다중 가상 주소 공간을 지원하는 운영체제 프로세스

김익순, 김선자, 김채규
한국전자통신연구원
융합기술연구부문
e-mail : {ik-soon.kim, sunjakim, kyu}@etri.re.kr

Multiple Virtual Address Spaces for the Operating System Process

Ik-Soon Kim, Sunja Kim, Chae-Kyu Kim
IT Convergence Technology Research Laboratory
Electronics and Telecommunications Research Institute

요 약

본 논문은 운영 체제(Operating System)에서 수행되는 프로세스(Process)의 가상 주소 공간(Virtual Address Space)을 기존의 단일 가상 주소 공간에서 다중 가상 주소 공간으로 확장시켜서, 하나의 프로세스가 기존보다 더욱 넓은 가상 메모리 영역을 쉽게 사용할 수 있도록 해주는 방안을 제안한다. 최근 컴퓨팅 기기들은 비약적으로 증가한 메모리를 쉽게 사용할 수 있는 수단이 필요하다. 최근 PAE(Physical Address Extension)를 지원하는 32 비트 프로세서나 32 비트 명령어를 같이 지원하는 64 비트 프로세서들은 프로세스의 가상 주소 크기보다 더욱 큰 용량의 메모리를 사용할 수 있어서, 한 프로세스가 장착된 메모리의 일부분 밖에 사용할 수 없는 일이 발생한다. 이를 해결하기 위해서 64 비트 프로세서의 경우 64-비트 명령어를 사용하지만 이는 프로그램의 명령어 크기나 포인터 변수 크기의 증가로 메모리 사용량을 크게 늘릴 수 있어서 서버 컴퓨터나 데스크탑 PC 와 같이 충분한 양의 메모리를 장착한 시스템에서만 효과적이다. 본 논문에서 제안하는 다중 주소 공간을 지원하는 프로세스는 모바일 및 임베디드 기기와 같이 상대적으로 제한된 용량의 메모리를 지원하는 시스템에 유용할 것으로 기대한다.

1. 서론

메모리는 컴퓨터 기기에 매우 중요하지만, 용량에 제약이 많은 자원이다. 메모리 없는 컴퓨터 기기는 상상하기 어려운데, 이는 최근의 범용 컴퓨터들이 프로그램과 자료들을 메모리 위에 배치한 후 메모리 상에서 연산을 수행하기 때문이다. 이런 이유로 메모리 용량을 넘어서는 프로그램과 자료를 처리하기 위해서는 개발자가 하드 디스크와 같은 보조 기억 장치와 연동하는 모듈을 직접 작성해야 하는 어려움이 있다.

메모리 용량의 제약을 극복하기 위하여 많은 방법들이 고안되었다. Swapping [1, 2]은 메모리 용량의 제약을 해결하기 위하여 메모리 상의 여분의 프로세스들을 보조 기억 장치로 옮기거나 보조 기억 장치에 있는 프로세스들을 다시 메모리로 옮겨와 수행을 재개하는 작업을 수행한다. 가상 주소(Virtual Address) [1, 2]도 Swapping 과 함께 메모리 용량의 제약을 극복하기 위하여 사용되는데, 가상 주소상에서 연속적인 주소의 메모리 영역들이 반드시 메모리 상에 물리적으로 연속된 메모리 영역일 필요는 없다. 또한, 가상 주소의 특정 메모리 영역은 메모리나 보조 기억 장치에 위치할 수 있어서, 사용 가능한 가상 메모리의 용량은 실제 메모리의 용량보다 클 수 있다. 따라서, 가상 주소와 Swapping 을 지원하는 최근의 운영체제는 실제 기기에 장착된 메모리보다 더 큰 프로그램과 자료를 가상 주소가 지원하는 범위 안에서 별도의 추가 작업 없이 쉽게 처리할 수 있다.

하드웨어 공정 기술의 발전 역시 메모리 용량의 제약을 극복하는데 크게 기여하고 있다. 메모리 공정 기술의 발전으로 대용량, 고성능, 저비용의 메모리들이 생산되고 있으며 이들은 많은 컴퓨터 기기에 장착되고 있다. 이렇게 증가된 메모리 용량은 컴퓨팅 연산 방식도 과거와 달리 고성능 연산을 위하여 대부분의 연산을 메모리 위에서 수행하는 계산 방식을 유행시키고 있다.

