MMU기능을 이용한 임베디드 시스템의 가상화

손 성 훈, 홍 진 욱

Virtualization of Embedded System Using Functions of MMU

Sunghoon Son , Jinwook Hong
Dept of Computer Science, Graduate School, Sangmyung University

요 약

근래의 가상화 기술에서 가상 메모리의 사용은 다양한 장점을 이끌어 내며 가상화의 중요한 구현 기술 중 하나로 인식되고 있다. 본 논문에서는 선행된 플랫 메모리 임베디드 시스템을 대상으로 하는 가상 머신 모니터에 가상 메모리를 구현한다. 제안된 가상 메모리 기술은 하나의 임베디드 시스템 상에 다수의 실시간 운영체제들을 동시 수행하는 것을 가능하게 하는 가상 머신 모니터의 본질적인 장점과 가상 메모리를 통해 예상할 수 있는 하부 시스템을 제안한다. 이를 통해 향후의 메모리 가상화를 통해 확장 가능한 연구 주제를 제안한다.

1. 서론

기본적인 가상화의 구현 기술은 각 운영체제에 대한 제어 이동과 물리 자원의 관리, 그리고 이를 위한 스케줄링 정책이다. 위의 기본적인 구현으로 얻을 수 있는 가상화의 기본적인 장점들은 시스템 부하와 공간 자원의 효율적 활용, 오류의 영향력 제한을 통한 시스템의 안정성 향상 등을 들 수 있다.

임베디드 환경에서의 가상화를 주제로 한 선행된 연구들을 통하여 가상화 환경의 필요성, 가상화를 위한 제반사항과 극복해야 할 요소들을 알 수 있다. 이 요소들 중 많은 부분이 메모리와 관련이 있으며 이는 시스템의 기본적인 운영에 영향을 미치는 부분이며 가상화의 목적 중 하나인 시스템의 안정성 향상을 위해 중요시 되는 부분이다.

또한 임베디드 환경에서의 가상메모리를 대상으로 한 연구도 다양한 부분에서 선행되었다. 이러한 선행 연구들을 통해 기존의 시스템들과는 다른 임베디드 환경에서의 메모리에 대한 고려사항들을 파악할 수 있으며 이는 임베디드 가상화 환경에서의 메모리 제반사항과 함께 고려해야할 부분으로 정의 할 수 있다.

본 논문에서는 임베디드 환경의 가상 머신 모니터를 대상으로 가상메모리 지원 및 관리 시스템을 설계하고 구현한다. 이후 제안한 시스템 가상화를 위한 가상 메모리 관리 시스템에 대해 기능적인 평가와 이를 통해 예상할 수 있는 효과들을 제시하고 성능적인 측면에서 가상메모리의 사용이 가져오는 영향을 파악한다. 또한 가상 머신 모니터를 위한 가상 메모리 시스템의 확장을 통해 예상 가능한 응용 및 확장을 제안한다.

2. 가상화에서 가상 메모리의 필요성

가상화된 임베디드 시스템에서도 가상 메모리의 지원이 필요한 이유는 다음과 같다. 먼저 가상화를 지원하는 가상 머신 모니터가 MMU 기능을 활용함으로써 가상 머신들에게 공간적 독립성을 보장할 수 있다. 또한 가상 메모리를 사용하는 임베디드 시스템 운영체제들을 가상화하기위해서는 가상 머신 모니터가 게스트 운영체제의 가상 메모리 지원을 위한 메모리 관리 기능을 가지고 있어야한다.

일반적으로 가상화를 통하여 얻을 수 있는 이점은 다음과 같다[6]. 첫 번째로 복잡한 시스템을 분리할 수 있다. 두 번째로 하나의 기계에서 다수의 운영체제를 동시에 수행시킬 수 있다. 세 번째로 보안성과 안정성을 향상시킬 수 있다. 네 번째로 멀티코어 CPU로의 확장이 용이하다. 다섯 번째로 라이선스를 분리할 수 있다. 최근 이러한 이점을 목적으로 하여 임베디드 환경에서 진행된 다양한 연구들이 있다.

