JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 5, pp. 11-19, May 2022
https://doi.org/10.9708/jksci.2022.27.05.011

Improving Availability of Embedded Systems Using Memory Virtualization

Sunghoon Son*

*Professor, Dept. of Computer Science, Sangmyung University, Seoul, Korea

[Abstract]

In this paper, we propose a fault tolerant embedded system using memory redundancy on the full-virtualization based virtual machine monitor. The proposed virtual machine monitor first virtualizes main memory of embedded system utilizing efficient shadow page table scheme so that the embedded system runs as a virtual machine on the virtual machine monitor. The virtual machine monitor makes the backup of the embedded system run as another virtual machine by copying memory contents of the embedded system into memory space of backup system according to predefined schedules. When an error occurs in the target virtual machine, the corresponding standby virtual machine takes the role of target virtual machine and continues its operation. Performance evaluation studies show that such backups and switches of virtual machines are performed with minor performance degradation.

▸ **Key words**: Embedded system, Memory virtualization, Shadow page table, Fault tolerance, Availability

[요 약]

본 논문에서는 전가상화 방식의 가상 머신 모니터를 기반으로 메모리 중복을 통한 고장 감내 기능을 적용한 임베디드 시스템을 제안한다. 제안된 가상 머신 모니터는 우선 효율적인 섀도우 페이지 테이블 기법을 사용하여 메모리를 가상화한다. 이를 기반으로 대상 임베디드 시스템을 하나의 가상 머신으로 동작하게 하는 한편, 동일한 시스템을 별도의 가상 머신에서 동작하도록 백업 시스템을 구축함으로써 대상 임베디드 시스템의 메모리 영역이 미리 정해진 시점과 대상에 따라 백업 시스템의 메모리 공간으로 복사되도록 하였다. 이렇게 중복이 이루어진 임베디드 시스템은 고장이 발생하면 백업 시스템으로 전환하여 정상적인 동작을 이어나가게 된다. 성능 평가를 통해 제안된 기법이 임베디드 시스템의 성능을 크게 저하시키지 않으면서도 시스템의 가용성을 크게 향상시킬 수 있음을 확인하였다.

▸ **주제어**: 임베디드 시스템, 메모리 가상화, 섀도우 페이지 테이블, 고장 감내, 가용성

• First Author: Sunghoon Son, Corresponding Author: Sunghoon Son
*Sunghoon Son (shson@smu.ac.kr), Dept. of Computer Science, Sangmyung University
• Received: 2022. 04. 13, Revised: 2022. 04. 26, Accepted: 2022. 04. 27.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

다양한 산업 분야에 컴퓨터 기술이 적용되면서 국방, 의료, 자동차 등 고도의 신뢰성과 가용성을 요구하는 분야에서도 임베디드 시스템이 사용되고 있다. 이러한 환경의 임베디드 시스템은 오랜 시간 동안 고장 없이 동작하도록 하는 것이 중요한 설계 목표 중 하나이고, 이를 위해 임베디드 시스템에 발생하는 고장을 진단하고 격리하거나 고장으로부터 복구할 수 있는 고장 감내 (fault tolerance) 기술을 적용하는 것이 필요하다. 그간 고장 감내 기술은 주로 대형 컴퓨터 시스템 위주로 연구되어 왔으며, 비용에 상관없이 하드웨어와 소프트웨어 자원을 중복 사용하거나, 동일한 사양을 몇 가지 다른 방식으로 구현한 시스템을 운영하는 방식으로 제공되었다. 최근에는 임베디드 시스템을 위한 고장 감내 기술에 대한 연구도 다양하게 진행되고 있으나, 임베디드 시스템의 경우 범용 컴퓨터 시스템에 비해 하드웨어 자원 상의 제약이 많기 때문에 기존에 사용되어 온 중복에 의한 고장 감내 기술을 그대로 적용하기는 어렵다.

한편 가상화(virtualization) 관련 연구가 폭넓게 진행되면서 임베디드 시스템에도 가상화 기술을 적용하는 사례가 늘고 있다. 임베디드 시스템을 가상화하면 기존의 가상화 기술이 제공하는 여러 장점들을 취할 수 있다. 특히 임베디드 시스템을 가상 머신 상에서 수행시킴으로써 해당 시스템의 동작 중 고장이 발생하더라도 이 고장이 전체 시스템으로 전파되지 않고 가상 머신 상에만 한정되도록 격리하는 것이 가능하다. 또한 다수의 가상 머신을 생성하여 각각 동일한 임베디드 시스템을 동작시킴으로써 중복을 통한 고장 감내 효과도 추구할 수 있다. 이처럼 가상화를 통한 고장 감내 기술을 임베디드 시스템에 적용하는 것은 유지 보수 없이 오랜 시간 동작해야 하는 환경에서 수행하는 임베디드 시스템에는 중요한 기능이 될 것이다.

