다양한 사례 연구를 통해 고장 감내 시스템의 개념, 분류 및 주요 이론 등을 소개하고 있다.

전통적으로 고장 감내 기술은 국방, 의료, 항공 분야와 같이 단 한 번의 시스템 고장이 치명적인 결과로 이어지는 미션 크리티컬 시스템(mission-critical system)에 주로 적용되어 왔다. 또한 24시간/365일 서비스가 제공되어야 하는 고가용성을 요구하는 금융사의 전산 시스템이나 항공회사의 예약 시스템 등에도 고장 감내 시스템이 사용되고 있다.

흔히 고장 감내 시스템은 다음 네 가지 정도의 기본적인 요구사항을 가진다. 첫째, 단일 장애 지점(single point of failure)이 없어야 한다. 시스템 내 특정 부분에 오류가 발생하더라도 이에 대한 복구가 이루어지는 동안에도 시스템이 여전히 동작할 수 있어야 한다. 둘째, 고장이 발생한 경우 문제가 되는 컴포넌트로부터 고장을 격리할 수 있어야 한다. 이를 위해 고장 격리만을 목적으로 하는 탐지 메커니즘이 필요하다. 셋째, 고장이 시스템 내의 다른 부분으로 전파되는 것을 방지해야 한다. 오류 발생 후 해당 오류가 다른 오류의 발생을 일으키지 않도록 봉쇄할 수 있는 장치를 가져야한다. 넷째, 오류 복귀 시 복귀 개체의 가용성이 보장되어야 한다. 모든 오류에 대한 복구에서 복구할 개체에 대해 그 안정성을 보장해야한다.

기존의 고장 감내 시스템들은 주로 하드웨어 중복(redundancy)을 통해 고장에 대처하는 접근 방식을 사용한다. 같은 기능을 하는 두 개 이상의 동일 시스템을 중복 운영하면서 한 시스템의 고장 발생 시 다른 시스템이 이를 대체하게 하거나, 시스템의 주요 하드웨어 구성 요소를 이중으로 장착한 후, 한 구성 요소가 고장이 발생하는 경우 정상적으로 동작하는 다른 구성 요소가 이를 대체하는 방식이 대표적이다. 또 다른 형태로는 여분의 자원을 사용하여 주기적으로 처리 상태를 저장하고, 고장 발생 시 저장된 내용을 기반으로 고장 이전의 상태로 시스템을 복구하는 기법도 널리 사용되고 있다.

특히 최근 임베디드 시스템의 발전으로 임베디드 시스템 운영체제를 위한 고장 감내 기법에 대한 연구가 다양하게 진행되고 있다. 다만 임베디드 시스템의 경우 알려진 바와 같이 일반적인 컴퓨터 시스템에 비해 하드웨어 성능 상의 제약이 많기 때문에 기존의 고장 감내 시스템에 적용되었던 하드웨어 자원의 중복을 기반으로 하는 기술들은 적용하기 어렵다. 최근 이루어진 임베디드 시스템을 운영체제 수준의 고장 감내 기법들에 대한 연구는 다음과 같다.

우선 [5]에서는 리눅스 운영체제에서 디바이스 드라이버의 오류를 탐지하고 복구할 수 있는 섀도우 드라이버 (shadow driver) 개념을 소개하고 있다. 리눅스 시스템의 각 디바이스 드라이버마다 섀도우 드라이버를 두고, 디바이스 드라이버에서 오류가 발생하면 해당 드라이버를 섀도우 드라이버가 대체하도록 하여 리눅스 시스템의 신뢰성을 높이고 있다. [6]에서는 L4 마이크로커널을 기반으로 하는 일종의 가상화 시스템을 사용하여 고장을 격리시키고 시스템을 복구하는 방식을 취하고 있다. [7]에서는 모바일 장치를 기반으로 한 자체 회복(self-healing) 기능을 가진 Choices 운영체제를 소개하고 있다. Choices 운영체제는 모바일 장치의 고장에 대비하여 예외처리, 컴포넌트 격리 기법, 코드 재로드(reload), 운영체제 컴포넌트의 격리, 부분적인 재부팅 기법 등을 통한 자체 복구 기능을 가지고 있으며, 고장이 복구된 후에는 자동적으로 서비스를 재식작할 수 있도록 하고 있다. [8]에서는 리눅스 시스템의 커널 모듈에서 발생하는 오류를 리눅스 커널로부터 격리하여 커널의 신뢰도를 높일 수 있는 커널 자원 보호기를 제안하였다.

2. Virtualization for embedded systems

임베디드 시스템을 가상화하면 하나의 임베디드 장치에 여러 개의 운영체제를 동시에 실행하여 기능을 추가하는 것이 가능하다. 그러나 하드웨어 자원의 제약이 있는 임베디드 장치에 가상화

계층을 추가하는 것은 성능 상의 부담을 초래할 수도 있다. 그러나 최근의 임베디드 시스템 용 마이크로프로세서에는 대부분 효율적인 가상화를 지원하는 하드웨어 가상화 확장 (hardware virtualization extension) 기능이 포함되어 있다. 이러한 하드웨어 가상화 확장을 효과적으로 활용하면 임베디드 시스템에서도 성능 저하 없이 가상화의 장점을 충분히 누릴 수 있다. 그간의 임베디드 시스템의 가상화에 대한 연구는 [9]에 잘 정리되어 있다. 특히 이 논문에서는 CPU, 메모리, I/O 등에 대한 가상화와 관련된 이슈와 문제점들을 잘 정리하였고, 모바일 가상화 기법들의 효율성 평가를 위한 비교 분석 결과를 제시하고 있다.

