韓國컴퓨터情報學會 論文誌
第15卷 第2號, 2010. 2.

2010-15-2-1-3

윈도우 운영체제에서 레지스트리 가상화 구현

신 동 하*

Implementation of Registry Virtualization on Windows

Dongha Shin*

요 약

윈도우 운영체제의 레지스트리는 시스템 및 응용 프로그램의 설정 데이터가 저장되는 계층적 구조를 가지는 데이터베이스이다. 본 논문에서는 윈도우 운영체제에서 레지스트리 가상화 알고리즘을 제안하고 구현하였으며 그 성능을 측정하였다. 본 논문에서 제안하는 레지스트리 가상화 알고리즘은 Copy-One-level On Write-Open(COOWO)이라고 불리는데 이는 일반적인 Copy On Write(COW) 방식을 레지스트리 가상화에 적합하도록 수정한 것이다. 본 논문에서는 제안한 알고리즘을 윈도우 운영체제에서 동적 라이브러리로 구현하였고 다양한 윈도우 응용 프로그램에 적용하였다. 본 논문은 레지스트리 가상화를 구체적으로 다루는 논문이 많지 않은 현실에서 레지스트리 가상화 알고리즘을 상세하게 기술하였고, 알고리즘의 성능이 실제 응용에 사용 가능하다는 것을 발견하였다는 점에서 의의가 있다.

Abstract

The Windows registry is a hierarchical database where the configuration data of a system or application programs is stored. In this paper, we presented and implemented a registry virtualization algorithm, and measured its performance. The registry virtualization algorithm presented in the paper is called Copy-One-level On Write-Open(COOWO) that is a modified version of general Copy On Write(COW) method to make it suitable for registry virtualization. In this paper, we implemented the proposed algorithm as a dynamically loadable library in Windows and applied it to many Windows application programs. This paper is meaningful since we described a registry virtualization algorithm in detail in situation where we can not find papers that describe the registry virtualization in detail, and we could find the performance of the algorithm can be used in the real applications.

▸ Keyword : 레지스트리(registry), 가상화(virtualization), 윈도우 운영체제(Windows Operating System), 프로세스 가상화(process virtualization)

• 제1저자 : 신동하
• 투고일 : 2010. 01. 22, 심사일 : 2010. 02. 01, 게재확정일 : 2010. 02. 22.
* 상명대학교 컴퓨터과학부 교수
※ 본 논문은 2008년 상명대학교 교내연구비에 의하여 지원되었습니다.

Ⅰ. 서 론

최근 들어 가상화(virtualization) 기술이 정보통신 분야에서 많은 관심을 모으고 있다. 기술 예측 기관인 가트너 그룹은 2008년 발표에서 가상화 기술이 2012년까지 정보통신 인프라 부분에서 가장 큰 영향력을 가지는 기술이라고 강조하였다[1]. 가상화 기술이란 컴퓨터의 자원을 적절한 추상 단계(예: API, ABI, ISA 등)에서 제어하고 관리하여 가상의 컴퓨팅 환경을 제공하는 기술이다[2]. 가상화 기술은 매우 폭 넓은 활용 분야를 제공한다. 가상화 기술은 기업에서 사용하는 물리적 서버의 수를 줄일 수 있고, IT 인프라 구축에 걸리는 시간을 줄일 수 있으며, 개발자에게는 효율적이고 융통성 있는 개발 환경을 제공하고, 오래 전에 개발된 구형 소프트웨어를 지속적으로 수행시킬 수 있는 환경을 제공할 수도 있으며, 보안에 적합한 컴퓨팅 환경을 제공할 수도 있다[3]. 가상화 기술은 최근 내장형 시스템 분야에 까지 적용되면서 활용 분야를 넓히고 있다[16][17].

가상화는 적용하는 컴퓨터 구조의 추상 단계(abstraction level)에 따라 '프로세스 가상화'와 '시스템 가상화'로 나누어진다[2][12]. 프로세스 가상화는 API(Application Programming Interface) 단계 혹은 ABI(Application Binary Interface) 단계 아래에서 가상 수행 환경을 제공하는 가상화이고, 시스템 가상화는 ISA(Instruction Set Architecture) 단계 아래에서 가상 수행 환경을 제공하는 가상화이다. 즉 프로세스 가상화는 응용 프로그램의 수행을 가상화하는 방법이고 시스템 가상화는 운영체제의 수행을 가상화하는 방법이다. 본 논문에서는 프로세스 가상화 기술을 다룬다. 프로세스 가상화 기술을 사용하면 기업에서 사용하는 많은 데스크 탑에 설치된 응용 프로그램들을 손쉽게 중앙에서 관리할 수 있고, 응용 프로그램을 컴퓨터에 설치하지 않고 수행시킬 수 있으며, 응용 프로그램의 설치, 설정, 배포 및 관리가 용이해지며, 응용 프로그램에게 높은 보안을 제공할 수도 있으며, 여러 응용 프로그램 사이에 발생하는 충돌도 방지할 수 있다[4].