반가상화를 통해 CPU와 인터럽트의 분배하고 가상 머신의 동적관리, 실시간성 유지에 대한 연구[4]가 있다. 하지만 이 연구는 가상머신간의 메모리 영역 관리에 대해 미흡하며 이는 가상머신의 보안성과 안정성에 대한 한계로 이어질 수 있다. 또한 플랫 메모리 환경에 기반한 게스트 운영체제를 대상으로 하고 있어서 가용 운영체제의 적용에 한계를 드러낸다.

가상 메모리는 이미 범용 시스템과 일부 임베디드 시스템 환경에서 그 목적에 부합하는 다양한 형태로 제안되어 연구되어 사용되어 왔다. 시스템 소프트웨어에서 일반

적인 컴퓨팅 환경에서 가상 메모리의 목적은 제한적인 물리 메모리 공간의 논리적 확장, 응용 및 상위 계층의 개발 편의성 보장, 프로세스간의 독립성 보장 및 통신, 프로세스 및 운영체제의 안정성 보장 등이 있다.

근래의 임베디드 시스템에서 역시 이와 유사한 목적성을 가지는 환경이 제공되고 있다. 하지만 대상이 되는 시스템의 목적성과 환경에 따라서 다양한 구조를 갖는 임베디드 시스템의 특성상 하나의 일반적인 구조를 취하지 않는다. 이와 같은 환경에 대해 [2]에서는 임베디드 환경에서의 실시간 운영체제에 대해 4가지로 메모리 모델을 제시하고 있다.

그 첫 번째는 MMU를 사용하지 않는 플랫 메모리 모델이다. 이 모델에서는 어떠한 태스크나 운영체제도 모든 메모리 주소에 접근이 가능하며 일반적으로 포인터의 조작을 통해 데이터를 전달한다. 두 번째는 MMU를 포함하는 프로세스에서 플랫 메모리 모델과 같은 1대1 주소대응구조를 가지며 단순히 MMU와 캐시를 사용하는 모델이다. 세 번째는 모델은 MMU를 사용하고 고정된 메모리 영역을 만들어서 태스크나 운영체제를 정상적이지 않은 접근으로부터 보호하는 구조를 가진다. 마지막 모델은 메모리의 페이지를 동적으로 할당, 제거, 재할당의 동작을 취하고 지능적인 메모리 관리자를 갖는 구조이다. 이 모델을 통해 각 태스크와 운영체제 영역은 보호를 받게 되고 공간에 대한 독립성을 유지할 수가 있다.

가상화 지원환경은 최근 사용용도의 다양성과 복잡도가 증가하는 임베디드 환경에서 그 필요성 또한 맞물려 증가하게 되었다. 하지만 여전히 물리적인 제약이 존재하며 이에 대한 지능적인 활용이 요구된다. 본 논문에서는 임베디드와 가상화 환경간의 필연적으로 예상이 되는 메모리 사용의 증가를 MMU를 사용 하여 공간 효율성과 시스템구성의 유연성을 제공하는 가상 메모리 체계를 제안한다. 또한 각 소프트웨어 계층들 간의 독립성과 게스트 운영체제 간의 독립성보장을 목표로 한다.

3. 가상 메모리 확장 설계 및 구현

본 장에서는 임베디드 시스템의 가상화를 위한 MMU의 기능을 이용한 가상메모리의 설계 및 구현 방법을 제안한다.

1. VMsquare 전체 구조 및 메모리 구조

VMsqure는 물리 자원과 가상 머신 사이에 위치하여 운영체제에게 가상화된 물리자원을 지원하고 관리한다. VMsquare는 하나의 가상 머신을 가지며 이 가상 머신은 사용자에게 관리를 위한 응용을 지원한다. 게스트 운영체제와 VMsuqre의 자원요청 및 반환 등의 통신은 가상 머신 인터페이스를 통해 이루어진다.

다음 그림 2는 본 연구를 통해 수정 되는 VMsuare의 메모리 구조의 수정 전과 후의 모습이다.

