임베디드 시스템 가상화에서의 메모리 압축을 통한 페이지 스왑 기법 디자인*

이치영*, 유혁*
*고려대학교 컴퓨터전파통신공학과
e-mail : cylee@os.korea.ac.kr, hxy@os.korea.ac.kr

A Page Swap Technique using Memory Compression of Virtual Machines for Embedded System: Proposal and Design

Chiyoung Lee*, Chuck Yoo*
*Dept. of Computer and Radio Communications Engineering, Korea University

요 약

가상화 기법은 PDA, 스마트 폰과 같은 임베디드 시스템에서 다양한 운영체제와 응용 프로그램들을 제공할 수 있게 한다. 그러나 임베디드 시스템은 매우 제한된 컴퓨팅 자원을 갖고 있기 때문에 많은 수의 가상 머신을 동작하기 어렵다. 특히, 프로세스 동작에 필수적인 메모리 공간의 부족은 반드시 해결되어야 하는 문제이다. 데스크탑과 같은 시스템은 페이지 스왑을 통해 이를 해결하지만, 디스크가 없는 임베디드 시스템은 해결이 쉽지 않다. 본 논문은 메모리 공간 부족 문제를 해결하기 위해 불필요한 메모리 공간의 압축을 이용한 여유 공간의 추가 확보 기법을 제안한다. 페이지 압축을 통해 페이지 스왑하는 것과 유사한 효과를 얻을 수 있게 한다. 이는 가상화로 인한 메모리 분할과 불필요한 프로세스의 메모리 상주 등의 이유로 인한 임베디드 시스템 가상화 환경에서의 메모리 부족 문제를 해결할 수 있다. 본 논문은 기능 구현에 앞서 임베디드 시스템과 가상화 환경에 맞춘 메모리 압축 스왑 기법을 디자인한다.

1. 서론

가상화는 다양한 운영체제와 서비스를 제공할 수 있도록 컴퓨팅 자원에 대한 추상화를 제공한다. 이를 통해 사용자는 더욱 폭넓은 응용프로그램 또는 서비스를 이용할 수 있다. 최근에는 임베디드 시스템의 효율성을 극대화하기 위해 이런 가상화 기법을 임베디드 시스템에 적용하는 연구가 증가하는 추세이다[1][3][4]. 임베디드 시스템의 가상화는 한정된 자원의 효율적인 분배를 통해 다양한 서비스 이용이 가능하게 한다. 또한, 가상 머신(virtual machine) 간의 고립성(isolation)을 제공하여 서로 간 간섭을 막는다. 이는 사용자에게 하나의 시스템을 이용하는 것과 같은 효과를 준다. 그러나 한정된 자원의 분배는 각 가상 머신의 이용 가능한 자원이 더욱 줄어들게 된다는 단점이 있다. 그러므로 각 자원을 효율적으로 분배 및 관리하기 위한 기법들이 요구된다.

특히, 메모리는 프로세스 실행에 필수적인 자원으로, 여유 공간이 부족할 경우 프로세스 실행 속도를 저하시키거나 실행이 중단될 수 있다. 기존 컴퓨팅 시스템에서는 메모리의 공간 부족을 해결하기 위해서 페이지 스왑(page swapping)을 이용한다. 페이지 스왑은 사용하지 않는 페이지를 디스크로 이동시키고, 프로세스가 요구한 페이지를 메모리에 적재하는 기법이다. 그러나 임베디드 시스템은 디스크 대신 NAND 플래시 메모리(NAND flash memory)를 저장 장치로 이용하기 때문에 페이지 스왑이 어렵다[5]. NAND 플래시 메모리는 다시 쓰기 성능이 좋지 않고, 잦은 쓰기 작업으로 플래시 메모리의 수명이 줄어들기 때문이다. 그러므로 페이지를 저장 장치에 기록해야 하는 페이지 스왑은 플래시 메모리를 사용하는 임베디드 시스템에 적합하지 않다. 임베디드 시스템을 위한 운영체제들은 페이지 스왑을 이용하지 않고 메모리 부족 문제를 해결하기 위해 프로세스 강제 종료 기법을 이용한다. 사용하지 않는 메모리 상주 프로세스 또는 현재 동작하지 않는 프로세스를 강제 종료하고 할당된 메모리를 회수한다. 그러나 이 기법은 프로세스 간의 상호 참조 또는 의존성 문제로 예상치 못한 시스템의 오류를 발생시킬 가능성이 있다. 그러므로 임베디드 시스템에서의 메모리 부족 문제를 해결하기 위한 새로운 기법이 요구된다.

