WAS(wide address system)에서의 주소 공간 설계

김일민 한성대학교 정보 전산학부
박재희 계명대학교 컴퓨터 공학부

Address Space Design in Wide Address Space Systems

IlMin Kim HanSung Univ.
JaeHee Park Keimyung Univ.

요약

새로운 DEC Alpha, MINPS R4000[1], PowerPC등과 같은 64-bit 마이크로프로세서는 운영체제와 응용프로그램에 매우 광활한 64-bit 주소공간(wide address space)을 제공한다. 64-bit 주소공간은 중소규모 분산 컴퓨터 시스템의 모든 데이터를 포함할 수 있는 크기이다. 이 64-bit 주소공간은 기존의 32-bit 주소공간과 다른 방법으로 활용하는 것을 가능하게 해주었다. 지금까지의 시스템과는 달리 WAS(wide address system)에서는 모든 프로세스들이 하나의 주소공간을 공유함으로서 프로세스간 자료의 공유 및 통신이 간편하게 이루어 질 수 있다. 공유된 광활한 64-bit 주소공간의 사용방안은 WAS 시스템 연구에서 매우 중요하다. 본 논문에서는 WAS 시스템의 보다 구현하기 쉬운 64-bit 주소공간의 설계에 대해서 제안한다.

1. 서론

VLSI설계 및 구현기술의 발달에 따라 RISC 프로세서의 제작회사들은 64-bit 데이터와 주소공간을 지원하는 DEC Alpha, HP PA-RISC등의 마이크로 프로세서를 제작하기 시작하였다. 64-bit 주소공간은 매초 100 Mbyte의 공간을 소비하더라도, 5000년 동안을 사용할 수 있는 매우 광활한 공간이다[2]. 그러므로 64-bit 시스템에서 기존의 시스템과 같이 각 프로세스마다 64-bit 주소공간을 할당하는 것은 불필요하다.

UNIX와 같은 기존의 운영체제는 보호 도메인(protection domain)과 주소공간이 동일하게 간주되었다. 즉 모든 프로세스는 자신의 주소공간을 가지며, 자신의 주소공간은 다른 프로세스들로부터 보호된다. 모든 프로세스는 자신의 주소공간만을 액세스할 수 있다. 이러한 분리된 프로세스 공간은 프로세스간의 데이터 공유를 어렵게 만들었다. 이와는 달리, WAS 시스템의 모든 프로세스들은 논리적으로 하나의 주소공간을 공유한다. WAS 시스템의 주소공간은 시스템의 분산환경을 감춘다. 즉 WAS 시스템의 모든 분산자원은 주소공간에 사상되어, 응용프로그램은 분산자원의 물리적 위치에 관계없이 자원의 주소로 접근할 수 있다.

WAS의 주소공간은 사용자의 정의에 따라 persistent 속성을 지닐 수 있다. UNIX와 같은 운영체제에서 프로세스가 다시 실행될 때 사용할 모든 데이터는 디스크에 저장되어야한다. WAS 시스템에서 persistent 주소공간의 데이터는 소멸되지 않는다. 즉 WAS 시스템의 프로그래머는 더 이상 자료를 파일에 저장하거나 읽어 올 필요가 없다. 파일 입출력 프로그래밍은 전체 코딩의 20-30%를 차지하는 작업이다.

WAS 시스템의 메모리 계층구조는 기존의 시스템보다 1단계를 더 가진다. 기존 시스템에서 페이지 폴트가 발생하였을 시, 그 페이지는 로컬 하드

디스크에서 적재할 수 있지만, WAS 시스템의 경우, 폴트된 페이지가 로컬 디스크에 있을 수도, 리모트(remote) 디스크에 있을 수도 있기 때문이다. WAS 시스템의 메모리 계층구조를 그리면, [그림 1]과 같다.

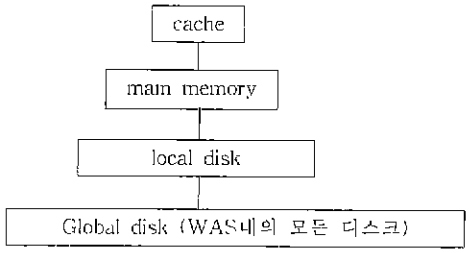

그림 1. WAS 시스템의 메모리 계층구조

기존의 운영체제와 WAS 시스템간의 여러 특징에 관한 비교 연구는 [2], [3], [4] 에 자세히 기술되어 있다.