본 논문은 프로세스 가상화 기술 중 윈도우 운영체제 상에서 적용되는 레지스트리(registry) 가상화 기술에 대하여 다룬다. 레지스트리는 윈도우 운영체제 상에서 시스템 혹은 응용 프로그램의 설정 데이터가 저장되는 계층적 구조를 가지는 데이터베이스이다[13]. 레지스트리 가상화란 윈도우의 호스트 프로세스가 사용하는 레지스트리 자원과는 분리된(isolated) 레지스트리 자원을 ABI 단계에서 가상 프로세스에게 제공하는 기술이다. 가상 레지스트리는 ABI 단계에서 제공되기 때문에 그 상위 계층에서 수행되는 응용 프로그램은 호스트 레지스트리를 사용하면서 수행하고 있다고 생각하지만 실제로는 가상 레지스트리를 사용하면서 수행하게 된다. 윈도우 운영체제의 초기 레지스트리는 매우 크기 때문에 공간 효율성을 위하여 가상 프로세스가 사용하는 레지스트리는 초기에 비어(empty) 있으며 호스트 레지스트리를 읽기 공유(read share)한다. 가상 프로세스가 수행하면서 레지스트리에 데이터를 쓰면(write) 그 때 호스트 레지스트리의 해당 부분을 가상 레지스트리로 복사하여 가상 레지스트리를 서서히 구성한다. 이와 같은 방식으로 자원을 관리하는 방식을 일반적으로 Copy On Write(COW) 방식[5][6]이라고 한다. 본 논문에서는 이 방식을 수정한 Copy-One-level On Write-Open(COOWO)이라는 알고리즘을 제안하고 구현한다. 본 논문에서 제안하는 방식은 호스트 레지스트리와 가상 레지스트리의 유니온(union) 동작이 필요하지 않으며, ABI 단계에서 쉽게 구현이 가능하다는 장점이 있다. 구현된 레지스트리 가상화 모듈은 동적 라이브러리 형태로 구현되어 기존의 윈도우 응용 프로그램에 주입되면(inject) 수행 시 가상 프로세스로 동작되어 가상 레지스트리를 사용하게 된다. 구현된 레지스트리 가상화 모듈은 레지스트리 입출력 동작에 대하여 약 26.24%의 수행 시간 증가를 가지는 것으로 측정되었다. 일반적으로 응용 프로그램은 수행 시간 중 극히 일부 시간을 레지스트리 동작에 할애하기 때문에 전체 응용 프로그램의 수행에는 성능 저하를 느끼기 힘들 것으로 판단된다. 예를 들어 어떤 응용 프로그램의 수행 시간 중 약 5%를 레지스트리 동작에 할애한다면(주: 이 수치는 매우 큰 수치임) 전체 프로세스 입장에서는 5%*26.24%=1.3% 정도의 수행 시간 증가를 가져온다. 본 논문은 레지스트리 가상화 구현에 대한 구체적인 소개가 많지 않은 현실에서 레지스트리 가상화에 대한 설계 및 구현의 실제를 구체적으로 소개하였고 성능 측정 결과 충분히 실제 응용에 사용 가능한 성능을 가진다는 것을 확인하였다는 점에서 의의가 있다.

본 논문의 2장에서는 레지스트리 가상화에 대한 기본 개념을 소개한다. 여기서 윈도우 레지스트리의 개념, 레지스트리 관련 시스템 호출 함수, 레지스트리 가상화의 기본 개념에 대하여 설명한다. 3장에서는 레지스트리 가상화에 대한 기본 설계에 대하여 기술한다. 여기에서 본 논문에서 제안하는 COOWO 알고리즘의 동작 원리를 설명하고 이를 시스템 호출 함수에 적용하는 방법을 기술한다. 4장에서는 레지스트리 가상화의 구현에 관하여 기술한다. 이 장에서 레지스트리 가상화 라이브러리를 응용 프로그램에 주입하여 가상 프로세스를 생성하는 방법을 설명한다. 5장에서는 구현된 레지스트리 가상화 라이브러리를 윈도우 레지스트리 유틸리티인 'REG'에

적용하여 가상화에 따른 수행 시간 증가에 대한 성능을 평가한다. 마지막으로 6장에서는 본 논문의 결과 및 의미에 대하서 기술하고 향후 연구 내용에 관하여 기술한다.

II. 레지스트리 가상화 개념

2.1. 윈도우 레지스트리

레지스트리는 윈도우 운영체제에서 시스템 및 응용 프로그램의 설정 데이터가 저장되는 계층적 구조를 가지는 데이터베이스이다. 레지스트리는 레지스트리 키(key)와 레지스트리 값(value)으로 구성된다. 레지스트리 키는 파일 시스템의 폴더와 유사하여 하나의 레지스트리 키는 다수의 레지스트리 키 혹은 레지스트리 값을 가질 수 있다. 레지스트리 값은 파일 시스템의 파일의 개념과 유사하여 하나의 값은 다양한 형식(예: 스트링, 32비트 정수, 이진 데이터, 스트링의 배열 등)의 데이터를 가진다. 레지스트리 키 및 레지스트리 값은 스트링 형의 이름을 가진다. 그림 1은 레지스트리의 개념을 설명하기 위하서 인위적으로 구성한 레지스트리 보기이다. 이 그림에서 원으로 표현된 부분은 레지스트리 키를 사각형으로 표현된 부분은 레지스트리 값을 표현한다.