따라서, 본 논문은 임베디드 시스템에서 가상화 특성을 고려한 메모리 여유 공간 확보 기법을 제안하고 디자인한다. 제안된 기법은 Xen on ARM[1][2]을 기반의 가상화 환경에서 각 가상 머신의 페이지 압축을

* 이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2010-0029180)

수행한다. 페이지 압축으로 기존 페이지가 가진 공간의 크기를 줄여 추가적인 여유 공간을 확보한다. 이를 위해 페이지 압축을 이용한 페이지 스왑 기법을 디자인하고, 향후 구현과 성능 평가의 기초로 삼는다.

본 논문의 구성은 다음과 같다. 2 장에서는 기존의 메모리 여유 공간 확장 기법을 공유와 압축 두 가지 방법을 이용한 기법들을 소개하고, 3 장에서 임베디드 시스템과 가상화 환경을 고려한 페이지 압축 기법을 디자인한다. 마지막으로 4 장에서는 향후 연구 방향에 대한 논한다.

2. 메모리 여유 공간 확장 기법

프로세스는 가상 메모리 공간을 가정하고 동작하기 때문에 실제 물리 메모리의 크기보다 큰 메모리를 요구한다. 데스크 탑 환경에서는 하드디스크를 이용한 페이지 스왑을 통해 이를 해결한다. 그러나 임베디드 시스템에서는 하드디스크의 부재로 스왑이 어렵다[5]. 또한, 가상화는 물리 메모리를 여러 가상 머신들이 나누어 사용하기 때문에 프로세스가 이용할 수 있는 물리 메모리 크기가 더 크게 줄어든다. 이런 메모리 부족 문제를 해결하기 위한 많은 연구가 진행되어 왔다. 기존의 메모리 여유 공간 확보 기법에는 프로세스 또는 가상 머신 간의 동일한 페이지 공유를 통한 기법과 사용하지 않는 페이지의 압축을 통한 기법이 있다. 이 장에서는 메모리 여유 공간 확보를 위한 다양한 기법들을 분석한다.

2.1. 공유를 이용한 여유 공간 확장

공유를 이용한 여유 공간 확보는 각 프로세스 또는 가상 머신에서 사용 중인 페이지를 대상으로 한다. 즉, 사용 중인 페이지 중 동일한 페이지를 하나의 페이지로 통합, 공유한다. 이를 통해 중복된 페이지들 제거하여 메모리의 여유 공간을 늘린다. 페이지의 공유를 위해서는 동일한 페이지를 찾는 것이 선행되어야 한다. 그러므로 페이지 공유를 사용하는 기법들은 가장 효율적인 페이지 비교 기법을 찾는 것에 초점을 맞춘다.

페이지 내용을 비교하는 방법으로 가장 생각하기 쉬운 방법은 하나의 페이지를 모든 페이지와 내용을 비교하는 것이다. 이런 페이지 스캔과 비교를 이용한 페이지 공유 기법으로 Content-based page sharing[6]이 있다. 이 기법은 메모리 스캔, 페이지 비교, 페이지 공유의 3 단계로 구성된다. 메모리 스캔 단계에서는 메모리 내의 모든 페이지를 해싱(hashing)하고, 해싱 테이블을 생성 및 유지한다. 이 해싱 테이블은 동일한 페이지가 존재하는지 해싱 값 비교로 찾는데 사용된다. 만약 동일한 해싱 값이 존재하면 페이지를 비트 비교(Bitwise-Comparison)한다. 해싱 값만으로는 내용이 동일함을 신뢰하기 어렵기 때문이다. 공유된 페이지에 대해 쓰기 명령이 요청된 경우, COW(Copy On Write)를 사용하여 페이지를 분리한다. 이를 위해 공유된 페이지에 대해 쓰기 오류(Write Fault)를 발생시키기 때문에 쓰기 연산에 대한 지연이 발생한다. 또한 주기적인 메모리 스캔 및 해싱으로 인한 오버헤드가 발생하는 단점이 있다.