[10]에서는 반가상화 (Paravirtualization) 기법에 기반을 둔 임베디드 시스템을 위한 가상 머신 모니터를 제안하고 있다. 이 시스템은 프로세서 가상화, 인터럽트 가상화, 가상화 관리 기법 등을 통해 각 게스트 운영체제에게 가상화된 하드웨어를 제공한다.

이 가상 머신 모니터는 사용자에게 시스템 관리 인터페이스를 제공하는 역할을 하는 VM0라는 특수한 가상 머신을 가지고 있다. VM0를 통해 사용자는 가상 머신을 쉽게 생성, 시작, 정지, 삭제 등을 수행할 수 있다. 각 가상 머신은 가상 머신 인터페이스를 통해 가상 머신 모니터에게 하드웨어 자원을 요청한다. 가상 머신 인터페이스는 공유 메모리를 기반으로 가상 머신이 다른 가상 머신이나 가상 머신 모니터와 통신할 수 있도록 하는 일련의 함수 집합이다.

제안된 가상 머신 모니터는 각 가상 머신이 사용하는 메모리를 관리하기 위하여 마이크로프로세서의 MMU 기능을 사용한다 [11]. 이를 위해 가상 머신 모니터는 일종의 페이지 테이블을 만들어 사용하고 있으며, 가상 머신들의 동작 중 발생하는 모든 메모리 접근 시에는 이 페이지 테이블을 통해 주소 변환이 이루어진다.

제안된 시스템의 전체 가상 주소 공간은 크게 고정 주소 영역, 동적 주소 영역, 오류 영역의 세 가지 영역으로 구성된다. 고정 주소 영역에서는 가상 주소와 물리 주소가 일대일로 대응되며, 플래시 메모리 영역, 페이지 테이블, 그리고 가상 머신 모니터 자체가 이 영역에 위치한다. 동적 주소 영역에서는 페이지 테이블을 기반으로 가상 주소가 물리 주소로 동적으로 변환되며 가상 머신 상의 게스트 운영체제와 그 태스크들이 이 영역에 위치한다. 이와 같이 구성함으로써 가상 머신 모니터는 가상 머신에서의 메모리 사용을 태스크 레벨까지 제어할 수 있다. 고정 주소 영역과 동적 주소 영역을 제외한 나머지 영역은 오류 영역으로 페이지 테이블에 의해 정의되지 않은 영역을 의미한다. 이 영역에 대한 접근은 모두 부적절한 메모리 접근으로 간주되어 적절한 예외 처리의 대상이 된다.

가상 머신 모니터는 두 단계로 이루어진 다단계 페이지 테이블을 가지고 있다. 첫 번째 단계의 페이지 테이블은 마스터 페이지 테이블로서 전체 가상 주소 공간을 담당한다. 고정 주소 영역의 페이지들, 플래시 메모리나 주변 장치를 위한 주소 공간은 이 레벨의 페이지 테이블이 담당한다. 두 번째 단계 페이지 테이블은 각 게스트 운영체제와 그 태스크를 위해 사용된다.

가상 머신 모니터의 동작 중에는 크게 두 가지 형태의 문맥 전환 (context switch), 즉 게스트 운영체제 간의 문맥 전환과 태스크 간의 문맥 전환이 발생한다. 태스크 수준의 문맥 전환의 경우 게스트 운영체제는 가상 머신 인터페이스를 통해 가상 머신 모니터에게 새로 수행할 태스크를 가상 머신 모니터에게 알린다. 가상 머신 모니터는 수행이 시작되는 태스크를 인지하고 해당 태스크의 가상 주소 공간을 활성화하도록 페이지 테이블을 조작한다. 게스트 운영체제 간의 문맥 전환의 경우, 가상 머신 모니터는 현재 실행 중인 게스트 운영체제의 문맥을 저장한 후 페이지 테이블을 조작해 다음 게스트 운영체제와 그 태스크를 위한 가상 주소 공간을 활성화한다.

III. Design and Implementation of Error Recovery Technique

본 장에서는 임베디드 시스템 용 가상 머신 모니터를 기반으로 MMU 기능을 활용한 고장 감내 시스템을 제안한다. 우선 제안된 고장 감내 시스템의 전체 메모리 레이아웃과 페이지 테이블 구조 등을 소개하고, 게스트 운영체제나 태스크에 오류가 발생하는 경우 이를 탐지하고 복구하는 기법에 대해 기술한다.

1. Memory layout of the proposed system

본 논문에서 제안하는 고장 감내 기법은 가상 머신 상에서 동작하는 임베디드 시스템에 대해 일정한 체크포인트마다 시스템의 동작 상태를 별도의 가상 머신에 저장해 두었다가, 오류 발생 시 백업 가상 머신에 저장해 두었던 정상 동작 상태의 이미지로 전환하는 방식을 사용한다. 특히 가상 머신의 백업 이미지 저장과 오류 발생 시 복구 과정을 효율적으로 수행하기 위해 일종의 페이지 테이블을 구성하고 프로그램이 메모리를 참조할 때 참조 주소가 이 페이지 테이블을 통해 변환되도록 구성한다. 본 논문에서 제안한 가상화된 임베디드 장치를 위한 고장 감내 시스템의 전체 메모리 구성은 Fig. 1과 같다.