하지만, 이와 같이 증가한 메모리 용량을 최대한 활용하기 위해서는 프로세스(Process)가 사용할 수 있는 가상 주소의 크기를 확장할 필요가 있다. 전통적으로 운영체제는 기기에 장착된 메모리와 보조 기억 장치를 연동하여 좀 더 넓은 영역의 가상 메모리를 쉽게 쓸 수 있도록 지원해왔다. 하지만, 최근에 급격히 늘어난 메모리 용량은 프로세스가 사용할 수 있는 가상 주소 영역보다 그 용량이 더 커지고 있다. 가령,

PAE(Physical Address Extension)[5]를 지원하는 32-비트 프로세서나 32-비트 명령어를 같이 지원하는 64-비트 프로세서들의 경우, 프로세스의 가상 주소 공간보다 더 큰 용량의 메모리를 장착하여 사용할 수 있지만, 프로세스는 32-비트 가상 주소의 크기를 넘어서는 메모리를 사용할 수 없어서 장착된 메모리의 일부분 밖에 사용할 수 없는 일이 발생한다.

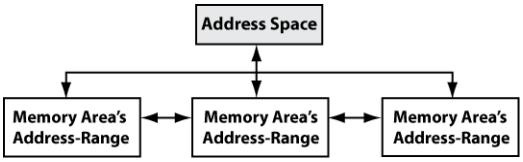
64-비트 프로세서들의 경우 프로세서들이 지정할 수 있는 Address 의 크기를 확장함으로써 프로세스의 가상 주소 공간 역시 같이 확장시키는 방안도 사용되고 있다. 하지만 64-비트 프로세서의 경우, 프로세서 명령어의 크기와 포인터 변수의 크기를 모두 증가시켜서 결과적으로 프로그램과 자료의 크기를 모두 증가시킨다는 단점이 있다. 64-비트 프로세서를 사용하는 방안은 메모리가 충분히 장착된 기기에서는 유용한 방법이지만, 모바일, 임베디드 기기와 같이 가격, 크기, 전력 등의 문제로 상대적으로 제한된 용량의 메모리를 장착한 기기에서는 심각한 문제가 될 수 있다.

본 논문에서는 현재 일반적으로 사용되는 프로세스의 Linear Virtual Address 의 크기를 확장하여, 프로세스가 보다 넓은 가상 메모리 영역을 쉽게 사용할 수 있는 방안을 제안한다. 최근 소프트웨어는 대용량의 자료를 처리할 것을 요구 받고 있어서 더욱 넓은 메모리 영역에 쉽게 접근해서 사용할 수 있는 방안이 필요하다. 가령, 특정 프로세스가 32-비트 크기의 가상 주소를 사용하는 경우, 이를 확장하여 32-비트 이상의 가상 메모리 영역을 사용할 수 있는 방안을 제공함으로써 기존의 명령어나 포인터 변수의 크기를 늘리지 않고도 더 많은 가상 메모리 영역을 사용할 수 있는 방안이 필요하다. 본 논문에서 제안하는 방안은 PAE 등을 지원하는 32-비트 프로세서를 장착한 기기나 32-비트 명령어를 수행하는 64-비트 프로세서를 장착한 기기 등에 효율적으로 적용할 수 있을 것으로 기대한다.

2. 예비 지식

최근의 운영 체제들은 가상 주소(Virtual Address)를 사용한다 [1,2]. 가상 주소를 사용하도록 설정한 경우, 프로세서는 프로그램에 사용되는 주소를 가상 주소로 인식하며, 프로그램 수행 시에 가상 주소를 물리 주소(Physical Address)로 변환한 후, 물리 주소를 이용하여 메모리에 접근, 연산을 수행한다. 프로세서는 MMU(Memory Management Unit)와 같은 하드웨어 모듈을 이용하여 가상 주소를 물리 주소로 빠르게 변환하며, 이 과정에서 페이지 테이블(Page Table)이라는 변환 테이블을 사용한다. 프로세스마다 해당하는 페이지 테이블이 있으며, 페이지 테이블에는 해당 프로세서의 가상 주소와 물리 주소간에 Address-Mapping 정보를 담고 있다.

최근의 운영 체제는 프로세스에 단 하나의 Linear 한 가상 주소 공간을 제공한다. 프로세스들은 서로 독립적인 가상 주소 공간을 가져서, 서로 다른 프로세스에서의 동일한 가상 주소는 일반적으로 서로 다른 메모리 위치를 가리킨다. 최신 운영 체제는 임의의 프로세스의 가상 주소 공간을 (그림 1)과 같이 관리한다 [3,4]. 최근의 운영 체제는 프로세스의 접근 가능한 가상 메모리 영역에 대한 가상 주소 범위(Address Range)를 리스트 형태로 관리한다. 이러한 가상 주소 공간은 프로세스당 하나씩 제공한다. 리스트의 각 접근 가능한 메모리 영역의 주소 범위는 서로 겹치지 않으며, Linear 형태의 가상 주소 공간을 제공한다.