수정 전의 VMsquare는 MMU기능을 사용하지 않는

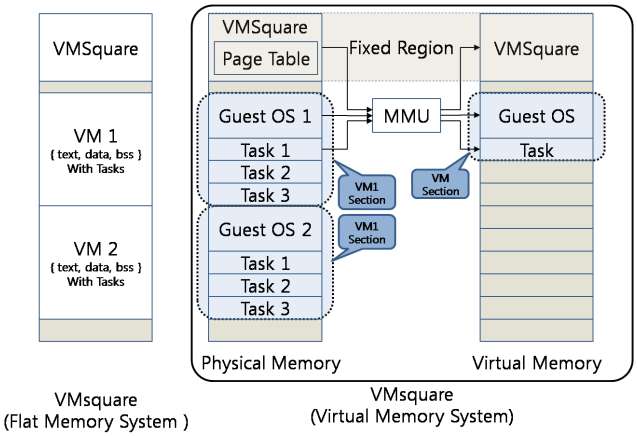

그림 1 VMsaure 수정 전후 메모리 구조

플랫 메모리 구조를 사용하여 구성되었다. 각 가상 머신은 일정한 영역에 고정적으로 위치하여 동작한다.

수정 후의 VMsquare는 MMU를 사용하여 각 가상 머신의 메모리 영역을 구분하여 관리한다. 물리 메모리와 가상 메모리 사이에는 ARM 프로세서가 지원하는 MMU가 위치하여 두 메모리의 주소변환을 수행한다. 가상 주소 공간은 가상 주소가 물리 주소에 대응되는 방법에 따라 크게 세가지 영역으로 나누어진다.

고정 주소 영역은 물리 메모리 주소와 가상 메모리 주소가 1대1로 대응되는 영역이다. 가상화 되지 않은 일반적인 시스템의 운영체제 상에서는 이 영역에 주변장치, 롬 메모리, 페이지 테이블, 그리고 커널 등이 위치한다. 하지만 가상화된 시스템에서는 가상 머신 모니터가 전체 메모리 관리를 담당하기 때문에 게스트 운영체제의 커널이 이 영역에서 빠지고 대신 가상 머신 모니터의 커널이 이 영역에 포함된다.

동적 주소 영역은 가상 주소가 페이지 테이블의 설정에 따라 동적으로 물리 주소로 대응되는 영역이다. 가상 머신을 이루고 있는 게스트 운영체제와 그 태스크들 까지 이 영역에 위치한다. 이를 통해 가상 머신 모니터가 각 가상 머신의 태스크 수준까지 메모리를 관리할 수 있게 되어 일괄적인 메모리 관리가 가능해 진다.

오류 영역은 위에서 언급한 두 영역 이외의 부분에 해당하는 영역으로 페이지 테이블 상에 정의 되지 않은 가상 메모리 공간을 지칭한다. 시스템의 동작 중 이 부분에 대한 접근은 해당 소프트웨어 계층의 오류로 인식되어 적절한 오류 처리를 수행하게 된다.

2. 메모리 접근 관리

본 논문에서는 가상 메모리를 사용하는 게스트 운영체제를 대상으로 가상 머신 모니터를 구성하기 때문에 세 개의 소프트웨어 계층이 존재하고 그에 따라 세 가지 접근권한이 필요하다. 그러나 ARM920T의 MMU는 두 가지의 접근권한만을 지원하기 때문에, 그 성격이 유사한 가상 머신 모니터 계층과 게스트 운영체제 커널 계층은 서로 신뢰할 수 있는 코드라는 전재 하에 동일한 접근권한을 부여한다. 그림 2는 세 소프트웨어 계층들 간의 접근 권한

관점에서의 관계를 도식화한 것이다.