임베디드 시스템에 가상화를 적용하는 경우 시스템의 주요 하드웨어 자원인 마이크로프로세서, 메모리, 디바이스 등을 각각 가상화하게 된다. 특히 임베디드 시스템의 마이크로프로세서가 가상 메모리(virtual memory)를 위한 MMU(Memory management unit) 기능을 가지고 있고, 가상 머신 상에서 동작하는 임베디드 운영체제도 이를 이용한 가상 메모리 기능을 제공하는 경우 메모리 가상화는 중요한 이슈 중 하나이다. 특히 고장에 대비하여 여러 개의 가상 머신에 동일한 임베디드 운영체제를 설치하여 운영하고자 할 때 가상 머신들 간에 동일한 메모리 이미지를 사용한다는 점을 적극적으로 활용할 수 있는 효율적인 메모리 가상화 기법을 적용하는 것이 중요하다. 섀도우 페이지 테이블 (shadow page table) 기법은 널리 사용되는 메모리 가상화 기법 중 하나이다. 이 기법은 가상 머신 상에서 동작하는 운영체제, 즉 게스트 운영체제의 페이지 테이블로부터 주소 변환에 대한 정보를 가져와서 섀도우 페이지 테이블을 구성하고, 메모리 접근 시 MMU가 이 섀도우 페이지 테이블을 참조하여 주소 변환을 하도록 하는 기법이다. 섀도우 페이지 테이블 기법을 설계할 때 중요한 사항 중 하나는 게스트 운영체제 상에서 동적으로 변경되는 페이지 테이블의 내용을 섀도우 페이지 테이블에 반영하는 과정에서 오버헤드를 최소화할 수 있는 효율적인 동기화 기법을 제공하는 것이다.

본 논문에서는 전가상화 (full virtualization) 방식의 가상 머신 모니터(virtual machine monitor)를 기반으로 메모리 중복을 통한 고장 감내 기능을 제공하는 신뢰성 높은 임베디드 시스템을 제안한다. 제안된 가상 머신 모니터는 우선 섀도우 페이지 테이블 기법을 사용하여 메모리를 가상화한다. 특히 가상 머신 상의 게스트 페이지 테이블과 완전히 동기화된 방식으로 섀도우 페이지 테이블을 관리하기보다 섀도우 페이지 테이블의 수정을 최소화함으로써 섀도우 페이지 테이블 운용의 오버헤드를 줄이고자 하였다. 이러한 메모리 가상화를 기반으로 다음과 같이 고장 감내 시스템을 구축하였다. 일단 대상 임베디드 시스템을 가상 머신 상에서 동작하도록 하고, 별도의 가상 머신 상에서 대상 가상 머신에 대한 백업 가상 머신을 구축한 후, 대상 임베디드 시스템의 메모리 영역을 미리 정해진 시점마다 백업 시스템의 메모리 영역으로 복사하도록 하였다. 특히 이러한 메모리 백업은 가상 머신 모니터가 섀도우 페이지 테이블을 조작하는 과정에서 부가적으로 이루어지게 되며, 메모리 간 복사로 인한 오버헤드를 줄이기 위해 메모리 백업의 횟수나 대상을 최소화할 수 있도록 설계하였다. 제안된 고장 감내 임베디드 시스템은 섀도우 페이지 테이블에 대한 조작을 통해 오류를 일으킨 가상 머신 상의 임베디드 운영체제를 쉽게 복구할 수 있다. 성능 평가를 통해 제안된 기법이 임베디드 시스템의 성능을 크게 저하시키지 않으면서도 시스템의 가용성을 크게 향상시킬 수 있음을 확인하였다.

논문의 전체적인 구성은 다음과 같다. 1장의 서론에 이어, 2장에서는 사례 연구를 중심으로 고장 감내 기법의 임베디드 시스템에의 적용 가능성과 메모리 가상화를 기반으로 한 고장 감내 시스템에 대해 소개한다. 3장에서는 메모리 가상화를 활용한 고장 감내 시스템의 설계 및 구현 내용을 제안하고, 4장에서는 이 고장 감내 시스템에 대한 성능 측정 결과를 제시한다. 마지막으로 5장에서 결론과 향후 연구 방향을 제시한다.

II. Preliminaries

이 장에서는 임베디드 시스템에서의 고장 감내 기법에 대해 필요성에 대해 설명하고 이와 관련된 기존의 연구 결과들을 소개한다. 이어서 메모리 가상화 기법을 중심으로 임베디드 시스템의 가상화에 대해 알아보고, 메모리 가상화 기법이 어떻게 임베디드 시스템을 위한 고장 감내 기법으로 활용될 수 있는지 밝힌다.

1. Fault-tolerant embedded system

고장 감내 시스템이란 일부 하드웨어 또는 소프트웨어 구성 요소에 장애가 발생하더라도 전체적으로는 정상적인 동작을 이어나갈 수 있도록 설계된 컴퓨터 시스템을 일컫는다. [1]은 고장 감내 시스템에 대한 다양한 사례 연구를 통해 고장 감내 시스템의 개념, 분류 및 주요 이론 등을 잘 정리하고 있다. 최근에는 임베디드 시스템을 위한 고장 감내 기법에 대한 연구도 활발하게 진행되고 있는데, 다만 임베디드 시스템은 흔히 범용 컴퓨터 시스템에 비해 성능상 제약이 많기 때문에 기존의 복제(replication)나 중복(redundancy)을 기반으로 하는 고장 감내 기술들을 그대로 적용하기는 어렵다. 임베디드 시스템에 적용할 만한 운영체제 수준의 고장 감내 기법들에 대한 연구는 다음과 같다. [2]는 리눅스 운영체제에서 디바이스 드라이버에서 발생하는 오류를 탐지하고 복구할 수 있는 섀도우 드라이버(shadow driver)를 소개하고 있다. 섀도우 드라이버는 디바이스 드라이버에 중복 기술을 적용한 것으로, 시스템의 각 디바이스 드라이버마다 섀도우 드라이버를 두어, 디바이스 드라이버의 동작에 오류가 발생하면 이를 섀도우 드라이버가 대체하도는 방식으로 리눅스 시스템의 신뢰성을 높이고 있다. [3]은 리눅스 운영체제의 로더블 커널 모듈(loadable kernel module)에서 발생한 오류가 리눅스 커널로 전파되지 않도록 차단하여 리눅스 시스템의 신뢰도를 높일 수 있는 커널 자원 보호기(kernel resource protector)를 제안하였다. [4]는 모바일 장치를 위한 Choice 운영체제를 소개하고 있다. Choice 운영체제는 모바일 장치의 고장 시 적용할 수 있는 다양한 기능들을 갖추고 있으며, 여기에는 운영체제 컴포넌트의 격리, 코드 리로드(code reload), 부분적 재부팅 등이 포함된다.