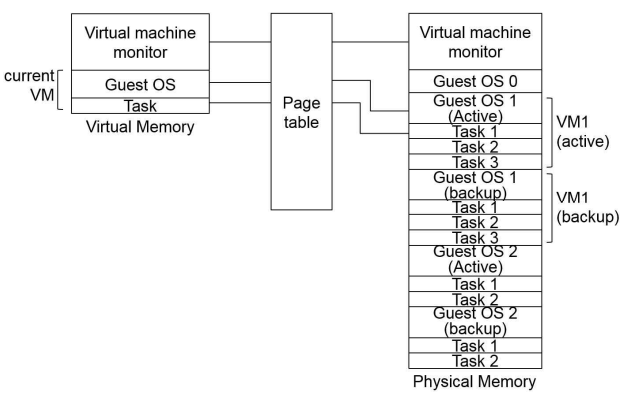

Fig. 1. Memory layout of the proposed system

Fig. 1에서 보는 바와 같이 한 가상 머신 상에서 동작하는 임베디드 시스템이 메모리상에서 차지하는 영역은 액티브 영역과 백업 영역으로 구성된다. 액티브 영역은 실행 중인 가상 머신이 정상적인 동작 중에 위치하는 영역이다. 반면 백업 영역은 정상 동작 중의 특정 시점에 해당 가상 머신의 메모리 이미지에 대한 복사본이 위치하고 있다. 오류가 발생하면 백업 영역은 액티브 영역으로 역할이 전환되어 정상 동작을 이어나가게 된다. 가상 머신 모니터 자신과 특수 가상 머신인 VM0는 별도의 백업 영역을 가지지 않는다.

모든 오류에 대한 탐지와 복구는 가상 머신 모니터에서 담당하므로, 게스트 운영체제나 그 태스크는 별도의 오류 탐지나 복구 메커니즘을 가지고 있지 않아도 된다.

2. Memory data structures

Fig. 2는 제안된 오류 복구 기능이 적용된 가상 머신 모니터가 관리하는 페이지 테이블 구조를 나타내고 있다. 또한 이 그림에서는 고장 감내 기법의 적용을 위한 관련 메모리 관련 자료구조들과 페이지 테이블과의 관계를 표시하고 있다.

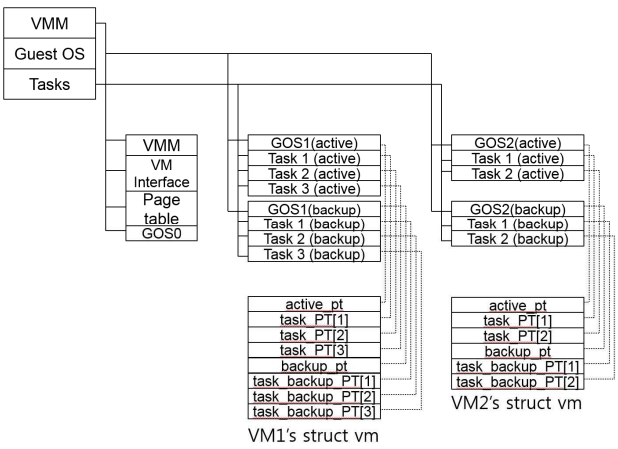

Fig. 2. Page tables and related data structures

앞 장에서도 간단히 언급한 바와 같이 제안된 가상 머신 모니터에서 사용하는 페이지 테이블은 두 단계로 구성된다. 첫 번째 단계의 페이지 테이블은 주 페이지 테이블로 전체 가상 주소 공간을 담당한다. 고정 영역이나 플래시 메모리 영역, 주변 장치를 위한 영역도 이 페이지 테이블에서 담당한다. 두 번째 단계의 페이지 테이블은 각 가상 머신 상의 게스트 운영체제와 그 태스크들, 그리고 이들에 대한 백업용 가상 머신을 위한 것이다.

이 그림은 가상 머신 모니터 커널 내의 오류 복구와 관련한 메모리 관리 자료구조들과 페이지 테이블 엔트리 간의 관계도 표시하고 있다. 제안된 가상 머신 모니터는 가상 머신을 관리하기 위해 각 가상 머신마다 struct vm 이라는 구조체를 두고 있는데, 이 구조체에는 active_pt와 backup_pt 라는 두 필드가 존재하며, 각각 액티브 가상 머신의 페이지 테이블과 백업 가상 머신의 페이지 테이블을 가리키게 된다. 시스템이 정상 동작 중에는 오직 active_pt를 통해 주소 변환이 이루어지지만 , 오류가 발생하면 가상 머신 모니터는 오류를 복구하기 위해 두 필드의 값을 교환한다. 또한 가상 머신 모니터는 게스트 운영체제상의 각 태스크와 그 복제본의 페이지 테이블을 위해 task_pt[]와 task_backup_pt[]라는 배열을 가지고 있다.

3. Error recovery

가상 머신 모니터는 시스템 동작 중에 가상 머신이 차지하고 있는 메모리 영역에 대한 백업을 실시하고 오류 발생 시 이를 기반으로 복구를 수행한다. 이러한 메모리 백업 과정은 흔히 커다란 시스템 부하 증가를 초래할 것으로 예상된다. 이를 방지하게 위해 본 논문에서 제안하는 시스템에서는 복구의 대상을 세분화 하여 백업을 수행한다. 앞서 언급한 메모리 구성에서 보는 바와 같이 가상 머신 모니터는 태스크 수준까지 페이지 테이블을 가지고 있기 때문에 이를 이용하면 스케줄링 과정에서 가상 머신 간 또는 태스크 간 실행이 전환되는 시점에 사용하던 메모리 영역을 효과적으로 백업할 수 있다. 특히 백업은 텍스트 영역을 제외한 BSS, 데이터, 스택, 힙 영역 등 시스템의 동작 중 내용이 변경되는 영역만을 대상으로 한다.