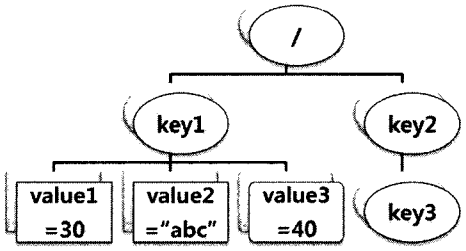

그림 1 레지스트리 보기
Fig. 1. Registry Example

그림 1에서 레지스트리 키 key1은 레지스트리 값 value1, value2 및 value3을 가지고 있고 레지스트리 키 key2는 레지스트리 키 key3을 서브키(subkey)로 가지고 있다. 레지스트리 값 value1, value2 및 value3에 저장된 데이터는 각각 30, "abc" 및 40이다. 이 그림에서 레지스트리 키 key3의 절대 경로명은 /key2/key3으로 표현된다.

2.2. 레지스트리 시스템 호출 함수

윈도우 운영체제는 응용 프로그램 작성을 위하여 다양한 종류의 Win API[14]를 제공한다. 32비트 윈도우 NT 계열의 운영체제의 경우 kernel32.dll, advapi32.dll, user32.dll, gdi32.dll 등이 Win API를 제공하는 동적 라이브러리이다. 이들 라이브러리 중 레지스트리 처리 Win API 함수는 advapi32.dll에 포함되어 있다. Win API는 하위 계층에 Native API[15]를 가지고 있다. Native API는 동적 라이브러리 ntdll.dll에 저장되어 있는데 이는 최종적으로 하위 계층에 있는 윈도우 커널을 불러서 구현되어 있다. Native API 계층에 있는 함수를 시스템 호출 함수라고 부른다. 윈도우 운영체제는 Native API 단계에서 약 30개의 레지스트리 처리 함수를 제공하고 있다. 이들 함수 중 레지스트리 가상화 구현과 관련된 주요 함수는 표 1과 같다.

표 1. 주요 레지스트리 처리 Native API 함수
Table 1. Main Registry Processing Native API Functions

함수 이름	함수 설명
ZwCreateKey	주어진 이름의 레지스트리 키를 생성하거나 오픈하고 핸들을 넘겨준다.
ZwOpenKey	주어진 이름의 레지스트리 키를 오픈하고 핸들을 넘겨준다.
ZwDeleteKey	핸들로 주어진 레지스트리 키를 제거한다.
ZwQueryKey	주어진 이름의 레지스트리 키에 대한 정보를 가져온다.
ZwEnumerateKey	주어진 이름의 레지스트리 키에 포함된 서브키 이름을 나열한다.
ZwSetValueKey	주어진 이름의 레지스트리 키에 레지스트리 값을 추가하거나 설정한다.
ZwQueryValueKey	주어진 이름의 레지스트리 값에 저장된 데이터를 가져온다.

2.3. 레지스트리 가상화 개념

윈도우 운영체제는 하나의 큰 호스트 레지스트리를 가지고 있다. 이 레지스트리는 파일 시스템과 마찬가지로 응용 프로그램이 수행되면서 레지스트리 키 경로명으로 접근하여 입출력을 수행한다. 레지스트리 가상화는 가상 프로세스에게 호스트 레지스트리와는 별개의 가상 레지스트리를 사용할 수 있게 제공한다. 이를 위하여 호스트 레지스트리의 특정 레지스트리 키 아래에 마운트되는 가상 레지스트리를 관리한다. 가상 레지스트리는 초기에는 비어있지만 가상 프로세스가 수행되면서 레지스트리에 데이터를 출력하면 크기가 증가한다. 그림 2는 호스트 레지스트리에 마운트된 가상 레지스트리의 보기를 나타낸 그림이다. 이 그림에서 회색으로 표시된 부분이 가상 레지스트리이다. 가상 레지스트리는 호스트 레지스트리에 마운트되어 있지만 호스트 프로세스가 접근할 수는 없게 설정되어 있다고 가정한다.