2. Virtualization for embedded system

가상화 기술을 적용하면 한 컴퓨터 시스템에서 여러 개의 가상 머신을 동시에 실행할 수 있다. 이를 통해 고장 격리(failure isolation)를 통한 시스템 가용성 증대 등의 이점을 얻을 수 있다[5]. 기존의 임베디드 시스템은 흔히 하드웨어 성능 상의 제약으로 이러한 가상화 기술을 적용하는데 많은 어려움이 있었으나, 임베디드 시스템에도 고성능 하드웨어가 사용됨에 따라 다양한 형태의 가상화 기술이 폭넓게 적용됨으로써 별도의 성능 저하 없이 가상화의 장점들을 충분히 누릴 수 있게 되었다[6].

일반적으로 컴퓨터 시스템을 가상화하기 위해서는 마이크로프로세서, 메모리 및 각종 I/O 디바이스 등을 각각 가상화한다. 이 중 메모리 가상화의 경우 임베디드 시스템의 마이크로프로세서가 MMU를 가지고 있는 경우 효율적인 메모리 가상화가 가능하다[7][8]. 특히 섀도우 페이지 테이블 기법은 가상 머신 상에서 동작하는 게스트 운영체제가 가상 메모리를 사용하는 경우를 지원하기 위한 메모리 가상화 기법이다 [9]. 섀도우 페이지 테이블은 전체 시스템 차원에서 MMU의 주소 변환 시 참조하는 페이지 테이블로서 게스트 운영체제 상의 프로세스가 만들어내는 가상 주소를 최종 물리 주소로 변환하는데 필요한 매핑들을 제공한다. 섀도우 페이지 테이블의 구현은 게스트 운영체제의 페이지 테이블의 주소 매핑 내용을 어느 시점에 섀도우 페이지 테이블에 동기화 하는지가 성능 상 매우 중요하다. 게스트 페이지 테이블 엔트리의 갱신 시마다 이를 섀도우 페이지 테이블에 바로 반영하는 방법보다는 게스트 페이지 테이블이 수정되는 경우에만 이를 섀도우 페이지 테이블에 반영하는 방법이 좋은 성능을 보인다.

3. Fault tolerance by memory virtualization

흔히 임베디드 시스템은 유지 보수 없이 오랜 시간 동안 중단 없이 동작해야 하는 경우가 많기 때문에 앞서 언급한 고장 격리를 통한 가용성 향상이라는 가상화의 이점은 임베디드 시스템의 운용에 큰 도움이 될 수 있다.

[10]은 MMU 기능을 가진 마이크로프로세서를 지원하는 가상 머신 모니터에 추가적인 고장 감내 기법을 적용하여 임베디드 시스템에 높은 신뢰성과 가용성을 제공한 선행 연구이다. 이 기법에서는 우선 기본적인 CPU 가상화를 통해 임베디드 시스템을 가상 머신 상에서 수행하게 하고, 가상 머신마다 별도의 가상 머신을 생성하여 대상 임베디드 시스템의 동작을 주기적으로 백업하도록 하였다. 만일 동작 중인 대상 시스템에 오류가 발생하는 경우 해당 백업 가상 머신이 주 가상 머신의 역할을 대신하여 동작을 이어나가도록 하였다. 특히 고장 시 백업 시스템으로의 전환이 가상 머신 모니터가 관리하는 메모리 관리 자료구조 상의 조작을 통해 효율적으로 이루어지게 함으로써 백업 시스템의 운영으로 인한 성능 저하가 크지 않도록 하고 있다.

이 고장 감내 임베디드 시스템에서는 메모리 자료구조의 조작만을 통해 오류를 일으킨 가상 머신 상의 게스트 운영체제나 태스크를 쉽게 복구할 수 있다. 특히 게스트 운영체제에 오류가 발생하는 경우와 태스크에 오류가 발생하는 경우를 구분하여 별도의 방법으로 오류 복구를 수행한다. 다만 이 기법은 본격적인 메모리 가상화를 적용했다고 보기는 어렵다. 대신 일반적인 CPU 가상화만 이루어진 가상 머신 모니터를 기반으로 마이크로프로세서의 MMU를 고장 감내를 위한 중복에 활용하고 있을 뿐이다. 이에 따라 가정하는 게스트 운영체제도 가상 메모리를 사용하는 본격적인 운영체제를 게스트 운영체제로 사용하기 보다는 플랫한 메모리 구조를 가지는 MicroC/OS-II 운영체제를 대상으로 하는 등 제약을 가지고 있다.

III. The Proposed Scheme

이 장에서는 임베디드 시스템 용 가상 머신 모니터를 기반으로 메모리 가상화 기능을 활용한 고장 감내 시스템을 제안한다. 제안된 고장 감내 시스템의 전체 구조, 고장 감내를 고려한 섀도우 페이지 테이블의 조작, 오류 발생을 대비한 백업과 복구 등을 다룬다.

1. System architecture

제안하는 시스템에서는 우선 고장 감내 기능을 적용하고자 하는 대상(target) 임베디드 시스템을 가상 머신 모니터가 제공하는 하나의 가상 머신 상에서 동작시킨다. 또한 이 대상 시스템에 대한 중복 시스템을 별도의 백업 가상 머신 상에 구축하고, 시스템이 동작하는 과정에서 미리 정해진 시점마다 대상 시스템의 동작 상태를 백업 가상 머신에 저장해 두었다가, 대상 임베디드 시스템에 오류가 발생하면 백업 가상 머신에 저장해 두었던 정상 상태의 이미지를 대상 시스템으로 교체하여 동작을 이어나가는 방식이다.