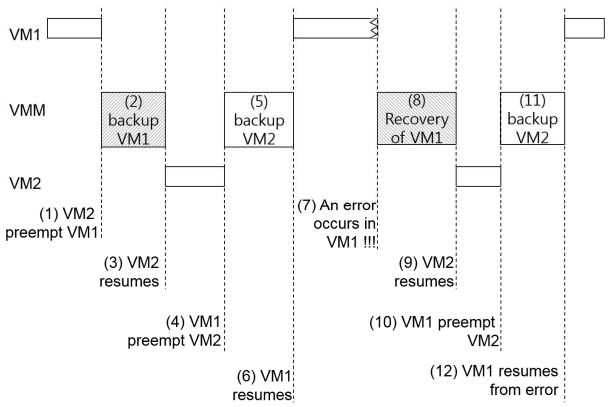

Fig. 3. Backup and recovery scenario

Fig. 3은 제안된 가상 머신 모니터에서 오류에 대비하여 가상 머신을 백업하고 이를 통해 오류 발생 시 복구하는 전체 과정을 보이고 있다. Fig. 3에서는 두 개의 가상 머신 VM1과 VM2 가 동작하는 시스템에서 이들 간의 전환 시 가상 머신이 백업되는 과정과 특히 VM1의 실행 중 오류가 발생한 후 복구되어 정상적으로 수행이 재개되는 과정을 보이고 있다. 매번 가상 머신의 전환이 일어날 때마다 가상 머신 모니터는 수행 중이던 가상 머신을 백업 영역에 저장한다(Fig. 3에서 단계 2, 5, 11에 해당). 시스템 동작 중 가상 머신 VM1에서 오류가 발생하면(단계 7에 해당), 가상 머신 모니터는 가장 최근 이루어진 VM1의 백업(단계 2에 해당)을 기반으로 복구를 수행하게 되고(단계 8), 이후 실행이 재개된 VM1은 정상적인 수행을 이어나가게 된다(단계 12). 특히 복구 과정은 오류가 발생한 액티브 영역과 정상적인 이미지가 저장되어 있는 백업 영역 간의 단순환 페이지 테이블의 교환으로 간단하게 가상 머신이 복구될 수 있도록 하였다.

태스크의 오류에 대해서도 유사한 동작이 이루어진다. 각 태스크의 전환 시마다 실행 중이던 태스크의 액티브 영역의 이미

지가 해당 태스크의 백업 영역으로 기록된다. 만일 태스크가 동작 중 오류가 발생하면 액티브 영역의 페이지 테이블과 백업 영역의 페이지 테이블의 서로 바뀌게 된다.

가상 머신의 복구와 태스크의 복구 간의 차이는 가상 머신의 복구에서는 해당 가상 머신에 속한 모든 태스크가 복구되지만, 태스크 복구에서는 오류를 일으킨 태스크만 교체된다는 점이다. 어느 경우이던지 가상 머신 모니터는 전체 오류 복구 과정을 책임지게 되며, 개별 가상 머신이나 태스크들은 이 과정과 무관하게 동작한다.

오류가 발생하면 MMU는 우선 오류 지점에 해당하는 메모리 주소를 특정 레지스터에 저장한 후 예외(exception)를 발생시킨다. 이 예외에 대한 처리는 가상 머신 모니터가 담당하게 되는데, 예외를 처리하는 과정에서 가상 머신 모니터는 해당 오류에 대한 복구를 수행하게 된다. 제안한 시스템에서는 오류를 발생시킨 주체에 따라 모두 세 가지 유형의 오류를 가정하고 있다. 오류의 유형은 오류 발생 시 예외 레지스터에 저장된 주소가 어느 영역에 해당하는지에 따라 결정된다.

제안한 시스템에서의 복구에 대한 기본 아이디어는 페이지 테이블 조작을 통해 오류가 발생한 영역을 백업 영역으로 교체하는 것이다. 예를 들어 오류가 발생한 태스크의 페이지 테이블을 새 페이지 테이블로 교체하는 것만으로 쉽게 오류 복구가 이루어지게 된다.

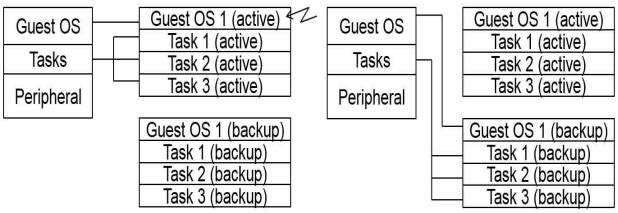

Fig. 4. Recovery from error generated by guest OS

Fig. 4는 게스트 운영체제에서 발생한 오류에 대해 복구하는 과정을 나타내고 있다. 가상 머신 모니터는 오류를 일으킨 게스트 운영체제의 페이지 테이블을 조작하여 게스트 운영체제의 액티브 영역을 백업 영역으로 대체함으로써 오류를 복구한다. 이 과정에서 해당 게스트 운영체제에 포함된 모든 태스크도 백업 영역의 태스크로 대체된다. 교체가 마무리되면 게스트 운영체제에 대한 또 다른 백업이 이루어진다. 전체 복구 과정은 다음 게스트 운영체제로의 문맥 전환이 이루어지면서 완성된다.