이런 문제를 줄이기 위해 실제 사용하는 페이지만을 해싱하는 기법(Satori[7])이 제안되었다. 이 기법은 사용 중인 페이지를 찾기 위해 페이지 캐시(Page Cache)를 이용한다. 모든 페이지는 디스크에서 메모리로 적재될 때 반드시 페이지 캐시에 먼저 적재되기 때문에 메모리 스캔 없이 모든 페이지를 찾을 수 있다. 선택된 페이지들은 해싱을 통한 공유(Content-based Sharing)와 COW 를 이용한 공유(Copy on Write Sharing)의 2 가지 방법에 의해 공유된다. 첫번째 방법은 페이지 캐시에 있는 블록(block)들을 해싱하고, 해싱 값 비교를 통해 동일한 페이지를 찾는다. 두번째 방법은 디스크에서 읽은 블록의 블록 번호(block number)와 I/O 버퍼의 MFN(Machine Frame Number)의 맵핑 정보를 이용한다. 다른 가상 머신이 동일한 블록을 읽기 요청하면, 맵핑된 MFN 을 공유한다. 쓰기 요청인 경우, COW 에 의해 새로운 머신 프레임(machine frame)을 할당한다. 페이지 캐시의 이용을 통해 메모리 스캔으로 인한 오버헤드를 줄일 수 있지만, 페이지 캐시를 거치지 않는 페이지는 공유할 수 없는 단점이 있다. 특히 xen 기반 환경에서는 커널 텍스트(kernel text)를 xen 도메인 빌더(xen domain builder)가 직접 메모리에 적재하기 때문에 페이지 캐시를 거치지 않는다.

이러한 공유 기법들은 서버와 같은 동일한 작업이 반복되는 환경에 매우 적합하다. 그러나 스마트 폰과 같은 시스템은 매우 다양한 응용 프로그램과 서비스를 실행하기 때문에 페이지의 내용이 동일하지 않는 경우가 많다. 그러므로 페이지의 공유로 인해 얻을 수 있는 이득이 줄어들기 때문에 다른 기법이 요구된다.

2.2. 압축을 통한 여유 메모리 확보

페이지 공유와 달리, 페이지 압축을 이용한 기법은 사용하지 않는 페이지들을 대상으로 한다. 따라서 사용하지 않는 페이지들을 디스크로 스왑하는 것과 유사하다. 페이지 스왑은 디스크를 이용하기 때문에 디스크라는 별도의 하드웨어가 필요하고 디스크의 느린 접근 속도로 인해 교체 속도가 상대적으로 느리다. 반면, 페이지 압축은 메모리의 일부를 이용하기 때문에 접근 속도가 빠르다.

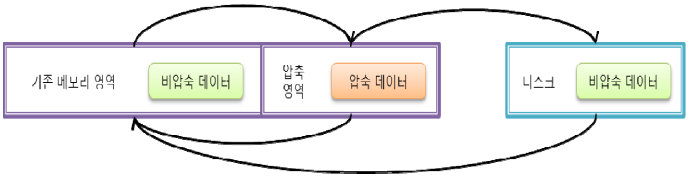

그림 1 페이지 압축을 이용한 페이지 스왑 성능 향상

이 같은 장점을 이용하여 페이지 스왑 성능을 높인 기법이 제안되었다[8]. 이 기법은 페이지 스왑의 성능이 느린 디스크의 I/O 속도로 인해 저하되는 문제를 메모리 내의 페이지 압축으로 해결하였다. 그림 1과

같이 기존 페이지의 스왑에 앞서 페이지를 압축하고, 압축된 영역이 가득 차면 압축된 페이지 중 일부를 디스크로 스왑한다. 또한, 기존 메모리 영역에 대한 영향을 줄이기 위해 여유 공간(free space)의 크기에 따라 압축 영역을 동적으로 구성할 수 있도록 한다. 이를 통해 디스크로의 접근 횟수가 줄어 스왑 성능이 향상된 반면, 스왑된 페이지를 찾기 위해 압축 영역과 디스크를 순차적으로 검색해야 하는 단점이 있다.