Fig. 1은 본 논문에서 제안하는 가상화된 임베디드 시스템의 전체 구조이다. 그림에서 보는 바와 같이 대상 임베디드 시스템은 가상 머신 모니터 상에서 하나의 가상 머신으로 동작한다. 또한 이 가상 머신에 대한 중복을 가지는 백업 가상 머신이 쌍으로 생성된다. 백업 가상 머신은 대상 가상 머신에 대해 스탠바이 시스템의 역할을 하는 가상 머신으로서, 시스템이 정상적으로 동작하는 중에는 대상 가상 머신의 메모리 영역의 내용을 복제하는 역할만 수행하고, 실제 대상 가상 머신에 복구할 수 없는 오류가 발생하는 경우 이를 대신해서 대상 가상 머신으로 동작하게 된다. 따라서 평상시에 백업 가상 머신은 가상 머신 스케줄링의 대상에서 제외된다. 가상 머신 모니터는 메모리 가상화 모듈과 고장 감내 모듈을 가지고 있다. 메모리 가상화 모듈은 다음 절에서 설명하는 섀도우 페이지 테이블 기법을 통해 대상 가상 머신과 백업 가상 머신에게 각각 일정한 머신 주소 공간을 제공한다. 고장 감내 모듈은 정상 동작 중의 백업 가상 머신으로의 저장, 오류 발생의 탐지, 그리고 복구 과정에서의 가상 머신 간의 전환 등을 담당한다. 고장 감내 모듈은 효율적인 메모리 백업과 간편한 복원을 위해 독립적으로 동작하기 보다는 메모리 가상화 모듈과 밀접하게 연관되어 동작한다.

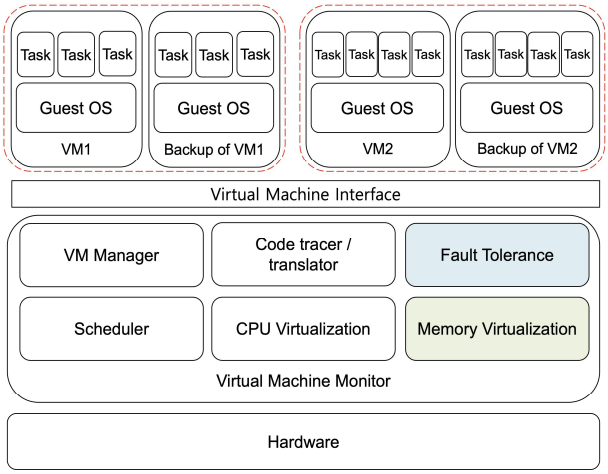

Fig. 1. System architecture

대상 시스템의 백업과 오류에 대한 탐지 및 복구는 전적으로 가상 머신 모니터의 고장 감내 모듈이 담당하고 있으므로 대상 임베디드 시스템의 운영체제나 그 태스크들은 가상 머신 모니터의 오류 탐지나 복구 메커니즘 여부과 관계없이 투명하게 동작한다.

2. Memory management for backup

제안하는 가상 머신 모니터는 섀도우 페이지 테이블 기법을 기반으로 하는 메모리 가상화를 지원한다. 기본적으로 제안된 시스템에서는 기존 연구들과 유사한 개념의 섀도우 페이지 테이블을 사용한다. 다만 게스트 운영체제의 페이지 테이블의 내용을 기반으로 섀도우 페이지 테이블을 만드는 과정에서, 게스트 페이지 테이블과 섀도우 페이지 테이블 간의 동기화 시점이나 방법에 따라 전체 시스템의 성능에 큰 차이가 발생하는 점을 고려하여, 게스트 운영체제의 페이지 테이블 내용을 섀도우 페이지 테이블에 동기화 하는 오버헤드를 크게 줄일 수 있는 방법을 사용하였다. 또한 통상적인 섀도우 페이지 테이블에 대한 조작 외에 가상 머신의 백업 이미지를 저장하기 위한 메모리 중

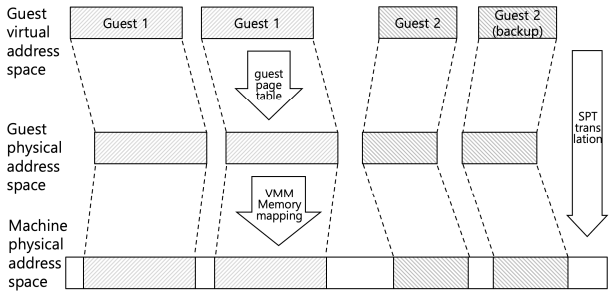

Fig. 2. Address mapping

복을 효과적으로 수행하고 오류 발생 시점에 중복된 내용을 활용한 복구 과정을 신속하게 수행하는 기능을 섀도우 페이지 테이블 조작 연산에 추가하였다.