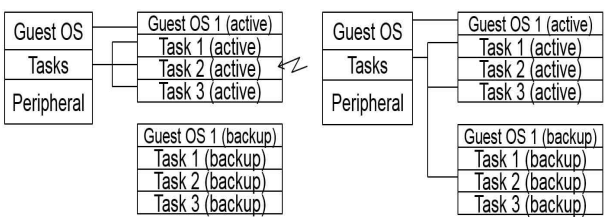

Fig. 5. Recovery from error generated by task

오류가 발생한 지점이 특정 태스크의 주소 공간에 해당하는 경우, 가상 머신 모니터는 태스크 오류 복구 과정을 시작한다. Fig. 5는 태스크가 발생시킨 오류를 복구하는 과정을 나타낸다. 가상 머신 모니터는 오류를 발생한 태스크(Fig. 5에서 태스크 2에 해당)에 해당하는 페이지 테이블 엔트리를 조작하여 태스크의 액티브 영역을 백업 영역으로 대체함으로써 오류를 복구한다. 또한 마스터 페이지 테이블의 해당 엔트리도 갱신한다. 이 과정은 다음 태스크로의 문맥교환으로 마무리된다.

만일 오류가 발생한 주소가 어느 게스트 운영체제나 태스크에도 속하지 않은 영역이라면 이 오류는 가상 머신 모니터의 문맥에서 처리되어야한다. 이 경우는 가상 머신 모니터에서 오류가 발생한 것으로써 전체 시스템이 즉시 종료하게 된다.

4. Implementation

제안한 시스템은 ARM920T 프로세서를 기반으로 하는 S3C2440 AP를 사용하는 개발 보드 상에서 이루어졌으며 LINUX 호스트 상에서 C언어 및 ARM 어셈블리어를 통해 개발 및 실험이 이루어졌다. 가상 머신 모니터 상에서 동작하는 게스트 운영체제로는 MMU를 사용하도록 수정한 MicroC/OS-II 운영체제[12] 버전 2.52를 대상으로 진행되었다.

또한 게스트 운영체제로 사용된 MicroC/OS-II를 가상 머신 모니터 상에서 동작할 수 있도록 수정하였다. 가상 머신에서 동작하는 운영체제는 하드웨어를 직접 접근하는 대신 모든 자원에 대한 접근을 가상 머신 인터페이스를 통해 가상 머신 모니터에게 요청해야 한다. 이를 위해 게스트 운영체제에 대해 크게 네 가지 정도의 수정이 이루어졌다.

첫 번째로 게스트 운영체제의 CPU 레지스터 접근 부분을 수정하였다. 기존 CPU 레지스터에 직접 접근하여 상태를 변경하였으나, 이를 가상 머신 인터페이스를 통해 가상 머신 모니터 내의 가상 레지스터 자료구조를 갱신하는 것으로 수정하였다. 기존 코드 내의 SRCPDN, INTPND 등의 인터럽트 관련 레지스터에 대한 접근은 모두 이러한 방식으로 수정하였다. 두 번째 여러 게스트 운영체제가 각자 관리하던 우선순위 테이블을 가상 머신 모니터가 통합하여 관리하도록 수정하였다. 이를 위해 기존 코드에서 우선순위 테이블을 참조하거나 수정하는 코드는 가상 머신 모니터 내의 통합 우선순위 테이블을 사용하도록 수정하였다. 이와 관련하여 MicroC/OS-II 운영체제의 태스크 관련 함수 중 OSTaskCreate(), OSTaskResume(). OSTaskDel(), OSTaskSuspend() 등, 이벤트 관련 함수 중 OSEventTaskRdy(), OSEventTO(), OSEvevntTaskWait() 등, 뮤텍스 관련 함수 중 OSMutexPend(), OSMutexPost(), 시간 관련 함수 OSTimeDlyResume(), OSTimeTick(), OSTimeDly() 등을 수정하였다. 세 번째 게스트 운영체제가 사용하는 메모리 영역 중 가장 첫 주소에 예외 처리 벡터가 위치하도록 수정하고, 가상 머신 모니터 내의 예외 처리 벡터 주소를 저장하는 자료 구조에 해당 가상 머신의 첫 주소를 등록하였다. 마지막으로 가상 머신 모니터에서만 모든 장치를 초기화하도록 게스트 운영체제의 장치 초기화 부분을 제거하였다.

IV. Performance Evaluation

본 장에서는 제안한 가상화된 임베디드 시스템 상에서의 오류 복구 기능이 정상적으로 동작하는지 확인하고, 이 기법의 적용이 시스템의 전체 성능에 어떠한 영향을 미치는지에 대해 성능 평가를 실시한 결과를 제시한다.

1. Demonstration

본 절에서는 메모리 접근 오류에 대한 복구 동작이 정상적으로 이루어지는지 여부를 확인한다. 이를 위해 여러 가상 머신이 동작하는 상황에서, 특정 가상 머신에서 한 사용자 태스크에 게스트 운영체제의 커널 영역을 접근하는 코드를 삽입하여 메모리 접근 오류를 인위적으로 발생시키고, 가상 머신 모니터가 이를 감지하여 정상 상태로 복구하는지 여부를 확인하였다.