(그림 1) 프로세스의 가상 주소 공간 [3,4]

최근 운영 체제들은 멀티 태스킹(Multi-Tasking) 방식을 지원한다. 이는 여러 프로세스들이 개념적으로 또는 실제로 동시에(Concurrently) 수행될 수 있도록 지원한다. 이 경우 한 프로세스에서 다른 프로세스로 문맥 전환(Context-Switching)이 발생하는 경우, 이전의 프로세스에서 새로운 프로세스로 페이지 테이블 교체를 포함한 적절한 문맥 전환 작업이 이루어져야 한다. 하지만, 프로세스 문맥 전환시마다 이전의 프로세스에 해당하는 문맥 정보를 모두 저장하고 새로운 프로세스에 해당하는 문맥 정보를 복원하는 작업은 상당한 Overhead 가 된다.

최근 운영 체제들은 쓰레드(Thread)라는 경량(Light-weight) 프로세스를 지원한다. 하나의 프로세스에서 생성된 쓰레드들은 같은 가상 주소 공간을 공유하지만, 각 쓰레드의 프로그램 수행 위치는 모두 다를 수 있다. 쓰레드간 문맥 전환 과정은 프로세스간 전환 과정과 비교하여 문맥 정보 내용이 단순하여 문맥 전환 과정이 훨씬 더 효율적이지만, 쓰레드간 별도의 프로그램 수행 위치를 추적해야 하는 부담이 발생한다. 쓰레드는 프로세스와 달리 동일한 프로세스의 가상 주소 공간을 공유하여서 쓰레드간 자료의 전송 및 공유가 프로세스간 자료의 전송 및 공유와 비교하여 훨씬 효율적이다.

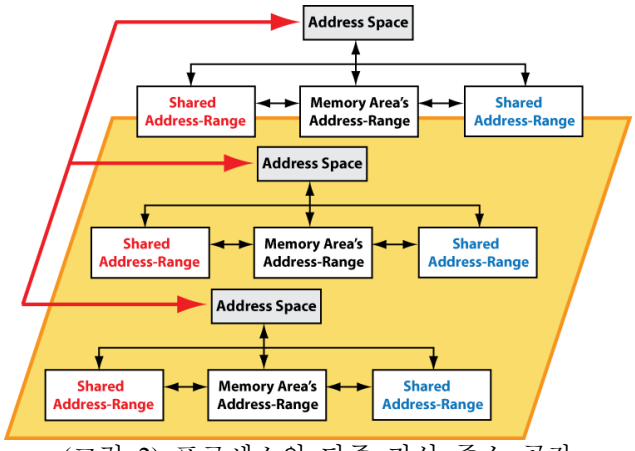
PAE (Physical Address Extension) [5]는 32-비트 프로세서가 4G 바이트 이상의 메모리를 사용할 수 있도록 지원해준다. 현재 인텔 및 AMD 의 프로세서들이 PAE 를 지원하고 있다. 프로세서는 추가적인 Address Line 을 제공하여 물리 주소 크기를 32 비트에서 36 비트로 확장시켜준다. PAE 를 통해서 메모리 크기는 이론적으

로 최대 4G 바이트에서 64G 바이트까지 확장 가능하다. 하지만 32-비트의 가상 주소 크기는 변하지 않는다. 따라서, PAE 를 지원하는 운영체제는 64G 바이트 메모리 안에서 여러 프로세서들을 효율적으로 수행할 수 있으나, 프로세스 입장에서는 이전처럼 4G 바이트의 가상 주소 공간 안에서 수행되며 전체 메모리의 일부분 밖에 사용할 수 없다는 단점이 있다.

3. 다중 가상 주소 공간 지원 프로세스

프로세스(Process)의 사용 가능한 가상 주소 공간을 확장하기 위하여, 우리는 운영 체제가 프로세스에 제공하는 하나의 가상 주소 공간(Virtual Address Space)을 하나가 아닌 여러 개의 가상 주소 공간으로 확장하고, 원하는 주소 공간을 선택할 수 있는 수단을 제공할 것을 제안한다.

구체적으로, 우리가 제안하는 운영 체제는 아래와 같은 기능을 제공한다. (1) 운영 체제는 프로세스가 가진 Linear 주소 공간 중 어떤 가상 주소 영역을 몇 개의 다중 주소 공간으로 만들 것인지에 대한 사용자 API (또는 System Call)을 제공한다. (2) 운영 체제는 이미 만들어진 다중 주소 공간 중 어떤 주소 공간을 사용할 것인지 선택할 수 있는 사용자 API(또는 System Call)을 제공한다. (3) 운영 체제는 기존에 만들어진 다중 주소 공간을 제거할 수 있는 사용자 API (또는 System Call)을 제공한다.