2. WAS 시스템의 새로운 문제점

WAS 시스템은 기존의 운영체제와 비교하여 많은 장점이 있지만, 기존 시스템에는 없었던 새로운 문제점을 발생시킨다. 첫 번째 문제점은 기존 운영체제와 호환성문제이다. 새로운 운영체제를 개발할 경우, 기존 기계어 소프트웨어가 어떠한 수정없이 실행되는 binary 호환성을 제공하여야 한다. WAS 시스템은 주소공간을 사용하는 방식이 기존 시스템과 근본적으로 다르므로, binary 호환성을 제공하지 않는다. UNIX의 fork()명령어는 WAS 시스템에서는 흉내낼 수 없다[3]. 두 번째, WAS의 메모리 시스템은 더욱 복잡한 일을 처리해야한다. 광대한 주소공간관리 및 리모트 페이지 폴트를 처리해야한다. 동일한 가상페이지는 여러 물리적 페이지를 가질 수 있으므로, 일관성(coherence) 및 동기화(synchronization) 문제를 처리해야한다. 특히 WAS의 주소공간이 매우 넓기 때문에, 참조되지 않는 자료 객체인 쓰레기(garbage)의 처리는 더욱 어렵다.

3. WAS 주소공간의 설계

WAS에서 주소공간은 프로세스가 액세스하는 유일한 자원이다. 파일, 프린터등의 다른 자원들은 그 물리적인 위치에 관계없이 공유된 주소공간에 사상된다. 주소공간의 페이지는 하나이상의 물리적인 복사본을 가질 수 있다. 분산 프로세스들은 동일한 페이지를 로칼 디스크/메모리로부터 액세스 할 수 있으며, 이는 시스템의 병렬성을 높여준다.

프로세스가 새로운 객체를 생성할 때, WAS는 어떻게 공간을 할당할 것인가? 기존의 시스템과는 달리 주소공간이 공유되었기 때문에 WAS에서는 상황이 다소 복잡하다. 가장 단순한 방법은 관리자 노드를 선정하고, 주소공간할당이 발생할 때마다, 메시지를 전송하는 것이다. 이는 WAS 시스템의 확장성에 심각한 악영향을 준다. 그러므로, WAS의 주소공간할당 오버헤드를 최소화하기 위하여 다음과 두 가지 원칙을 제안한다. 1) WAS의 어느 노드도 64-bit 주소공간에 전체에 대한 정보를 관리/저장하지 않는다. 2) WAS의 각 노드는 로컬정보만을 가지고 주소공간을 할당한다. 즉 주소공간할당 시에 메시지를 유발하지 않는다.

가상공간을 균일한 크기(52bit 내외)로 분할하여, 각 분할을 하나의 노드에 적재하는 것이다. 한 노드에 하나이상의 분할을 적재할 수 있다. 로컬 프로세스에 대한 메모리할당은 자신에게 적재된 분할에서 이루어진다. 하나의 분할은 여러 도메인으로 분리되며, 하나의 분할은 한 사용자에게 독점된다. 한 사용자가 실행한 프로세스는 64-bit 주소공간 전체를 액세스할 수 있지만, 메모리 할당요구는 자신의 도메인 안에서만 이루어진다. 별도의 액세스 제어 메카니즘이 있어야하지만, 기본적으로 프로세스가 자신의 도메인에 대한 액세스는 허용되고, 도메인 외부에 대한 액세스는 불허된다. 각 도메인은 시스템사용데이터와 사용자 객체로 이루어진다. [그림 2]

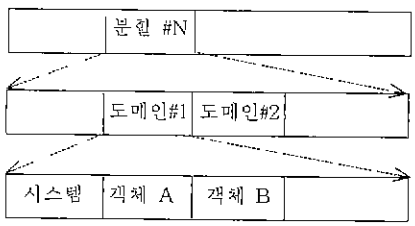

그림 2. WAS의 주소 공간

4. WAS 객체(object)의 구조

WAS에서 객체는 공유 및 액세스를 제어의 기본단위이다[5]. 각 객체는 연관된 ACL (access control list)를 가질 수 있다. 만일 연관된 ACL이 없으면, 프로세스와 객체가 동일한 도메인인 경우에만 액세스를 허용한다. 여기서 우리는 객체를 단순객체와 캡슐화 객체 두 가지 유형으로 구분한다. 단순객체는 하나이상의 연속된 페이지로 구성된 데이터 스트림을 의미한다. 캡슐화 객체는 프로그래밍언어의 추상화 데이터형과 비슷한 의미이다. 즉 캡슐화 객체는 실행 코드와 데이터를 포함하는 실행 모듈이다. WAS에서 여러 도메인간의 객체의 공유에 대해서 생각해보자. 객체의 코드는 공유 가능한 부분이고, 객체의 데이

터는 공유 가능할 수도 사적인 부분일 수도 있다. WAS에서 캡슐화 객체의 데이터부분은 다음 세 가지 유형으로 구분한다.