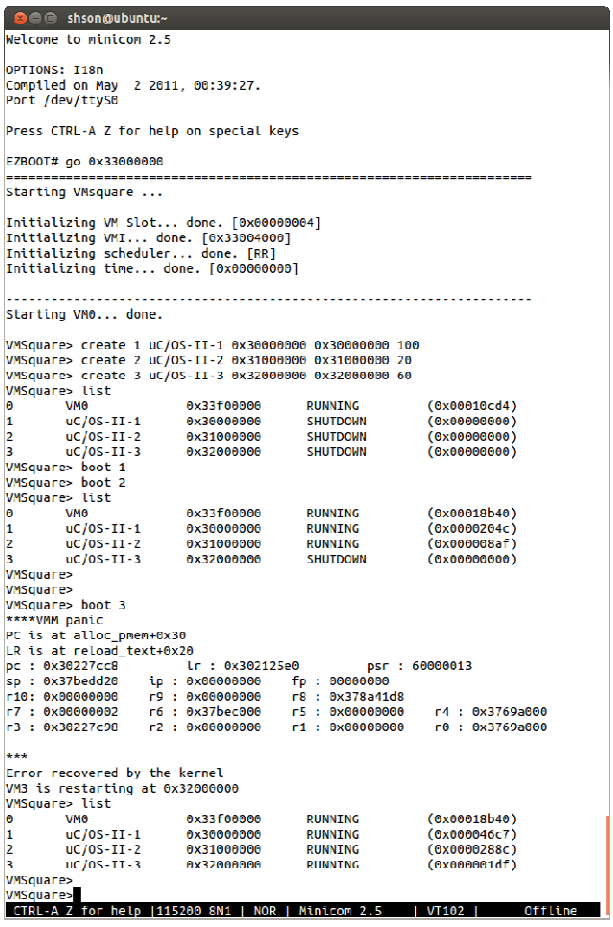

Fig. 6. Demonstration of error recovery

Fig. 6은 가상 머신 uC/OS-II-1, uC/OS-II-2, uC/OS-II-3을 실행하는 과정을 보이고 있다. 이 중 uC/OS-II-1과 uC/OS-II-2는 오류 없이 정상 동작하는 가상 머신이고, uC/OS-II-3는 게스트 운영체제의 커널 영역에 대한 접근을 시도하는 오류를 가진 태스크를 포함하고 있는 가상 머신이다. 이 태스크는 게스트 운영체제인 uC/OS-II 커널의 전역 변수인 OSTime의 값을 0으로 설정하려고 시도하는 코드를 포함하고 있다. 명령 boot를 사용하여 먼저 두 개의 정상적인 가상 머신을 시작시킨 후, 세 번째 가상 머신을 동작시키면 이 태스크가 커널 전역 변수에 대한 접근을 시도하는 과정에서 권한이 없는 메모리 영역에 대한 접근 시도로 인해 데이터 어보트 예외(data abort exception)가 발생하게 된다. 가상 머신 모니터는 이 예외를 처리하는 과정에서 예외를 일으킨 지점이 특정 태스크의 영역에 해당함을 인지하고, 이 태스크의 백업 페이지 테이블을 주 페이지 테이블로 전환시켜 정상 수행을 이어나가게 된다.

본 논문에서 제안한 오류 복구 기법은 다양한 형태의 메모리 접근 오류에 대한 복구에 중점을 두고 있다. 좀 더 구체적으로 말하자면, 위에서 예를 든 바와 같이 ARM 프로세서가 정의한 데이터 어보트 예외를 발생시키는 메모리 접근 오류를 복구할 수 있다. ARM에서 데이터 어보트 예외는 잘못된 주소 공간에 대해 읽기/쓰기를 시도할 때 MMU에 의해서 발생하는 예외이다. 예를 들어 접근하려는 주소의 정렬이 잘못되었거나, 권한이 없는 주소를 접근하거나, 해당 디스크립터에 유효하지 않다는 설정이 되어 있는 경우 등에서 이 예외가 발생한다. 현재 구현되어 있지는 않지만 메모리 접근 오류 외에도 ARM에서 사용하는 다른 유형의 예외를 일으키는 오류에 대해서도 쉽게 대처가 가능하다. 예를 들어 undefined instruction 예외나 prefetch abort 예외 등의 경우에도 예외의 종류만 다를 뿐 예외 처리에 대한 메커니즘은 동일하거나 매우 유사하므로 이에 대한 처리를 쉽게 추가할 수 있다.

2. Performance measurement

오류 복구 기법을 적용한 시스템의 성능에 가장 직접적으로 영향을 미칠 것으로 예상되는 가상 머신 간 문맥 전환에 소요되는 시간을 측정하였다. 오류 복구 기법을 적용한 시스템에서는 문맥 전환이 페이지 테이블에 대한 조작 등을 포함하고 있기 때문에 기존에 비해 문맥 전환에 시간이 더 걸릴 것으로 예상되었다. 실험은 태스크 간의 문맥전환과 게스트 운영체제간의 문맥전환으로 나누어 이루어졌으며, 오류 복구 기능을 적용하지 않은 버전의 가상 머신 모니터를 비교 대상으로 하여 측정하였다.

제안한 가상 머신 모니터는 가상 머신 간 전환 시 비트맵 방식의 우선순위 테이블을 사용하여 다음에 수행할 가상 머신을 결정하는데, 이러한 방법은 현재 수행 중이 가상 머신의 개수와 관계없이 항상 일정한 시간에 다음 수행할 가상 머신을 찾을 수 있다는 장점을 가지고 있다. 이러한 특징은 실시간 응용을 수행하는 임베디드 시스템의 가상화에서 매우 중요하다. 전체 가상 머신 간 전환 과정은 이전 가상 머신의 CPU 레지스터 저장, 다음 수행할 가상 머신의 선택 및 전환, 다음 가상 머신의 CPU 레지스터 복원 등으로 이루어진다. 따라서 각 단계의 소요 시간을 측정하면 정확한 소모 시간을 파악할 수 있다.