본 논문에서 제안하는 시스템에서의 메모리 주소 변환 과정은 Fig. 2와 같다. 우선 대상 임베디드 시스템의 운영체제는 일반적인 가상 메모리 기능을 제공하는 것을 가정한다. 따라서 게스트 운영체제에는 수행되는 프로세스마다 페이지 테이블이 존재하며, 이 페이지 테이블은 프로세스의 가상주소(virtual address)를 게스트 물리주소(guest physical address)로 변환하는데 사용된다. 그러나 실제 임베디드 시스템은 가상 머신 상에서 수행되고 있으므로, 여기서의 게스트 물리 주소는 실제 물리 주소가 아닌 일종의 의사 물리 주소(pseudo physical address)라 할 수 있다. 이 의사 물리 주소는 가상 머신 모니터의 메모리 할당 기법의 적용을 받아 최종적으로 머신 물리 주소(machine physical address)로 변환된다. 가상 머신 모니터는 대상 가상 머신과 백업 가상 머신에게 머신주소 공간 상에서 물리적으로 연속된 일정 크기의 주소 공간을 할당하는 것으로 하였다. 할당된 머신 주소 공간의 시작 주소와 크기 등의 정보는 가상 머신 모니터에 의해 관리된다. 고장 감내 기능을 위한 메모리 중복으로 인해 최초의 대상 가상 머신과 백업 가상 머신의 머신 주소 공간의 메모리 내용은 완전히 동일한 상태에서 시작한다.

가상 머신 모니터는 가상주소를 머신주소로 변환하기 위해 Fig. 3과 같이 섀도우 페이지 테이블을 사용한다. 섀도우 페이지 테이블은 가상 머신 모니터의 자료구조로서 실제 CPU가 접근하는 가상주소에 대해 MMU가 이 페이지 테이블을 참조하여 주소 변환을 수행하게 된다. 가상 머신 모니터는 가상 머신 상에서 실행 중인 프로세스마다, 즉 게스트 운영체제의 페이지 테이블마다 가상 머신 모니터 내에 하나의 섀도우 페이지 테이블이 존재하며, 이 섀도우 페이지 테이블의 엔트리 내용은 게스트 페이지 테이블 상의 가상페이지->물리페이지 변환 정보에 해당하는 가상페이지->머신페이지 변환 정보이다. 섀도우 페이지 테이블

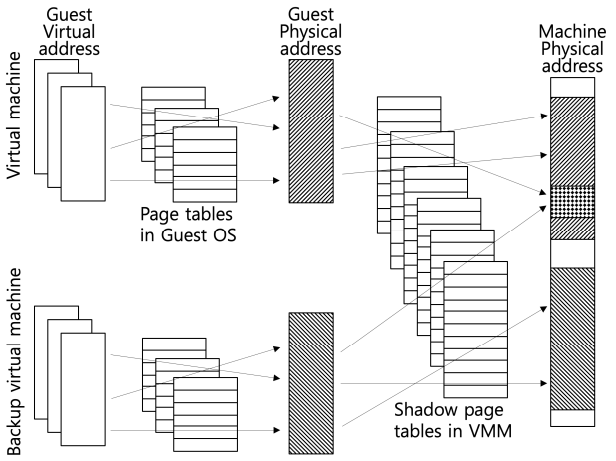

Fig. 3. Shadow page table

은 다양한 방식으로 설계할 수 있다. 한 가지 방식은 게스트 운영체제에서 새로운 페이지 테이블을 생성할 때 이 페이지 테이블의 모든 매핑 정보에 대하여 해당 섀도우 페이지 테이블을 구성한 후, 매번 게스트 페이지 테이블이 변경될 때마다 이를 동기적으로 섀도우 페이지 테이블에 반영해 나가는 것이다. 하지만 이러한 동기적인 갱신 방식은 빈번한 섀도우 페이지 테이블 엔트리의 수정과 그에 따른 문맥 전환으로 인해 심각한 시스템의 성능 저하를 가져오게 된다.

따라서 섀도우 페이지 테이블을 전체적으로 미리 구성하기 보다는 아직 섀도우 페이지 테이블 엔트리가 만들어지지 않은 페이지에 대한 접근이 발생하는 경우 즉석에서 해당 엔트리를 생성하는 방식을 선택하였다. MMU는 섀도우 페이지 테이블을 기준으로 가상 주소를 변환하므로 섀도우 페이지 테이블에 정의되어 있지 않은 주소 접근을 시도하면 Data Abort 예외가 발생하게 된다. 가상 머신 모니터의 예외 처리 함수는 이 예외가 발생한 가상 주소를 기반으로 게스트 페이지 테이블의 관련 가상주소->물리주소 매핑을 사용하여 섀도우 페이지 테이블에 해당 엔트리를 생성한 후 예외를 일으킨 명령을 다시 수행하게 된다.

다만 게스트 페이지 테이블을 접근하는 과정에서 게스트 페이지 테이블에 해당 주소 변환 정보가 존재하지 않는 경우도 있을 수 있다. 이 경우에는 가상 머신 모니터가 해당 게스트에게 가상의 Data Abort 예외를 발생시키고, 게스트 운영체제가 예외 처리 과정에서 해당 주소에 대한 가상주소->물리주소 매핑 엔트리를 생성하도록 후 예외를 발생시킨 명령을 다시 수행하게 한다. 이 과정에서 아직까지는 해당 섀도우 페이지 테이블 엔트리가 생성된 것은 아니므로 또다시 예외가 발생하게 되지만 게스트 운영체제에 관련된 페이지 테이블 엔트리가 이미 만들어진 상태이

므로 앞서 언급한 과정을 거쳐 섀도우 페이지 테이블 엔트리가 정상적으로 생성되고 주소 변환이 마무리될 수 있다. 특히 이와 같이 새로운 페이지에 대한 섀도우 페이지 테이블 엔트리에 추가되는 상황은 해당 페이지를 백업 가상 머신으로 백업할 수 있는 지점 중 하나가 된다.