또 다른 페이지 압축을 이용한 페이지 스왑 기법으로 CRAMES[9]가 있다. 이 기법은 페이지 압축을 수행하는 가상의 블록 장치(block device)를 제공한다. CRAMES 는 메인 메모리를 압축 영역(Compressed Area)과 비압축 영역(Uncompressed Area)으로 구분한다. 비압축 영역은 프로세스가 실행 중에 사용하는 페이지가 위치하는 공간이고, 압축 영역은 압축된 페이지가 저장되는 공간이다. 이 장치는 프로세스가 장치에 요청을 하는 경우, 읽기(read)와 쓰기(write), 2 가지의 명령에 의해 해당 블록에 대해 압축과 복원을 수행한다. 쓰기 명령은 요청된 블록을 압축하고 압축된 블록을 압축 영역에 할당된 공간에 저장한다. 이 후 맵핑 테이블에 기록하여 읽기 시에 원하는 블록을 찾을 수 있도록 한다. 읽기 명령은 블록 번호를 이용해 맵핑 테이블에서 원하는 압축 블록을 찾아 복원한다.

이들 기법들은 압축을 이용해 메모리의 여유 공간 문제를 해결하면서 성능 향상을 꾀한다. 그러나 이들은 하나의 커널이 동작하는 비가상화 환경에 기반한 기법들이기 때문에 모든 가상 머신이 이용하기가 어렵다. 이는 모든 가상 머신에서 동작 중인 운영체제에 맞게 구현을 달리해야 하기 때문이다. 그러므로 가상화 환경을 고려한 압축 기법이 요구된다.

3. 가상 머신 독립적인 페이지 압축 기법

우리는 가상화된 임베디드 시스템에서 실제로 메모리 문제가 발생하는가를 확인하기 위해 간단한 실험을 진행하였다. ARM11 기반의 S3C-6410 보드에서 Xen on ARM 을 이용한 가상화 환경을 구축하였다. 여기에 2 개의 가상 머신을 동작시키고, 각각 리눅스와 안드로이드를 구동하였다.

표 1 최소 메모리 요구량

	비가상화	가상화
리눅스(non-GUI)	8 MB	22 MB
안드로이드	96 MB	104 MB

표 1은 각 게스트 운영체제의 메모리 필요량을 의미한다. 비가상화는 xen on arm 없이 시스템에 하나의 운영체제만 동작시키는 경우 최소 메모리 요구량이고, 가상화는 2 개의 가상 머신 상에서 동시에 두 운영체제를 동작시킨 경우를 말한다. 표 1에서 보이는 바와 같이, 리눅스와 안드로이드는 각각 최소한 8MB, 96MB 를 필요로 한다. 또한 가상화를 하게 되면 이 요구량은 더욱 증가하여 22MB 와 104MB 를 필요로 한다. 이는 가상화 지원을 위한 도구와 라이브러리에 의해 나타난다. 즉, 2 개의 가상 머신을 동작하는 가상화 시스템의 메모리 요구량은 118MB 에서 126MB 로 매우 크다. 이 실험 환경과 같이 가상화된 시스템은 시스템 자원을 관리하는 하이퍼바이저와 그 위에서 동작하는 가상 머신으로 구성된다. 가상 머신은 게스트 운영체제에게 주어진 가상의 시스템 자원들을 제공한다. 따라서 게스트 운영체제가 접근하는 자원들은 실제 시스템과는 차이가 있을 수 있다. 이로 인해 기존의 커널에서 동작하는 페이지 압축 기법으로는 실제 메모리에서의 정상적인 동작을 보장할 수 없다. 그러므로 가상화의 특성을 고려하여 페이지 압축을 이용한 관리 구조를 새로 디자인해야 한다.

이를 위해 본 논문은 페이지 압축을 이용한 스왑 구조를 새로 제안한다. 제안하는 기법은 실제 메모리에 대한 정보를 게스트 운영체제의 커널이 모른다는 점과 각 게스트 운영체제의 개입을 최소화해야 한다는 점에 초점을 맞췄다. 그림 2은 새로운 스왑 구조의 개념도이다.