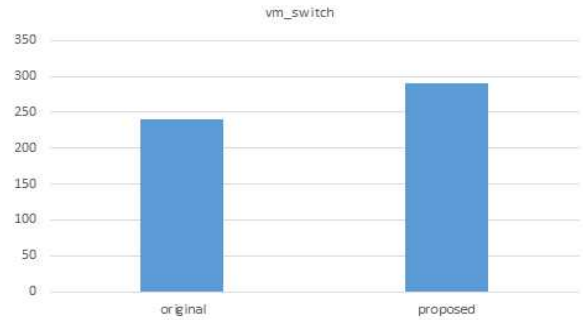

Fig. 7. Comparison on virtual machine switch latency

우선 오류 복구 기능이 적용되지 않은 가상 머신 모니터 상에서의 가상 머신 간 전환 시간을 측정하였다 (Fig. 7). S3C2440의 내부 하드웨어 타이머를 사용하여 가상 머신 전환에 소요된 시간을 측정한 결과 한 번의 가상 머신 간 전환에는 240 ns 가 소요되었다. 이를 전환의 세부 단계 별로 나누어 보면, 이전 가상 머신의 CPU 레지스터 저장 시간에 40 ns, 다음 가상 머신으로의 전환에 160 ns, 새 가상 머신의 CPU 레지스터 복원에 40 ns 가 소요되었다. 이 값들은 10회 이상의 반복 수행으로 얻어낸 결과 값이며 항상 동일한 결과를 나타낸다.

한편 오류 복구 기능이 추가된 가상 머신 모니터 상에서 가상 머신 전환에 걸리는 시간은 290 ns로 측정되었다. 이를 단계 별로 나누어 보면 가상 머신의 CPU 레지스터를 저장, 복원하는 데에는 오류 복구 기능이 포함되지 않은 경우와 동일하게 각 40 ns 가 소요되었으나, 다음 가상 머신으로의 전환에는 오류 복구 기능 포함 전의 경우보다 증가한 210 ns가 소요되었다. 이러한 결과는 시험 전에 예측된 바와 같이 새로운 가상 머신으로 전환하는 과정에서 가상 머신의 메모리 영역을 백업하는 등 오류 복구를 위한 추가적인 동작이 포함되었기 때문이다. 전체적으로 오류 복구 기능이 추가됨에 따라 가상 머신 간의 전환 시간이 이전보다 20.8% 증가했음을 확인하였다.

오류 복구 기능이 포함된 가상 머신 모니터의 경우 가상 머신 간 전환에 걸리는 시간은 전환 대상이 되는 가상 머신에 따라 달라진다. 즉 백업 영역이 큰 가상 머신의 경우 그만큼 전환에 많은 시간이 걸리게 된다. 다만 백업 범위는 실제 사용하는 메모리 공간뿐만 아니라 대상 영역 전체에 대해 백업하도록 하여, 백업에 항상 일정한 시간이 걸리도록 하였다.

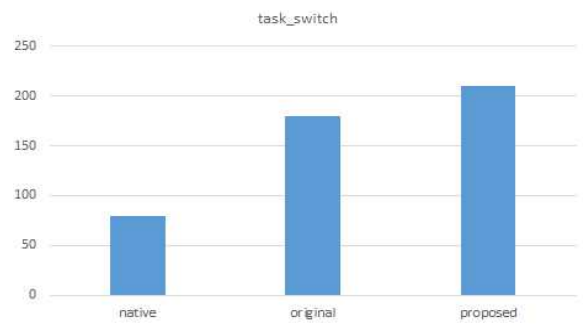

Fig. 8. Comparison on task switch latency

다음으로 한 게스트 운영체제 내에서 태스크 간 실행 전환에 걸리는 시간 변화에 대해 측정하였다 (Fig. 8). 우선 오류 복구 기법을 적용하기 전의 가상 머신 모니터 상에서 게스트 운영체제 내의 태스크 간 전환 시간은 180 ns로 측정되었다. 세부적으로는 CPU 레지스터의 저장과 복원에 각 40 ns, 그리고 다음 태스크로의 전환에 100 ns가 걸리는 것으로 나타났다. 특히 게스트 운영체제로 사용된 MicroC/OS-II도 비트맵 기반의 우선순위 테이블을 사용하기 때문에 태스크 개수에 관계없이 항상 일정한 시간 내에 새로운 태스크를 선택하는 것이 가능하다 (참고로 그림의 좌측 막대는 가상화를 적용하지 않은 경우의 태스크 간 문맥 교환 시간을 나타낸다. 즉, 동일한 운영체제를 동일한 하드웨어 상에서 가상화 없이 수행하는 경우의 측정 결과로, 제안한 기법을 적용하기 전 단순 가상화의 경우보다 짧은 문맥 전환 시간을 보이고 있다).

한편 오류 복원 기법을 적용한 시스템 상에서는 게스트 운영체제 내의 태스크 간 전환에는 210 ns가 걸리는 것으로 측정되었다. 이 경우에서 레지스터 저장 및 복원에는 동일한 시간이 소요되었으나 다음 태스크로의 전환 시간이 100 ns에서 130 ns로 증가되었다. 전체적으로는 오류 복구 기능을 적용함으로써 태스크 간 전환 시간이 16.7% 정도 증가했음을 확인하였다.

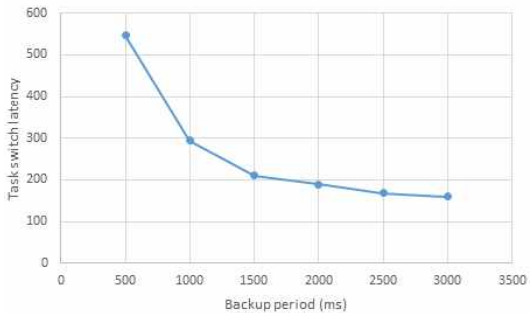

Fig. 9. Task switch latency

Fig. 9는 백업 주기에 따른 태스크 간 전환 시간을 나타낸다. 앞서 언급한 바와 같이 백업 주기가 짧을수록 오류 발생 시 가장 최신의 상태로 복구할 수 있다. 하지만 짧은 복구 주기는 잦은 백업을 의미하므로 이는 시스템의 성능에 많은 영향을 미치게 된다. 반면 복구 주기가 긴 경우 오류 발생 시 복구까지의 시스템 정지 시간은 길어지겠지만 정상 동작 중에 시스템 부하에 미치는 영향은 줄어들 수 있다. 따라서 성능과 복구를 위한 정지 시간 간에 적절한 지점을 정하는 것이 중요하다.