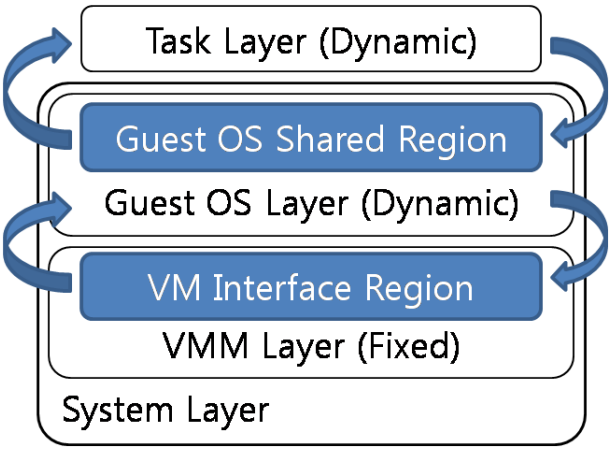

그림 2 소프트웨어 계층 간의 메모리 접근 권한

기본적으로 메모리 접근 권한 관점에서 태스크 접근 권한 계층과 시스템 접근 권한 계층으로 구분된다. 태스크 접근 권한 계층에는 일반적인 사용자 프로세스들이 위치하며 이들은 시스템 접근 권한 계층에 해당하는 메모리 영역에 대한 접근이 불가능하며 두 계층 간의 통신을 위해서는 사전에 정의된 공유 영역을 사용하여야 한다.

시스템 접근 권한 계층에는 게스트 운영체제와 가상 머신 모니터가 위치한다. 가상 머신 모니터와 게스트 운영체제 간에도 개념적인 상하 관계가 존재하긴 하지만 ARM 프로세서가 지원하는 계층이 제한적이기 때문에 한 시스템 계층에 위치한다. 하지만 둘 간의 메모리 접근을 제한하기 위해 둘 간의 통신은 반드시 사전에 정의된 가상 머신 인터페이스 영역을 통해 이루어진다.

게스트 운영체제와 사용자 프로세스는 같은 동적 주소 영역에 위치하지만 둘 간의 접근 권한 계층은 다르다. 둘 간의 통신을 위해서 각 게스트 운영체제는 자신의 태스크와 공유 메모리 영역을 설정하고 통신하여야 한다.

3. 동적 주소 영역의 문맥전환

가상화된 시스템에서는 두 가지의 문맥전환에 대해 고려한다.

첫 번째 태스크 간의 문맥전환 경우, 게스트 운영체제가 가상머신 인터페이스를 통해 다음 문맥전환 대상이 되는 태스크의 정보를 가상 머신 모니터에게 전달한다. 가상 머신 모니터는 요청받은 태스크의 정보를 인지하고 해당 페이지 테이블을 활성화하여 태스크 영역의 가상 메모리 공간에 위치시킨다.

두 번째 게스트 운영체제간의 문맥전환 시에는 해당 게스트 운영체제가 소유하고 있는 태스크를 포함한 문맥전환이 이루어져야한다. 이를 위해 가상 머신 모니터는 게스트 운영체제와 해당 태스크들의 정보를 가지고 있어야 한다. 문맥전환 시 이전 문맥의 운영체제와 당시 수행 태스크의 정보를 기록한 후 다음 문맥의 운영체제와 태스크에 해당하는 페이지 테이블을 활성화한다.

4. 페이지 테이블 구성

가상 메모리의 구성을 이루는 두 가지 요소는 메모리의 영역과 페이지 테이블의 구성이다. 페이지 테이블의 구성은 두 단계로 이루어 구성한다. 그림4는 페이지 테이블 구성의 예를 보여준다.

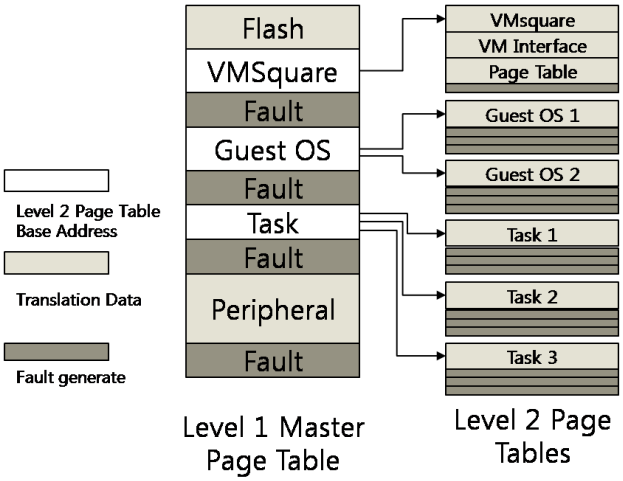

그림 3 페이지 테이블의 구성

1단계의 마스터 페이지 테이블은 가상 메모리 전체의 구조를 보인다. 또한 플래시나 주변 장치 등의 고정 영역에 대한 페이지 할당은 1단계에서 구성한다.