(그림 2) 프로세스의 다중 가상 주소 공간

우리가 제안하는 다중 가상 주소 공간을 갖는 프로세서의 구현은 (그림 2)와 같다. 전통적인 운영체제는 (그림 1)과 같이 프로세스의 가상 주소 공간을 접근 가능한 가상 주소 범위의 리스트(List)로 관리하며, 전통적으로 프로세서의 가상 주소 공간은 하나이므로 프로세스 당 단 하나의 리스트가 존재한다. 이와는 달리, 우리는 여러 개의 가상 주소 공간을 가질 수 있는 프로세스를 (그림 2)와 같이 제안한다. 한 프로세스는 접근 가능한 가상 주소 범위의 리스트를 여러 개 가질 수 있으며, 이는 운영체제가 제공하는 API 를 통하여 어떤 가상 주소 영역에 대하여 몇 개의 리스트를 가질 것인지를 지정할 수 있도록 한다. (그림 2)의 경우 검은색으로 표시된 가상 주소 영역에 대하여 총 3 개의 가상 주소 공간을 갖는 프로세스를 보여준다. 프로세스는 총 3 개의 가상 주소 공간 중 하나를 선택하여 사용할 수 있으며, 필요한 경우 이를 전환하여 사용할 수 있다. 검은색 부분의 가상 주소 영역은 가상 주소 공간이 다르다면 이 영역의 가상 주소 값이 같다고 하더라도 다른 메모리 영역을 나타낸다. 반면에, 빨간색과 파란색으로 표시된 가상 주소 공간은 공유된 영역으로, 가상 주소 값이 같다면 가상 주소 공간에 상관없이 모두 동일한 메모리 영역을 나타난다.

프로세스의 다중 가상 주소 공간의 전환 과정은 프로세스 전환과 비교하여 다음과 같은 차이점이 있다. 다중 주소 공간 중 하나를 선택할 때는 프로세스 전환시와 유사하게 적절한 페이지 테이블의 전환 작업이 필요하다. 하지만, 다중 주소 공간 중 하나를 선택하는 과정은 프로세스간 전환시와는 달리 페이지 테이블 교체 이외의 추가적인 작업이 필요하지 않아서 훨씬 효율적으로 진행될 수 있다.

또한, 프로세스의 다중 가상 주소 공간의 전환 과정은 쓰레드 전환시와 비교하여 다음과 같은 차이점이 있다. 다중 주소 공간 중 하나를 선택하는 것은 쓰레드 전환과 달리 페이지 테이블을 변경한다. 반면에, 쓰레드는 각각의 프로그램 수행 위치를 관리해야 하지만, 다중 주소 공간을 선택할 때는 이러한 오버헤드는 발생하지 않는다는 장점이 있다.

우리가 제안하는 프로세스상에서 수행되는 프로그램은 필요한 경우 운영 체제가 제공하는 사용자 API 를 통하여 특정 주소 영역을 다중 주소 공간으로 설정한다. 이렇게 설정된 주소 공간에 대해서 운영 체제가 제공하는 사용자 API 를 사용하여 원하는 주소 공간을 선택하여 메모리 연산을 수행할 수 있다. 이때 다중 주소 공간으로 설정된 가상 메모리 영역은 기존의 단일의 Linear 가상 주소에 비하여 그 크기가 확장된 주소 공간으로 기본보다 더욱 넓은 영역의 가상 주소 공간을 사용할 수 있다. 프로그램은 다중 주소 공간을 모두 사용한 후 운영 체제에서 제공하는 사용자 API 를 사용하여 다중 주소 공간을 해제할 수 있다.

4. 결론

우리는 기존의 운영 체제에서 제공하던 프로세스의 Linear 가상 주소의 크기를 확장하여 프로세서들의 더욱 넓은 영역의 가상 메모리 크기를 사용할 수 있는 방안을 제안했다. 우리의 다중 가상 주소 공간을 지원하는 프로세스는 기존의 운영 체제에서 제공하던 프로세스의 Linear 가상 주소의 크기를 확장함으로써 프로세스들이 더욱 넓은 영역의 가상 메모리를 쉽게

사용할 수 있도록 해준다. 본 논문은 프로세서의 Address 사이즈를 증가시키는 경우와 달리, 프로세서의 명령어 크기를 증가시키지 않으며, 포인터 변수의 크기를 증가시키지도 않는 장점이 있어서, 모바일 및 임베디드 기기와 같이 상대적으로 제한된 용량의 메모리를 지원하는 시스템에 매우 유용할 것으로 기대한다.

참고문헌

[1] Andrew S. Tanenbaum, "Modern operating system," 2nd edition, Prentice-Hall, 2011.

[2] Uresh Vahalia, "UNIX internals: the new frontiers," Prentice-Hall, 1996.

[3] Robert Love, "Linux kernel development," 3rd edition, Addison-Wesley, 2010.

[4] Daniel P. Bovet and Marco Cesati, "Understanding the Linux kernel," 3rd edition, O'Reilly, November, 2005.

[5] PAE(Physical Address Extension), http://en.wikipedia.org/wiki/Physical_Address_Extension