성능 평가를 통해 제안된 오류 복구 기법을 사용하는 경우 예상된 바와 같이 가상 머신 간의 전환이나 태스크 간의 전환에 다소간의 성능 저하가 있는 것으로 나타났다. 이러한 성능 저하는 오류 복구 기능을 위한 백업 과정에서 기인하는 것으로, 성능 저하의 정도가 그다지 크지 않은 뿐만 아니라, 임베디드 시스템과 같이 오랜 시간 동안 오류 없이 동작해야 하는 환경에서는 다소간의 성능 저하보다는 오류 복구 기능의 적용으로 시스템의 가용성을 높이는 것이 더 중요한 사안이 될 수 있다.

특히 가상 머신에 따라 백업에 걸리는 시간이 늘 일정하도록 하였기 때문에 전환 시간이 늘어나더라도 항상 동일한 가상 머신과 태스크들만이 실행되는 임베디드 시스템의 특성 상 문제가 없으리라 여겨진다.

V. Conclusions

임베디드 시스템의 하드웨어 성능 제약이 줄어들면서 임베디드 시스템에도 가상화를 적용하는 사례가 늘고 있다. 가상화 기술을 활용하면 임베디드 시스템에서도 일반적인 가상화의 장점들을 모두 누릴 수 있다. 특히 오랜 시간 중단 없이 동작해야 하는 환경에서는 임베디드 시스템의 가상화를 통해 시스템의 가용성을 높이는 것이 큰 도움이 될 수 있다. 본 논문에서는 MMU 기능을 가진 마이크로프로세서 상에서 동작하는 가상 머신 모니터를 기반으로 가상화를 활용한 고장 감내 기법을 적용하여 높은 신뢰성과 고가용성을 가진 임베디드 시스템을 제안하였다. 제안된 고장 감내 임베디드 시스템은 페이지 테이블을 조작을 통해 오류 발생 시 효과적으로 오류를 복구하여 전상적인 동작을 가능하게 한다. 성능 평가를 통해 제안된 기법이 임베디드 시스템의 성능을 크게 저하시키지 않으면서도 시스템의 가용성을 크게 증가시킬 수 있음을 보였다.

REFERENCES

[1] J. Sahoo, S. Mohapatra, and R. Lath, “Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues,” Proceedings of the Second International Conference on Computer and Network Technology, pp. 222-226, Bangkok, Thailand, April 2010.

[2] B. Egger, J. Lee and H. Shin, “Dynamic Scratchpad Memory Management for Code in Portable Systems with an MMU,” ACM Transactions on Embedded Computing Systems, Vol. 7, No. 2, pp. 1-38, February 2008.

[3] X. Zhou and P. Petrov, “Towards Virtual Memory Support in Real-Time and Memory-Constrained Embedded Applications: The Interval Page Table,” IET Computers and Digital Techniques, Vol. 5, No. 4, pp. 287-295, July 2011.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic Concepts and Taxanomy of Dependable and Secure Computing,” IEEE Trans. on Dependable and Secure Computing, Vol. 1, No. 1, pp. 11-33, January 2004.

[5] M. M. Swift, M. Annamalai, B. N. Bershad, and H. M. Levy, “Recovering Device Drivers,” ACM Trans. on Computer Systems, Vol. 24, No. 4, pp.333-360, November 2006.

[6] Gernot Heiser, “The Role of Virtualization in Embedded Systems,” Proceedings of the First Workshop on Isolation and Integration in Embedded Systems, pp. 11-16, Glasgow, UK, April 2008.

[7] F. M. David and R. H. Campbell, “Building a Self-Healing Operating System,” Proceedings of the Third International Symposium on Dependable, Autonomic and Secure Computing, pp. 3-10, Columbia, Maryland, USA, September 2007.

[8] J. Choi, S. Baek, and S. Y. Shin, “Design and Implementation of a Kernel Resource Protector for Robustness of Linux Module Programming,” Proceedings of the 21st Annual ACM Symposium on Applied Computing, pp. 1477-1481, Dijon, France, April 2006.

[9] J. Shuja, A. Gani, K. Bilal, A. Khan, S. A. Madani, S. U. Khan, and A. Y. Zomaya, “A Survey of Mobile Device Virtualization: Taxonomy and State of the Art,” ACM Computing Surveys, Vol. 49, No. 1, pp. 1-36, July 2016.

[10] Sunghoon Son and Jaehyeon Lee, “Design and Implementation of Virtual Machine Monitor for Embedded Systems,” Journal of The Korea Society of Computer and Information, Vol. 14, No. 1, pp. 57-64, January 2009.

[11] S. Son, “An MMU Virtualization for Embedded Systems,” Lecture Notes in Electrical Engineering 215, pp. 247-252, January 2013.

[12] J. J. Labrosse, “MicroC/OS-II The Real-Time Kernel Second Edition,” CMP Books, 2002.

Authors

Sunghoon Son received his B.S., M.S. and Ph.D. degrees in Computer Science from Seoul National University, Korea, in 1991, 1993 and 1999, respectively. Dr. Son joined the faculty of the Department of Computer Science at Sangmyung University, Seoul, Korea, in 2004. He is currently a Professor in the Department of Computer Science, Sangmyung University. He is interested in system software, embedded system, and virtualization.