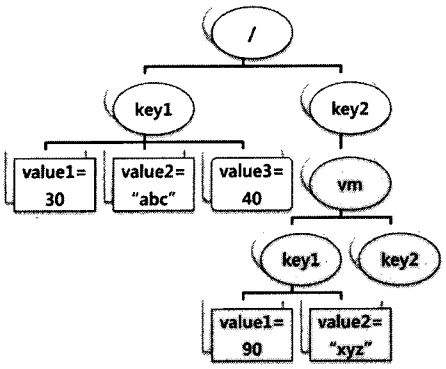

그림 2 호스트 레지스트리에 마운트된 가상 레지스트리
Fig. 2 Virtual Registry mounted at Host Registry

가상화가 이루어진 운영체제에서 모든 프로세스는 호스트 프로세스와 가상 프로세스로 나누어진다. 호스트 프로세스는 호스트 자원을 사용하고 가상 프로세스는 가상 자원을 사용한다. 예를 들어 그림 2의 상황에서 호스트 프로세스가 레지스트리 값 /key1/value1을 읽으면 30이 넘겨지고 가상 프로세스가 레지스트리 값 /key1/value1을 읽으면 90이 넘겨진다. 앞에서 설명한 바와 같이 초기에는 가상 레지스트리의 사용 공간을 절약하기 위하여 가상 레지스트리가 비어있고 호스트 레지스트리를 읽기 공유(read share)한다. 예를 들어 그림 2에서 가상 프로세스가 레지스트리 값 /key1/value3을 읽으면 이 값이 가상 레지스트리에 존재하지 않기 때문에 호스트 레지스트리에서 읽어 와서 40을 넘겨준다. 하지만 가상 프로세스가 레지스트리 값 /key1/value3의 데이터를 40에서 100으로 수정하면 레지스트리 가상화 모듈이 가상 레지스트리에 /key1/value3 (주: 실제 절대 경로명은 /key2/vm/key1/value3임)을 생성하고 데이터 값으로 100을 저장한다. 이와 같이 가상 레지스트리를 구성하는 방법을 COW라고 한다. COW는 앞의 보기처럼 가상 자원을 효율적으로 생성하기 위하여 사용될 수도 있고, 자원의 접근 히스토리를 기억하여 자원의 모습을 이전 상태로 되돌리기 위한 스냅샷(snapshot) 기능을 제공하기 위하여 사용되기도 한다[6]. 본 논문에서는 COW 방식을 레지스트리 동작에 적합하게 수정한 Copy-One-Level On Write-Open 방식을 제안한다.

III. 레지스트리 가상화 설계

3.1. Copy On Write-Open(COWO)

앞 장에서 소개한 COW는 가상 프로세스가 호스트 레지스트리에는 존재하지만 가상 레지스트리에는 존재하지 않는 레지스트리 키 및 값의 경로명에 대하여 쓰기 동작을 할 경우 호스트 레지스트리 키 혹은 레지스트리 값을 가상 레지스트리의 같은 경로명으로 복사한 후 쓰기 동작을 수행하는 방식이다. 이 방식은 호스트 공간에서 가상공간으로 자원을 복사하는 시간이 쓰기 동작이 일어날 때이다. 이를 구현하기 위해서는 커널 모드에서의 가상화 수정이 필요하기 때문에 사용자 모드에서 수행되는 라이브러리에서의 구현은 매우 힘들고 실제 프로그래밍도 복잡하다. 이를 사용자 모드의 라이브러리에서 해결하기 위하여 Copy On Write-Open(COWO) 방식을 도입한다. 이 방식은 쓰기 동작이 있을 때 자원의 복사가 일어나는 것이 아니라 쓰기 열기 동작이 있을 때 자원의 복사가 일어나는 방식이다. 이 방식은 쓰기 열기 후 실제 쓰기가 일어나지 않는 경우도 가끔씩 있기 때문에 비효율적이라고 판단될 수도 있지만 레지스트리의 경우 파일과는 달리 하나의 키 혹은 값이 매우 작은 데이터를 가지고 있기 때문에 시간 혹은 공간 효율 면에서는 큰 문제가 되지는 않는다. 이 방식은 구현이 간단하면서 라이브러리 단계에서 구현 가능하기 때문에 프로세스 가상 기계를 구현하였을 때 가상 기계의 이동성(portability) 향상에도 크게 이바지한다. 예를 들어 커널 드라이버로 구현된 가상 기계의 경우 윈도우 운영체제에서 수행되기 위해서는 관리자 권한이 필요한데 이 때문에 이동성을 보장하는 가상 기계 구현에는 부적합하다. VMware 사의 가상화 제품인 ThinApp의 경우 본 논문의 방식과 같이 라이브러리 방식으로 프로세스 가상 기계를 구현하여 이동성을 보장하였다는 점을 큰 장점으로 홍보하고 있다[7]. 표 2는 COW와 COWO의 특징 비교를 표로 나타낸 것이다.

표 2 COW와 COWO 비교
Table 2 Comparison of COW and COWO

방식	특징
COW	● 쓰기 열기 후 쓰기가 일어나지 않는 경우를 점검할 수 있어야 함 ● 커널 모드에서의 구현이 필요하여 이동성 있는 가상 기계 구현에는 부적합 ● 시간 및 공간 효율 면에서는 다소 우수
COWO	● 쓰기 열기 후 쓰기가 일어나지 않더라도 쓰기 열기 시 복사 동작이 수행됨 ● 라이브러리에서의 구현이 가능하여 이동성 있는 가상 기계 구현에 적합 ● 구현이 비교적 간단하여 시스템이 안정적임 ● 레지스트리는 데이터의 크기가 작아서 시간 및 공간 효율 면에서 큰 문제가 되지 않음

3.2. Copy-One-level On Write-Open(COOWO)

본 논문에서는 COWO를 레지스트리 가상화에 적합하게 변형한 COOWO 방식을 제안한다. COWO 방식을 그대로 레지스트리 가상화에 적용할 수도 있지만 아래와 같은 문제가 발생한다.