① Private-volatile 데이터: 객체가 호출될 때 할당되고, 실행이 끝날 때 반환되는 데이터이다. 한 쓰레드에 종속되는 데이터로서 쓰레드간 공유되지 않는다.

② Private-persistent 데이터: 객체가 실행 종료되더라도 반환되지 않고, 그 내용이 유지된다. 한 도메인에 종속되는 데이터이며, 다음 객체 호출시에 사용할 수 있다.

③ Shared-persistent 데이터: 객체를 호출할 수 있는 모든 도메인이 공유하는 데이터이다. 한 객체에 종속되어, 객체를 호출할 수 있는 모든 도메인이 공유한다.

WAS에서 캡슐화 객체의 주소공간 사용방법은 기존의 시스템과는 매우 다르다. 캡슐화 객체가 공유되는 방식을 WAS 및 기존 운영체제와 비교하면 다음과 같다. 기존의 운영체제는 공유된 객체의 코드와 데이터를 프로세스의 사적인 주소공간에 사상시킨다. 실행 코드는 공유데이터와 private 데이터를 구분없이 액세스할 수 있다. [그림 3, 4.]

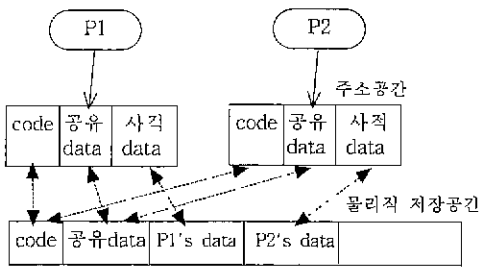

그림 3. 기존 시스템에서 객체공유

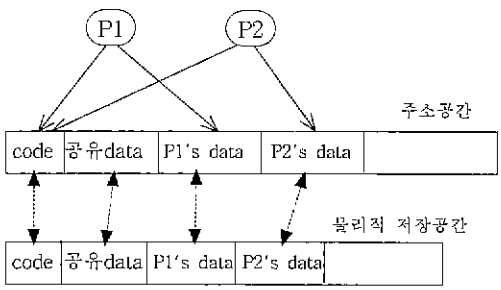

그림 4. WAS에서의 객체공유

그림 3, 4에서 볼 수 있듯이, 기존 32-bit시스템에서는 운영체제가 공유 가능한 코드와 private 데이터를 분리하여 개인 주소공간에 사상시킨다. WAS에서 실행코드는 private 데이터는 사용자에 따라 달라야한다. WAS에서 물리적 공간과 주소공간의 사상은 기존의 시스템에 비해 단순하며, 쓰레드가 자신의 private 데이터를 구분해서 액세스한다.

WAS에서는 객체의 식별자로서 객체의 64-bit 주소를 사용한다. 사용자들이 직접 운영체제에서 사용하는 64-bit 주소를 객체의 식별자로 사용하는 것은 매우 불편하다. WAS에서는 사용자 수준의 계층구조를 지닌 객체이름을 64-bit 주소로 변환하는 이름 서비스를 제공하여야 한다.

5. 결론

WAS는 기존의 32-bit 운영체제와 비교하여, 주소공간의 사용방법이 획기적이다. 객체들은 주소를 사용하여, 객체의 위치에 무관하게 액세스할 수 있으며, 자료의 공유도 주소를 넘겨줌으로서 해결된다. 그러나 WAS의 메모리 시스템의 설계 및 구현은 많은 문제점이 숨어있다. 많은 WAS 관련 논문들은 WAS의 장점만을 강조한 나머지 설계 및 구현에 숨어있는 문제점을 간과하고 있다. 본 연구에서 제안된 주소공간 설계와 객체구조는 WAS 시스템을 설계하고 구현하는 데 많은 참고가 될 것이다.

참고문헌

[1] MIPS Computer Systems, Inc., "MIPS R4000 Microprocessor User's Manual", Sunnyvale, 1994

[2] G. Heiser, K. Elphinstone, S. Russel and G. Hellestrand, "A Distributed Single Address Space Operating system supporting persistence", Univ. of New South Wales, TR-9314, March, 1993

[3] J. Chase, H. M. Levy, M. Baker-Harvery, and E. d. Lazowska "OPAL: A single address space system for 64-bit architectures." In proceedings of the third Workshop on Workstation Operating systems, P80-85, Key Biscayne, Florida, 1992

[4] Jeffrey S. Chase, "An operating system Structure for Wide-Address Architectures", PhD's thesis, Dept. of Computer Science, U. of Washington, Aug. 1995

[5] J. Rosenberg & J. L. Keedy, "Object management in the MONADS", Int' Workshop on Persistent Object system. volume2, Scotland, IEEE, 1987