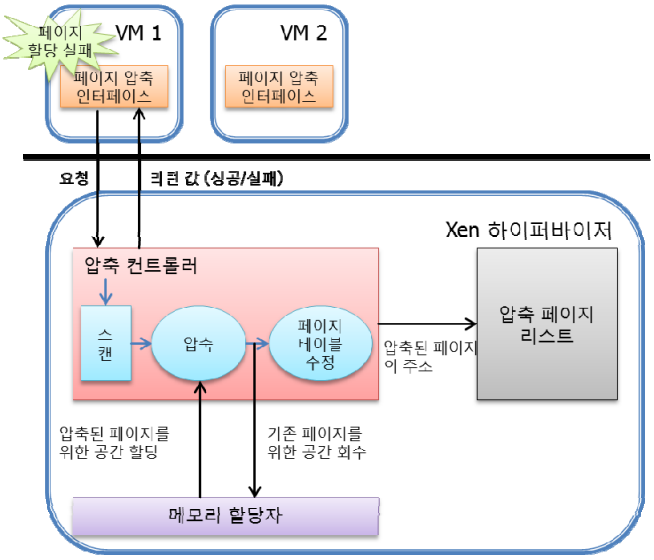

그림 2 임베디드 가상화 시스템에서의 페이지 압축을 이용한 스왑 기법

이 구조는 페이지 압축과 관리를 담당하는 압축 컨트롤러를 xen 하이퍼바이저가 갖도록 하고, 각 가상 머신의 개입을 최소화하였다. 물리 페이지에 대한 정보는 하이퍼바이저가 알고 있기 때문에 직접 페이지를 조작하는 기능을 모두 하이퍼바이저가 갖도록 하였다. 또한, 압축 컨트롤러를 하이퍼바이저로 옮겼기 때문에 가상 머신 상에서 동작 중인 각 커널은 단순히 압축 요청만을 수행하게 된다. 이는 다양한 커널이 동작할 가능성이 있는 환경에서 복잡한 압축 메커니즘을 커널마다 새로 구현하지 않도록 한다. 이를 통해 해당 기법을 추가함으로 인해 발생할 수 있는 가상화 노력(virtualization effort)을 줄일 수 있다. 압축 컨트롤러는 스캔과 압축, 맵핑된 페이지 테이블 수정이라는 세가지 루틴으로 구성된다. 스캔 루틴은 압축 대상 페이지를 찾기 위한 것으로, 요청된 현재 프로세스의 페이지 테이블을 통해 접근하는 모든 페이지

를 검색한다. 하나의 프로세스가 접근하는 페이지들만을 검색 범위로 하기 때문에 전체 메모리를 스캔하는 것에 비해 지연 시간이 크게 줄어든다. 검색된 페이지들 중 압축 대상을 선택하는 것은 LRU 알고리즘을 이용한다. 이는 압축된 페이지의 복원 횟수를 줄여 성능 저하를 막기 위한 것이다. 압축 루틴은 선택된 페이지를 LZO 압축 알고리즘[10]을 통해 압축한다. 이때 메모리 할당자로부터 임의의 공간을 할당받아 압축 완료된 페이지를 저장한다. 그 후, 원래 페이지가 차지하던 공간을 회수(free)한다. 이 과정을 통해 압축된 양만큼의 여유 공간을 추가 확보하게 된다. 이 후 대상 페이지가 맵핑된 페이지 테이블을 수정하여 페이지 접근 시 페이지 폴트(page fault)가 발생하게 한다. 이는 프로세스가 해당 페이지를 요청했을 때, 복원을 하기 위한 루틴을 수행하도록 하기 위함이다. 압축된 페이지는 리스트 형태로 기록되어 요청 시 페이지를 쉽게 찾을 수 있도록 한다. 이와 같은 구조의 변화는 페이지 압축을 통한 메모리의 여유 공간 확보와 함께 게스트 운영체제의 수정을 위한 노력을 최소화할 수 있다.

4. 결론

가상화된 임베디드 시스템은 항상 메모리 부족 문제가 발생할 가능성을 지니고 있다. 임베디드 시스템은 적은 크기의 메모리를 갖고 있고, 가상화로 인해 다수의 운영체제가 이를 분할해 사용하기 때문이다. 이는 운영체제와 관계없이 다양한 응용프로그램과 서비스를 제공하고자 하는 목적에 좋지 않은 영향을 준다. 그러므로 이런 메모리 부족 문제를 해결할 수 있는 기법이 요구된다.