가상 머신 모니터는 게스트 운영체제의 페이지 테이블의 내용이 수정되는 경우에도 그에 따라 섀도우 페이지 테이블을 갱신해야 한다. 게스트 운영체제는 자신의 페이지 테이블 내용을 수정하고 나면 보통 이를 TLB에 반영하기 위해 TLB 무효화(invalidate) 명령을 수행한다. 따라서 가상 머신 모니터가 이 TLB 무효화 명령을 가로채면 게스트 운영체제의 페이지 테이블 변경 사항을 섀도우 페이지 테이블에 반영할 수 있다. 본 논문에서 가정하고 있는 ARM 구조에서는 TLB 무효화 명령이 특권 명령이기 때문에 특권 모드에서 동작하지 않는 가상 머신이 이 명령을 수행하려고 하면 Undefined 예외가 발생하게 되어 가상 머신 모니터가 이 명령의 실행을 가로챌 수 있다. 가상 머신 모니터의 Undefined 예외 처리 함수에서는 무효화되는 게스트 페이지 테이블 엔트리에 해당하는 섀도우 페이지 테이블 엔트리를 제거함으로써 게스트 페이지 테이블의 수정 사항을 반영하게 된다. 이와 같이 섀도우 페이지 테이블 엔트리가 삭제되는 지점 역시 앞서 엔트리가 새로 만들어지는 지점과 함께 해당 페이지가 백업 가상 머신의 메모리 영역으로 복사되는 지점 중 하나이다.

게스트 운영체제의 페이지 테이블이 수정되는 또 다른 경우는 가상 머신에서 프로세스 간 문맥 전환이 이루어지는 경우이다. 프로세스 간 문맥 전환이 발생하면 새로운 프로세스가 자신의 주소 공간을 사용할 수 있도록 기존 프로세스의 TLB 엔트리를 모두 무효화해야 하기 때문에 가상 머신 모니터의 섀도우 페이지 테이블도 당연히 영향을 받게 된다. 다만 기존 섀도우 페이지 테이블 엔트리를 모두 삭제한 후 새 프로세스를 위한 섀도우 페이지 테이블을 새로 만드는 방식은 성능 저하가 우려되므로, 대신 게스트 운영체제의 각 프로세스마다 섀도우 페이지 테이블을 하나씩 할당해 놓고, 문맥 전환 시에는 단순히 새 프로세스의 섀도우 페이지 테이블로 전환함으로써 기존에 이미 만들어진 엔트리들을 그대로 활용하도록 하여 문맥 전환에 따른 성능 저하를 최소화하였다.

3. Backup and recovery

가상 머신 모니터는 동작 중 대상 가상 머신의 메모리 영역을 백업 가상 머신의 메모리 영역으로 백업을 한다. 이와 같이 시스템 운용 도중에 메모리를 백업하는 것은 시스템에 커다란 부하 증가를 초래할 것으로 예상된다. 이를 방지하기 위해서 메모리 백업 횟수를 최소화하고 백업의 대상이 되는 내용도 반드시 필요한 부분에 대해서만 백업이 이루어질 수 있도록 설계하였다.

우선 메모리 백업은 가상 주소 변환을 위해 사용하는 섀도우 페이지 테이블을 활용한다. 백업의 단위도 전체 메모리 영역에 대해 일괄적으로 복사하기 보다 페이지 단위로 백업을 수행한다. 또한 백업 시점도 앞서 언급한 바와 같이 섀도우 페이지 테이블의 엔트리가 생성되거나 변경되는 시점에 한해 해당 페이지에 대해서만 백업이 이루어질 수 있도록 하였다. 그리고 텍스트 영역에 해당하는 페이지와 같은 읽기 전용 페이지는 백업 대상에서 제외하고 데이터, 스택, 힙 영역에 해당하는 페이지만 백업 대상으로 하였다. 텍스트 페이지의 경우는 실질적으로는 대상 시스템과 백업 시스템이 공통의 페이지를 공유하고 있는 상태이기 때문에 앞 절의 그림 3에도 표시되어 있는 바와 같이 섀도우 페이지 테이블 수준에서 동일한 페이지 영역이 접근될 수 있도록 구성하였다. 또한 데이터 영역과 같이 백업의 대상이 되는 영역이라도 페이지 테이블 엔트리를 생성한 후 페이지 내용에 수정이 없는 페이지는 백업 대상에서 제외함으로써 전체적으로 메모리 백업에 따른 오버헤드를 최소화 하였다.

Fig. 4는 두 개의 가상 머신이 실행 중인 가상 머신 모니터 상에서 가상 머신들을 백업하고 오류 발생 시 이 백업을 이용해 복구가 이루어지는 시나리오를 보이고 있다. 그림 중 가상 머신 수행 부분 안에서 음영으로 표시되어 있는 것처럼 백업이 수행되는 시점은 가상 머신이 실행되는 도중 메모리 참조에 따라 새로운 섀도우 페이지 테이블 엔트리가 생성되거나, 관련 게스트 페이지 테이블에 새 엔트리가 추가되는 경우, 게스트 페이지 테이블 내용이 변경되는 경우, 게스트 프로세스 간 문맥 전환으로 인해 게스트 페이지 테이블이 교체되는 경우 등이다. 또한 가상 머신 모니터 상에서 가상 머신 간의 전환 시점에도 추가적인 백업이 이루어진다. 이는 그림에서 2, 5, 11에 해당한다. 그림의 7과 같이 동작 중인 가상 머신 VM1에 오류가 발생

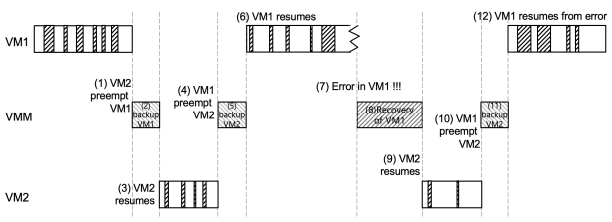

Fig. 4. Backup and recovery

하면, 가상 머신 모니터는 8에서 VM1에 대한 백업 가상 머신을 사용하여 복구를 수행하게 된다. 이 때 복구에 사용되는 백업 가상 머신은 오류 시점을 기준으로 가장 최근 실시된 백업 시점인 2에서 백업된 내용을 기반으로 하는 것이다. 이후 실행이 재개된 가상 머신 VM1은 12에서 정상적인 수행을 이어나가게 된다. 특히 대상 가상 머신과 백업 가상 머신 간에 섀도우 페이지 테이블의 교환으로 간단하게 복구가 이루어지도록 하였다.