- 어떤 레지스트리 키 혹은 레지스트리 값이 호스트 레지스트리와 가상 레지스트리에 동시에 존재할 경우 이 레지스트리 키에 포함된 서브키 혹은 레지스트리 값을 모두 리스트하려고 할 때(예: 함수 ZwEnumerateKey의 호출의 경우) 호스트 레지스트리에 포함된 모든 서브키와 가상 레지스트리에 포함된 모든 서브키를 나열하여 합친(union) 후에 양쪽에서 중복되게 나열된 서브키는 호스트 레지스트리에 있는 서브키를 제거하여야(subtract) 한다. 이 동작은 자주 일어나는데 수행 비용이 크다.
- 어떤 레지스트리 키에 대하여 쓰기 동작이 일어날 경우 자식 레벨에 있는 서브키에 대하여도 자주 쓰기 동작이 일어나는 경우가 많다. 이는 키 쓰기 동작에 대한 로칼리티(locality) 특성이 있다는 점을 가정한 것이다.

이러한 문제를 효율적으로 해결하기 위하여 레지스트리 가상화에서는 가상 프로세스가 호스트에 존재하는 레지스트리 키에 대하여 쓰기 동작이 일어나면 그 키에 포함되는 모든 서브키 및 모든 레지스트리 값에 대하여(주: 한 단계 자식에만 해당됨) 동시에 복사 작업을 수행하면 효율적이다. 이 방식이 Copy-One-level On Write-Open(COOWO) 방식이다. 이 방식으로 복사된 키는 호스트 레지스트리 키와 가상 레지스트리 키가 한 단계 서브키까지 동일하여 완전하다고(complete) 이야기하고 itag(주: 불완전한 키를 표시함)이 존재하지 않는다. 보통의 레지스트리 키 및 값에는 매우 작은 양의 데이터가 들어 있기 때문에 한 단계 서브키 및 값의 복사에 드는 비용은 크지 않다. 이 문제는 본 논문에서 구현되기 전에 연구[8][9]에서 아이디어가 논의되었지만 구체적인 알고리즘은 발표되지 않았는데 본 연구에서 알고리즘을 구체적으로 설계하고 구현하게 되었다.

3.3. 제거 로그 처리 문제

가상 프로세스가 호스트 레지스트리에는 존재하지만 가상 레지스트리에는 존재하지 않는 키를 제거하는 경우 호스트 레지스트리의 키에 대하여 제거를 할 수 없기 때문에 이 동작을 제거 로그(delete log)에 기록한다. 제거 로그에 기록된 레지스트리 키는 나중에 열기 동작이 있을 때 호스트 레지스트리에 존재하지만 존재하지 않는다고 가정한다. 제거 로그는 보통 해쉬 테이블로 구현한다.

3.4. 상세 알고리즘 설계

본 절에서는 앞의 2.2에서 소개한 레지스트리 처리 Native API를 가상화하는 상세 알고리즘을 기술한다. 특히 함수 ZwCreateKey, ZwOpenKey, ZwDeleteKey의 가상화 동작에 대하여 중점적으로 기술한다. 각 함수의 가상화 동작은 몇 가지 기본 동작의 조합으로 이루어지는데 이들 기본 동작을 먼저 정의한다. 아래는 기본 동작 정의이다.

- mkdir V: 가상 레지스트리에 레지스트리 키 V를 생성한다. 이 때 생성된 키는 불완전하기 때문에 itag을 가진다.
- mkdir HVP: 호스트 레지스트리에 주어진 레지스트리 키에 대한 부모 키가 존재하면 가상 레지스트리에도 부모 키를 생성한다. 이 과정에서 생성되는 모든 키는 불완전하기 때문에 itag을 가진다.
- copy 1L: 주어진 레지스트리 키에 대하여 호스트 레지스트리에서 가상 레지스트리로 이 키의 모든 서브키 및 레지스트리 값을 복사한다. (주: copy-one-level 동작)
- logH+: 호스트 레지스트리에 해당 키가 존재하면 이 키 이름을 제거 로그(delete log)에 기록한다.
- log-: 주어진 레지스트리 키 이름을 제거 로그에서 제거한다.
- logfix: 제거 로그의 불일치를 해결하기 위하여 log-를 수행한다. (주: 이 경우는 호스트에 키가 없는데 제거 로그에 이 키가 포함되어 있는 경우로 실제 일어나지 않지만 만약을 위하여 만든 동작임)
- H: 호스트 레지스트리 경로명으로 함수를 수행한다.
- V: 가상 레지스트리 경로명으로 함수를 수행한다.
- A; B: A 동작 수행 후 B 동작을 수행한다.
- A & B: A 동작 수행 후 결과가 성공이면 B 동작을 수행한다.
- error: 오류 코드를 복귀한다.