이를 위해 기존에는 메모리 공유와 압축을 이용한 기법들이 연구되어 왔다. 메모리 공유는 동일한 내용의 페이지를 하나의 페이지로 통합하는 기법이다. 이런 공유 기법들은 서버와 같은 동일한 작업이 반복되는 환경에 매우 적합하다. 그러나 다양한 응용 프로그램과 서비스를 실행해야 하는 스마트폰에는 적합하지 않다. 또 다른 해결책으로 압축을 이용한 기법이 있다. 이는 앞으로 사용하지 않을 페이지를 찾아 압축을 함으로써 해당 페이지가 차지하는 메모리 공간의 크기를 줄인다. 대상 페이지의 선택과 메모리 압축/복원으로 인한 지연시간이 발생하기 때문에 이를 최소화해야 한다. 또한, 압축 이용한 메모리 관리 기법들은 아직 가상화된 환경을 고려하지 않고 있다.

본 논문에서는 압축/복원의 지연시간을 줄이기 위해 가장 효율적인 압축 성능을 가진 것으로 알려진 LZO 알고리즘을 사용한다. 또한, 페이지의 검색의 지연시간을 줄이기 위해 메모리 스캔의 범위를 전체가 아닌, 현재 프로세스가 접근 메모리로 제한한다. 마지막으로 가상화 환경을 고려하여 복잡한 페이지 압축 및 관리 루틴을 하이퍼바이저가 갖도록 한다. 이를 통해 가상화 환경에서 성능 저하를 최소화하면서 메모리의 여유 공간을 확보할 수 있다.

본 논문은 이런 가상화 환경에서의 페이지 압축을 이용한 페이지 스왑 기법을 제안하고 디자인하였다. 이를 향후 실제 시스템의 구현과 성능 평가를 위한 기초 토대로 사용한다. 향후에는 실제 임베디드 시스템에 제안한 스왑 기법을 구현하고, 각 가상 머신에서의 증가된 메모리의 양과 압축/복원으로 인한 지연시간, 전체 시스템 구동에 있어서 미치는 영향 등을 측정할 계획이다.

참고문헌

[1] J. Hwang, S. Suh, S. Heo, C. Park, J. Ryu, S. Park, and C. Kim, “Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones,” Proceedings of the 5th Annual IEEE Consumer Communications & Networking Conference, USA, January 2008.

[2] Xen on ARM, http://wiki.xen.org/xenwiki/XenARM

[3] Cheol-Ho Hong, Miri Park, Seehwan Yoo, Chuck Yoo, “Hypervisor Design Considering Network Performance for Multi-core CE Devices,” IEEE International Conference on Consumer Electronics, Jan. 2010.

[4] Seehwan Yoo, YoungPil Kim and Chuck Yoo, “Real-time Scheduling in a Virtualized CE Device,” IEEE International Conference on Consumer Electronics, Jan. 2010.

[5] Sangduck Park, Hyunjin Lim, Hoseok Chang, and Wonyong Sung, “Compressed Swapping for NAND Flash Memory Based Embedded Systems,” LNCS - Embedded Computer Systems: Architectures, Modeling, and Simulation, Vol. 3553, pp. 314-323, July 2005.

[6] J. Kloster, J. Kristensen, and A. Mejlholm, “On the Feasibility of Memory Sharing: Content-Based Page Sharing in the Xen Virtual Machine Monitor,” Master's thesis, Department of Computer Science, Aalborg University, June 2006.

[7] Grzegorz Miło´s, Derek G. Murray, Steven Hand, and Michael A. Fetterman, “Satori: Enlightened page sharing,” USENIX Annual Technical Conference 2009, June 2009.

[8] Irina Chihaia Tuduce and Thomas Gross, “Adaptive Main Memory Compression,” USNIX Annual Technical Conference 2005, April 2005.

[9] L. Yang, R. P. Dick, H. Lekatsas, and S. Chakradhar, “Online Memory Compression for Embedded Systems,” ACM Transactions o Embedded Computing Systems, Vol. 9, No. 3, Article 27, February 2010.

[10] LZO Algorithm, http://www.oberhumer.com/opensource/lzo/