2단계의 페이지 테이블들은 동작 영역에 대한 페이지들을 지정한다. 1단계의 마스터 페이지 테이블의 검색과 2단계 페이지 테이블의 검색을 통해 물리 메모리의 위치를 찾는다. 2단계의 페이지 테이블은 VMsquare가 운영되는 필요한 시스템 관련 페이지 테이블과 동적 영역에 위치하는 게스트 운영체제, 태스크가 위치한다.

4. 성능 평가

본 장에서는 앞에서 제안한 임베디드 시스템 가상화를 위한 가상 메모리 설계 및 구현에 대한 기대 동작 및 성능에 대한 평가를 제시한다. 본 논문에서 구현 및 실험은 ARM920T 기반의 S3C2440칩을 사용한 개발 보드에서 이루어졌으며 LINUX 호스트 상에서 C언어 및 ARM 어셈블리어를 통해 개발 및 실험이 이루어졌다. 게스트 운영체제로는 MMU를 사용하도록 수정된 µCOS-II를 대상으로 하여 진행되었다.

1. 기대 기능 동작 유무

본 실험에서는 메모리 접근 오류에 대한 VMSquare의 처리 동작의 정상 유무를 확인한다. 실험은 3개의 게스트 운영체제가 운영될 때 미리 지정된 태스크의 시스템 영역 접근을 유도하도록 하여 이에 대한 VMSquare의 동작반응을 확인한다.

실험 결과는 해당 태스크의 시스템 접근은 오류를 발생시키고 VMSquare는 이를 감지하여 해당 태스크를 소유하고 있는 게스트 운영체제를 유휴 상태로 전이 시키고 다음 스케줄링 대상으로 제어를 이동시켜 동작이 수행됨

을 볼 수 있었다. 이를 통해 메모리 영역내의 독립성을 보장할 수 있고 이는 시스템 전체의 보안성 향상을 가져다 준다.

2. CPU활용율

본 실험에서는 일정시간 내에 태스크에게 할당 되는 시간의 비율을 측정하여 MMU의 사용이 CPU활용율에 미치는 영향을 확인한다. 실험은 10개단위의 게스트 운영체제를 설정하고 각 게스트 운영체제는 10개의 태스크를 운영한다. 태스크 설정은 다음과 같이 세 가지 형태로 이루어진다.

- 96% VM: CPU-bound 태스크 9개와 I/O-bound 태스크 1개로 구성된 µC/OS-II. (CPU Usage 96%)
- 60% VM: CPU-bound 태스크 5개와 I/O-bound 태스크 5개로 구성된 µC/OS-II. (CPU Usage 60%)
- 12% VM: CPU-bound 태스크 1개와 I/O-bound 태스크 9개로 구성된 µC/OS-II. (CPU Usage 12%)

CPU 활용율

	96%		60%		12%	
nVM(nTick)	non-MMU	MMU	non-MMU	MMU	non-MMU	MMU
10VM(500000)	85%	83%	72%	71%	32%	31%
20VM(750000)	79%	78%	69%	69%	30%	30%
30VM(1000000)	76%	74%	66%	65%	28%	28%
40VM(1250000)	65%	61%	52%	49%	15%	12%
50VM(1500000)	54%	52%	40%	37%	8%	6%

*VM당 할당 tick은 2000Tick으로 설정

표 1 CPU활용율 측정

표1은 해당 실험의 결과를 보여준다. 시스템 상의 부하로 인해 40개의 가상머신 설정부터는 전체적인 CPU활용율의 하락을 볼 수 있지만 MMU의 사용이 CPU활용율에 대해 많은 영향을 미친다고 판단 할 수는 없음을 알 수 있다.