표 3 및 표 4는 함수 ZwCreateKey/ZwOpenKey 및 ZwDeleteKey에 대한 가상화 함수 알고리즘을 조건표로 나타낸 것이다. 본 논문에서는 알고리즘을 절차적으로 표현하지 않고 주어진 함수의 인수에서 발생하는 모든 조건을 표로 표현하여 보다 정확하게 알고리즘을 표현하였다. 이 표에서 회색 부분은 조건을 나타내고 흰색 부분은 실제 동작을 나타낸다.

표 3. 함수 ZwCreateKey 및 ZwOpenKey 가상화 알고리즘
Table. 3. Virtualization Algorithms for ZwCreateKey and ZwOpenKey

delete log?	key exist?		VM_ZwCreateKey/VM_ZwOpenKey	
	H	V	write=0	write=1
0	0	0	V	mkdir HVP; V
1			logfix	logfix
0	0	itag	error	error
1			error	error
0	0	1	V	V
1			logfix	logfix
0	1	0	H	mkdir V; copy 1L; V
1			V	mkdir HVP; (V & log-)
0	1	itag	H	copy 1L; V
1			error	error
0	1	1	V	V
1			V	V

표 4. 함수 ZwDeleteKey 가상화 알고리즘
Table. 4. Virtualization Algorithm for ZwDeleteKey

key handle	VM_ZwDeleteKey
H	error
V	V & logH+

표 3 및 표 4의 알고리즘에 표현된 여러 조건의 의미는 아래와 같다.

- delete log?: 인수로 주어진 경로명이 제거 로그에 포함되어 있으면 1 그렇지 않으면 0임. (주: ZwCreateKey 및 ZwOpenKey의 인수는 경로명 및 읽기/쓰기 모드임)
- key exist?: 인수로 주어진 경로명에 해당되는 레지스트리 키가 호스트 레지스트리에 존재하면 H=1임. 가상 레지스트리에 존재하는 경우 itag이 없으면 V=1이고 있으면 V=itag임.
- write: 함수의 인수가 쓰기 열기이면 1이고 그렇지 않으면 0임.
- key handle: 인수로 주어지는 핸들이 호스트 레지스트리의 키를 가리키면 H, 가상 레지스트리 키를 가리키면 V임. (주: ZwDeleteKey의 인수는 핸들임)

IV. 레지스트리 가상화 구현

구현하는 레지스트리 가상화 함수의 이름은 Native API 함수 이름 앞에 'VM_'이 추가된다. 예를 들어 함수 ZwCreateKey에 대한 가상화 함수는 VM_ZwCreateKey이다. 본 논문에서는 2.2에서 기술한 표 1에 있는 함수들에 대한 가상화 함수를 C 언어로 프로그램하여 윈도우 운영체제에서 동적 라이브러리로 구현하였다. 구현된 프로그램의 소스 코드는 약 4,500줄이고 생성된 라이브러리 이름은 vm.dll이다.

윈도우 운영체제 상에서 가상 프로세스는 가상 자원을 사용하여야하기 때문에 가상 프로세스 생성 시 라이브러리 vm.dll을 프로세스에 주입(inject)하여야 한다. 이를 위하여 본 논문에서는 마이크로소프트에서 개발한 Detours[10]라는 주입 도구를 사용하였다. 이 도구는 주어진 라이브러리를 프로세스에 주입하여 프로세스가 주입된 라이브러리를 사용하여 수행되게 하는 도구이다. 예를 들어 워드패드(wordpad.exe) 응용 프로그램에 본 논문에서 개발한 vm.dll 라이브러리를 주입하여 수행시키면 워드패드 응용 프로그램이 가상화한 함수를 불러서 가상 프로세스로 수행하게 된다. 그림 3은 워드패드 응용 프로그램에 vm.dll이 주입되어 가상 프로세스로 수행되는 모습을 프로세스 관찰 도구인 Process Explorer[18]를 사용하여 확인한 결과이다. 그림 3에서 워드패드에 레지스트리 가상화 라이브러리 vm.dll이 주입되어 가상 프로세스로 수행됨을 확인할 수 있다.

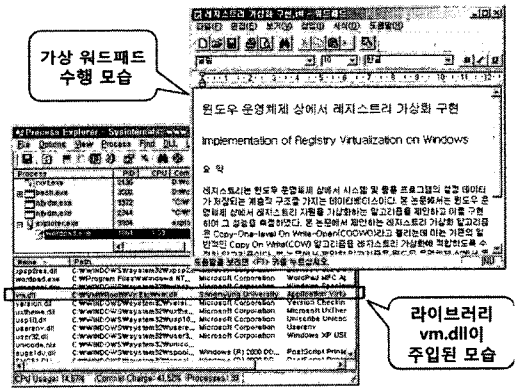

그림 3 가상 워드패드 수행 모습
Fig. 3. Execution of a Virtual Wordpad

V. 성능 평가

가상 프로세스는 호스트 프로세스에 비하여 같은 작업을 수행하는데 걸리는 수행 시간이 길어진다. 본 논문에서는 가상 프로세스가 호스트 프로세스보다 어느 정도 수행 시간이 더 길어지는지에 대하여 측정하고 평가한다. 우리가 윈도우에서 수행하는 대부분의 응용 프로그램은 프로그램의 작업 시간 중 극히 일부 시간 동안 레지스트리를 사용한다. 또 어떤 응용