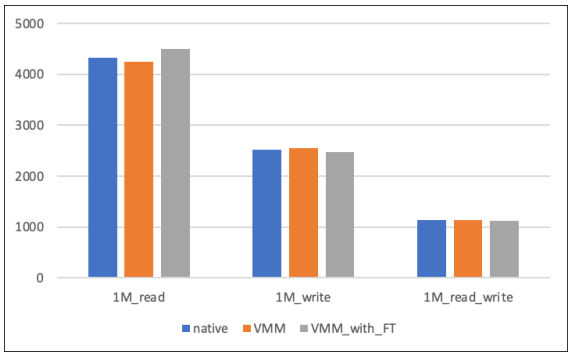

Fig. 5. Comparison on memory bandwidth

IV. Performance Evaluation

이 장에서는 제안한 시스템의 오류 복구 기능이 전체 시스템의 성능에 어떤 영향을 미치는지 파악하기 위해 성능 평가를 실시한 결과를 제시한다. 통상 전가상화 방식으로 컴퓨터 시스템을 가상화하면 가상화하지 않은 네이티브 시스템에 비해 성능 저하가 있을 수밖에 없다. 게다가 제안한 시스템의 경우 오류에 대비한 백업 기능까지 추가되었기에 일정 수준의 성능 저하가 예상된다. 다만 이러한 성능 저하가 오류 복구 기능을 얻는 것에 견주어 감당할만한 수준인지 가늠해 보는 것을 성능 평가의 목표로 하였다. 이를 위해 널리 사용되는 벤치마크를 사용하여 가상화하지 않은 네이티브 시스템, 가상화된 임베디드 시스템, 그리고 오류 복구 기능이 적용된 가상화된 임베디드 시스템의 성능을 각각 측정하여 오류 복구 기능이 전체 시스템의 성능에 어떤 영향이 미치는지 파악하고자 하였다. 제안한 가상 머신 모니터와 고장 감내 기능은 AM37x 1GHz ARM 구조의 Cortex-A8 프로세서가 탑재된 BeagleBoard-xM [11]에서 구현하였다. 게스트 운영체제로는 리눅스 4.9를 사용하였고, 성능 측정을 위해 널리 사용되는 마이크로 벤치마크 프로그램인 lmbench 3.0-a4 버전을 사용하였다.

먼저 lmbench의 메모리 대역폭 측정 벤치마크 bw_mem을 사용해 가상 머신 모니터의 성능을 측정하였다. bw_mem은 일정 크기의 메모리를 두 번 할당하여 이들 간에 1초 동안 복사한 데이터의 양을 측정하여 메모리 대역폭을 측정한다. Fig. 5는 각각 1MB 크기의 데이터를 각각 읽는 경우, 1MB를 메모리에 쓰는 경우, 1MB를 읽고 쓰는 경우에 대해, 가상화하지 않은 네이티브 머신, 가상화된 임베디드 시스템, 고장 감내 기능을 적용한 가상 머신의 성능 측정 결과를 비교한 것이다. 그림에서 보는 바와 같이 세 시스템 간에 메모리 대역폭에는 큰 차이가 없는 것으로 나타나고 있다.

다음은 lmbench가 제공하는 몇 가지 지연 시간(latency) 측정 벤치마크들을 이용해 성능 측정을 수행하였다. 우선 시스템 콜 지연 시간을 측정하는 lat_syscall 벤치마크를 이용해 프로세스 ID를 구하는 getpid 시스템 콜, /dev/zero로부터 한 바이트를 읽는 read 시스템 콜, /dev/null에 한 바이트를 쓰는 write 시스템 콜에 대해 측정하였다. Fig. 6은 가상화하지 않은 네이티브 머신, 단순히 가상화된 임베디드 시스템, 그리고 고장 감내 기능을 포함한 가상화된 임베디드 시스템에서의 이 시스템 콜들의 지연 시간을 비교한 것이다. 고장 감내 기능이 없는 가상 머신 모니터의 경우 가상화하지 않은 시스템에 비해 약 2배 정도의 성능 차이가 나며, 고장 감내 기능이 포함되는 경우 약 7배까지의 성능 차이가 나는 것을 알 수 있다.