3.Throughput

본 실험은 일정 시간 내에 시스템 전체에서 설정된 동일한 태스크의 작업 완료 여부를 측정하여 MMU의 사용이 전체 시스템 Throughput에 미치는 영향을 확인한다. 실험의 설정은 앞의 CPU활용율 실험의 설정과 동일하다. Throughput의 계속되는 측정과 유휴 시간의 제거를 위해 태스크는 작업 완료 후 종료를 하지 않고 계속하여 다시 수행한다.

Throughput

	96%		60%		12%	
nVM(nTick)	non-MMU	MMU	non-MMU	MMU	non-MMU	MMU
10VM(500000)	156	154	132	132	87	85
20VM(750000)	264	248	232	221	175	172
30VM(1000000)	376	357	304	276	264	258
40VM(1250000)	423	397	385	364	346	321
50VM(1500000)	465	446	435	416	396	365

*VM당 할당 tick은 2000Tick으로 설정
*Task는 자신의 작업 완료후 계속해서 다시 수행

표 2 Throughput 측정

표2에서는 MMU를 사용한 시스템과 MMU를 사용하지 않은 시스템의 Throughput 차이를 볼 수가 있다. 하지만 이 차이가 비율적인 계산으로는 5%미만이고 크지 않은 차이를 가지기 때문에 MMU의 사용 여부가 Throughput에 대해 미치는 영향은 미비하다는 것을 파악할 수 있다.

5. 결론 및 향후 연구

MMU의 사용은 기능적인 면에서 물리 메모리 주소와 가상 메모리의 주소변환을 지원하여 시스템에서 지정된 각 메모리 영역의 독립성을 보장해 주었다. 이는 가상화 환경이 각 가상머신에게 신뢰성을 제공하기 위해서 필요한 부분이다. 이를 통해 시스템은 공간과 권한에 대한 오류에 대해 감지 및 대응이 가능해 진다. 하지만 MMU의 사용은 시스템상의 명령어를 증가시키게 된다. 이와 같이 MMU의 사용에서 예상되는 문제점인 시스템 부하에 대해 그 정도를 파악하고자 실험을 실시하였다. 그 결과 시스템의 성능상 차이는 발생하지만 그 성능의 차이가 non-MMU시스템과 비교하여 5%이하의 미미하다고 판단할 수 있는 차이만을 보였다.

본 논문에서 제시한 연구는 임베디드 시스템의 가상화 환경에 대해 MMU를 사용한 가상 메모리의 적용에 따른 장단점을 파악하였다. 이를 통해 MMU가 시스템 보안성 향상이 기반이 되는 메모리 영역의 독립성 보장을 제공함을 파악하였고 이는 향후 가상화 시스템의 보안성 향상 연구에 기반이 될 수 있을 것이다. 본 논문의 연구를 통해서 가상화 시스템의 오류 검출 및 감내 시스템의 연구를 계획하고 있다.

참고문헌

[1] 손성훈, 이재현, “MobiVMM: 임베디드 장치를 위한 가상머신 모니터”, 한국정보과학회 2009 한국컴퓨터종합학술대회 논문집, 제36권, 제1호(B), 304-309쪽, 2009년 6월.

[2] Leon Tietz. "Virtual memory within embedded systems - marketing hype or engineering reality", Dedicated Systems Magazine, 2001

[3] X. Zhou, P. Petrov, "The interval page table: virtual memory support in real-time and memory-constrained embedded systems", Conference on Integrated Circuits and Systems Design (SBCCI), pp. 294 - 299, 2007.

[4] 손성훈, 이재현, '임베디드 시스템을 위한 가상 머신 모니터의 설계와 구현", 한국컴퓨터정보학회논문지, 제14권, 제1호, 55-64쪽, 2009년 1월.

[5] Disheng Su, Wenzhi Chen, “SmartVisor: Towards an Efficient and Compatible Virtualization Platform for Embedded System”, IIES 2009, pp. 37-41, 2009.

[6] Gernot Heiser, “The Role of Virtualization in Embedded Systems”, IIES 2008, pp. 11-16, 2008.