프로그램은 수행 시간 중 전혀 레지스트리를 사용하지 않는 경우도 많다. 이러한 이유 때문에 일반 응용 프로그램을 대상으로 가상화 수행 시 수행 시간을 측정하여 비교하는 것은 불합리하다. 왜냐하면 대부분의 측정에서 수행 시간의 차이가 나지 않을 것이기 때문이다. 본 논문에서는 이러한 불합리한 점을 막기 위하여 레지스트리만 집중적으로 사용하는 윈도우 유틸리티인 REG[19]를 사용하여 성능을 측정하고 평가한다. 윈도우 레지스트리 유틸리티 REG는 수행 도중 대부분의 시간을 레지스트리 입출력에 소비한다. 표 5는 REG 도구의 주요 기능을 설명한 표이다.

표 5. 윈도우 REG 유틸리티
Table. 5. Windows REG Utility

REG 명령어 옵션	기능
REG COPY	주어진 레지스트리 키를 다른 곳으로 복사한다. (재귀적 수행 가능)
REG DELETE	주어진 레지스트리 키 및 모든 하위 서브키를 제거한다.
REG QUERY	주어진 레지스트리 키에 대한 정보를 출력한다. (재귀적 수행 가능)
REG EXPORT	주어진 레지스트리 키 및 모든 하위 서브키를 .reg 파일에 저장한다.
REG IMPORT	.reg 파일에 저장된 레지스트리 키를 원래 레지스트리 위치에 읽는다.

본 논문에서는 약 15,000 개의 레지스트리 키 및 레지스트리 값으로 구성된 시험 용 레지스트리를 대상으로 표 5에 있는 동작을 호스트 REG 프로세스와 가상 REG 프로세스로 각각 동작시켜 CPU 수행 시간을 측정하였다. CPU 수행 시간은 사용자 모드에서의 CPU 수행 시간과 커널 모드의 CPU 수행 시간으로 나누어지는데 본 논문에서는 이들 시간을 합하였다. 측정값의 정확성을 높이기 위하여 같은 작업을 100회 수행하여 평균값을 계산하였다. 표 6은 각 동작의 CPU 수행 시간 차이를 표로 나타낸 표이다. 이 표에서 모든 동작을 종합한 평균 CPU 시간 증가율은 26.24%이다.

표 6. 수행 시간 비교
Table. 6. Comparison of Execution Times

REG 명령어 옵션	호스트 프로세스 CPU 수행시간 (milli sec)	가상 프로세스 CPU 수행시간 (milli sec)	CPU 수행 시간 증가율 (%)
REG COPY	994.13	1075.72	8.21
REG DELETE	231.46	379.68	64.04
REG QUERY	976.44	1016.29	4.08
REG EXPORT	132.45	202.64	52.99
REG IMPORT	918.33	935.38	1.86
평균 CPU 수행 시간 증가율 (%)			26.24

보통의 응용 프로그램이 수행 시간 중 짧은 시간 동안 레지스트리를 사용하는 것을 생각하면 레지스트리 가상화에 의하여 전체 프로세스의 수행 시간 증가는 크지 않을 것으로 판단된다. 예를 들어 수행 시간 중 약 5%의 시간을 레지스트리 사용에 소비하는 프로세스가 있다고 가정하면(주: 매우 크게 가정한 것임) 레지스트리 가상화 프로세스로 수행시켰을 때 5%*26.24%=1.3%의 CPU 수행 시간의 증가가 있을 것이라고 예측할 수 있다. 일반적으로 가상화 기계가 호스트 기계에 비하여 약 5% 이내의 성능 차이가 있으면 사용자가 큰 불편이 없다고 사용할 수 있다고 판단된다. 이러한 측면에서 보면 본 논문에서 설계한 레지스트리 가상화 프로그램은 충분히 사용 가능한 범위 내에 들어가는 것으로 평가된다. 본 논문에서는 설계와 구현에 중점을 두었지만 앞으로 성능 향상을 위한 연구가 추가로 이루어진다면 더 좋은 성능이 충분히 가능할 것으로 판단된다. 표 6의 수행 시간 비교에서 KEY DELETE 및 KEY EXPORT의 CPU 수행 시간 증가율이 큰 이유는 이들 프로그램이 읽기 열기보다 쓰기 열기 혹은 제거 동작을 많이 수행하여 CPU 수행 시간을 많이 소비한 것으로 추측된다.