프로세스 생성 지연 시간을 측정하는 lat_proc 벤치마크를 사용하여 fork로 프로세스를 생성하는 경우와 fork 이후 exec을 통해 새로운 프로그램을 실행하는 경우의 지연 시간을 측정하였다. Fig. 7은 가상화하지 않은 네이티브 머신, 단순히 가상화된 임베디드 시스템, 그리고 고장 감내 기능을 포함한 가상화된 임베디드 시스템의 프로세스 생성 시간을 비교한 것이다. 그림에서 보는 바와 같이 고장 감내 기능이 적용된 가상 머신 모니터는 가상화하지 않은 네이티브 시스템에 비해 평균 8배 정도의 성능 차이

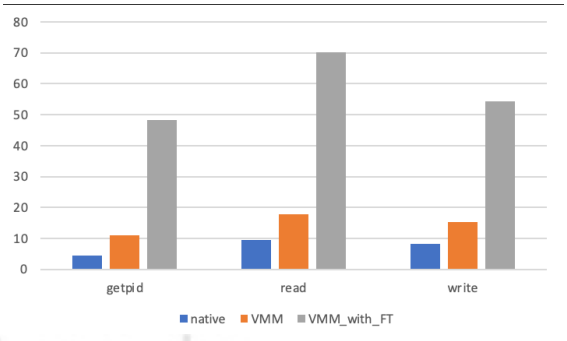

Fig. 6. Comparison on system call latency

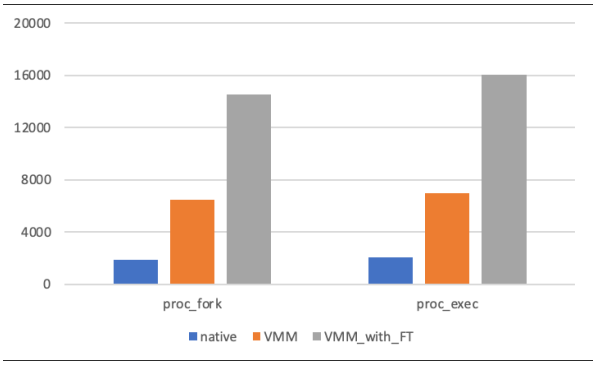

Fig. 7. Comparison on process creation latency

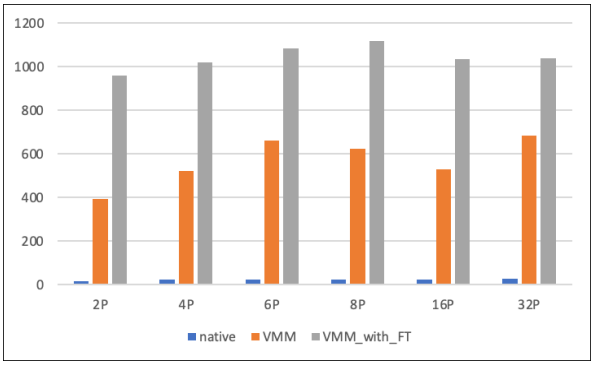

Fig. 8. Comparison on context switch latency

를 보이고 있다. 또한 고장 감내 기능이 배제된 가상화 시스템과도 약 3배 정도의 차이가 나는 것을 알 수 있다.

마지막으로 lat_ctx 벤치마크를 사용하여 문맥 전환에 걸리는 시간을 측정하였다. 이를 위해 64KB 크기의 프로세스를 각각 2개, 4개, 6개, 8개, 16개, 32개 실행하는 경우에 대해 각각 문맥 전환 지연 시간을 측정하였다. Fig. 8은 각 경우마다 가상화하지 않은 네이티브 머신, 단순히 가상화된 임베디드 시스템, 그리고 고장 감내 기능을 포함한 가상화된 임베디드 시스템에서의 문맥 전환에 걸리는 시간을 비교한 것이다. 현재는 가상화하지 않은 시스템과는 약 45배 정도의 차이가 있으며, 고장 감내 기능이 배제된 가상머신과도 약 1.5~2배 정도의 성능 차이가 나는 것을 알 수 있다. 이 부분은 앞으로 개선해야 할 여지가 많다.

이러한 성능 저하는 대부분 섀도우 페이지 테이블에 대한 갱신과 특히 해당 시점에서의 페이지 백업에 따른 지연 시간으로 추정된다. 대부분의 측정 항목들에서 적지 않은 성능 저하가 나타났다 그러나 앞서 언급한 바와 같이 임베디드 시스템과 같이 오랜 시간 동안 오류 없이 동작해야 하는 환경에서는 일정 수준의 성능 저하를 감수하더라도 오류 복구 기능의 적용으로 얻게 되는 가용성 증대가 더 중요한 시스템 요구사항이 될 수 있다. 특히 가상 머신마다 백업에 걸리는 시간의 상한을 일정하게 유지할 수 있다면 항상 동일한 가상 머신과 프로세스만 실행되는 임베디드 시스템의 특성 상 큰 문제가 없으리라 여겨진다.

V. Conclusions

가상화 기술을 적용하면 임베디드 시스템의 가용성을 크게 높일 수 있다. 본 논문에서는 섀도우 페이지 테이블 기법을 기반으로 하는 메모리 가상화를 활용한 고장 감내 기법을 사용하여 높은 신뢰성을 가진 임베디드 시스템을 제안하였다. 제안된 고장 감내 임베디드 시스템은 효율적으로 섀도우 페이지 테이블을 관리할 뿐만 아니라, 섀도우 페이지 테이블 조작 과정에서 메모리 영역을 백업하여 오류 발생 시 효과적으로 오류를 복구하여 정상적인 동작을 가능하게 한다. 성능 평가를 통해 제안된 기법이 임베디드 시스템의 성능을 크게 저하시키지 않으면서도 시스템의 가용성을 크게 증가시킬 수 있음을 보였다.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.
[8] 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.
[9] M. Rosenblum and T. Garfinkel, “Virtual Machine Monitors: Current Technology and Future Trends,” IEEE Computer, May 2005.
[10] Sunghoon Son, "Error Recovery Technique for Improving Reliability of Embedded Systems," Journal of The Korea Society of Computer and Information, Vol. 22, No. 6, pp. 1-8, June 2017.
[11] BeagleBoard-xM, http://www.beagleboard.org/beagleboard-xm

Authors

Sunghoon Son received his B.S., M.S. and Ph.D. degrees in Computer Science from Seoul National University, Seoul, 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.