VI. 결 론

본 논문에서는 윈도우 운영체제 상에서 레지스트리 자원을 가상화하는 알고리즘을 제안하고 이를 구현하여 그 성능을 측정하였다. 본 논문에서 제안하는 레지스트리 가상화 알고리즘은 Copy-One-level On Write-Open(COOWO)라고 불리는데 이는 기존의 일반적인 COW) 알고리즘을 레지스트리 가상화에 적합하도록 수정한 알고리즘이다. 본 논문에서 제안한 알고리즘은 윈도우 운영체제 상에서 동적 라이브러리 형태로 구현되어 다양한 윈도우 응용 프로그램의 가상화 수행에 적용하였으며 레지스트리 가상화에 따른 성능 저하가 어느 정도인지를 측정하였다. 측정 결과 레지스트리 동작에 대하여 약 26.24%의 수행 시간 증가가 있음을 발견하였다 이 수치는 약 5%의 레지스트리 사용 응용 프로그램의 경우 1.3%의 프로세스 수행 시간의 증가를 의미한다. 본 논문은 레지스트리 가상화 구현에 대한 구체적인 내용을 소개한 논문이 많지 않은 현실에서 레지스트리 가상화 알고리즘 및 구현 내용을 상세하게 기술하고 구현된 프로그램의 성능을 측정해 보았다는 관점에서 의의가 있다. 본 논문에서 구현한 '레지스트리 가상화 모듈'은 좀 더 성능 향상을 위한 연구를 거친 뒤 본 연구 팀에서 기개발한 '파일 가상화 모듈'[11]과 통합되어 '윈도우 프로세스 가상 기계' 개발에 적용될 예정이다.

참고문헌

[1] Philip Dawson and Thomas J. Bittman, "Virtualization Changes Virtually Everything," Gartner Research ID Number G00156488, Mar. 2008.

[2] James E. Smith and Ravi Nair, "The Architecture of Virtual Machines," IEEE Computer, Vol. 38, No. 5, pp. 32-38, May 2005.

[3] Steve Herrod, "The Future of Virtualization Technology", Proceedings of the 33rd International Symposium on Computer Architecture, 2006.

[4] Philip Winslow, Robert Semple, Jason Maynard, Dennis Simson and Bryan McGrath, "Desktop Virtualization Comes of Age," Equity Research of Credit Suisse, Nov. 2007.

[5] A. Azagury, M. E. Factor, and J. Satran. "Point-in-time copy: Yesterday, today and tomorrow," Proceedings of the Tenth Goddard Conference on Mass Storage Systems and Technologies, pp. 259 - 270. IEEE, April 2002.

[6] Zachary Nathaniel Joseph Peterson, "Data Placement For Copy-On-Write Using Virtual Contiguity," MS Thesis, University of California Santa Cruz, September 2002.

[7] VMware, "VMware ThinApp User's Manual," VMware ThinApp 4.0, VMware, Inc., 2008.

[8] Yang Yu Fanglu Guo Susanta Nanda Lap-chung Lam Tzi-cker Chiueh, "A Feather-weight Virtual Machine forWindows Applications," Proceedings of Virtual Execution Environment, June 2006.

[9] Yang Yu, "OS-level Virtualization and Its Applications," Ph.D. Thesis, Stony Brook University, December 2007.

[10] Galen Hunt and Doug Brubacher, "Detours: Binary Interception of Win32 Functions," Proceedings of the 3rd USENIX Windows NT Symposium, July 1999.

[11] Dongha Shin, "A File Virtualization Scheme on Windows," International Transaction on Computer Science & Engineering, Feb. 2009.

[12] James E. Smith and Ravi Nair, "*Virtual Machines Versatile Platforms for Systems and Processes*," Elsevier, 2005.

[13] Jerry Honeycutt, "*Microsoft Windows Registry Guide*," Second Edition, Microsoft Press, August 2005.

[14] Johnson M. Hart, "Windows System Programming," Third Edition, Addison Wesley Professional, Oct. 2004.

[15] Gary Nebbett, "*Windows NT/2000 Native API Reference*," MTP, 2000.

[16] 신동하, 김지연, "uC/OS-II 실시간 커널의 가상화를 위한 하이퍼바이저 구현," 한국컴퓨터정보학회논문지, 제 12권, 제 5호, 103-112쪽, 2007년 11월.

[17] 손성훈, 이재현, "임베디드 시스템을 위한 가상 머신 모니터의 설계와 구현," 한국컴퓨터정보학회논문지, 제 14권, 제 1호, 57-64쪽, 2009년 1월.

[18] Process Explorer v11.33, *http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx, Mark Russinovich, 2009.*

[19] Managing the Windows Registry with Reg.exe, *http://commandwindows.com/reg.htm*

저 자 소 개

신 동 하

1980년 : 경북대학교 전자공학과 전자계산기전공 학사

1982년 : 서울대학교 전자계산기공학과 석사

1994년 : University of South Carolina 컴퓨터과학과 박사

1982년 ~ 1996년 : 한국전자통신연구원 책임연구원

1997년 ~ 현재 : 상명대학교 컴퓨터과학부 교수

관심 분야 : 가상화, 임베디드 컴퓨팅, 논리 프로그래밍, 리눅스 및 윈도우 시스템 프로그래밍