몬드리안 메모리 프로텍션의 구현

감근진[0] 이인환
한양대학교 전자통신컴퓨터공학과
kjkam[0]@csl.hanyang.ac.kr ihlee@hanyang.ac.kr

An Implementation of Mondriaan Memory Protection

Keunjin Kam[0] Inhwan Lee
Dept. of Electronics and Computer Engineering, Hanyang University

요 약

몬드리안 메모리 프로텍션(Mondriaan Memory Protection)은 워드 단위까지 접근 권한 설정을 제공하는 메모리 보호 개념으로, 프로세서에서 요청하는 주소에 대하여 올바른 접근 권한을 가지고 있는지 검사한다. 기존 프로세서의 명령어 셋에 대한 추가 또는 변경과 프로그램 소스의 변경이 필요 없으면서도, 프로그래머 또는 사용자에 의해서 생길 수 있는 잘못된 메모리 접근을 원천적인 방법으로 방지할 수 있다. 본 논문에서는 몬드리안 메모리 프로텍션을 마이크로프로세서를 이용한 실제 하드웨어로 구현 하면서 발생 되는 고려 사항과 문제점에 대하여 짚어보고, 그 해결책을 제시하였다.

1. 서 론

에니악이 발명된 이래 지난 60년간 컴퓨터의 하드웨어는 매우 빠른 속도로 발전하고 성장해오고 있다. 더불어 소프트웨어도 양적, 질적으로 매우 빠른 성장을 거듭해 오고 있다. 이러한 성장은 복잡도의 증가를 가져오게 되어, 그에 따라 개발자와 테스터들도 미쳐 잡아 낼 수 없는 많은 버그들을 만들어내고 있다. 그 중에서 가장 많은 비중을 차지하는 것은 잘못된 메모리 접근이다. 잘못된 메모리 접근에 대하여 보호 받지 못하는 메모리는 버퍼 오버플로우나 잘못된 포인터 주소, 의도적이지 않은 코드 실행과 같은 문제점들이 발생한다. 이러한 문제점들은 작게는 데이터의 손실에서부터 크게는 비행기 추락이나 의료사고와 같은 결과를 만들어낸다[1].

가상 메모리 기법은 가장 흔하게 쓰이는 메모리 관리 방법 중에 하나이다. 메모리 매니지먼트 유닛(MMU)을 이용하여, 프로그램에서 사용하고 있는 일부분을 메모리에 올려 사용하는 방법으로 흔히 알려져 있기도 하지만, 메모리 보호 매커니즘도 한 부분으로 제공하고 있다. Intel의 x86 아키텍처에서 사용하는 MMU를 예로 들면, CR3 레지스터가 가리키고 있는 물리 메모리상의 주소에, 가상 주소에서 물리 주소로의 변환을 위한 페이지 테이블이 있고, 이 테이블의 엔트리에 주소 변환을 위한 페이지 베이스 주소와 각종 정보들이 기록되어 있다. 정보 중에는 해당 페이지의 접근 권한에 대한 정보도 기록되어 있기 때문에, MMU에서는 접근 권한을 위반하는 접근이 있을 때, 페이지 폴트를 발생시킨다[4]. 그렇지만, 매커니즘의 한계로 인하여 페이지 크기 까지만(보통 4kB) 권한 설정을 할 수 있다는 단점이 있다. 따라서 더욱 작은 크기에 대하여 접근 보호를 하기 위한 상황에서는 페이지 안의 나머지 공간을 낭비할 수밖에 없다.

몬드리안 메모리 프로텍션(Mondriaan Memory Protection, MMP)[3]은 워드 단위(4 바이트)까지의 접근 권한 설정을 제공하는 메모리 보호 개념이다. 가상 메모리 기법에서 제공하는 접근 보호 방법과는 별도로 전용 하드웨어를 사용하여, 프로세서에서 요청하는 주소에 대하여 접근 권한을 가지고 있는지 검사한다. 기존 프로세서의 명령어 셋의 추가 또는 변경과 프로그램 소스의 변경이 필요 없으면서도, 프로그래머 또는 사용자에 의해서 생길 수 있는 잘못된 메모리 접근을 하드웨어적인 검사를 통하여 원천적인 방법으로 방지한다.

2. 몬드리안 메모리 프로텍션

몬드리안 메모리 프로텍션은 Butler Lampson이 제안한 프로텍션 도메인(Protection domain)이라는 개념을 사용한다[2]. 하나의 프로세스 메모리 공간에 대해서 여러 개의 프로텍션 도메인을 만들 수 있으며, 실행 쓰레드는 반드시 하나의 프로텍션 도메인을 통하여 프로세스의 메모리 공간에 접근을 한다. 그림 1은 MMP의 개념을 이용하여, 쓰레드와 프로텍션 도메인, 주소 공간과의 관계를 예로 든 것이다.

쓰레드는 자신과 결합된 프로텍션 도메인에서의 권한 설정에 따라서, 해당 주소에 대한 접근을 할 수 있다. 그림 1

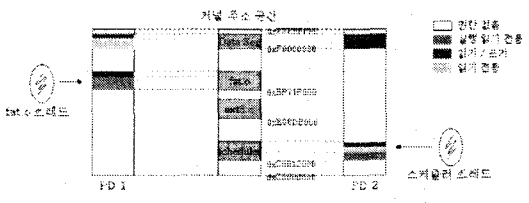

그림 1 주소 공간과 프로텍션 도메인

표 1 퍼미션 테이블에 저장되는 값과 의미

권한 값	의미
00	권한 없음
01	읽기 전용
10	읽기 / 쓰기
11	실행 읽기 전용

의 상황을 예로 들면, fat.o 쓰레드는 프로텍션 도메인 1과 결합된 상황에서 0xF0000000 번지에 대해서는 읽기만 가능하고, 쓰기를 할 경우에는 MMP에서 퍼미션 폴트(Permission fault)를 발생시킨다. 반대로 스케쥴러 쓰레드는 프로텍션 도메인 2와 결합이 되었을 때, 같은 0xF0000000 번지에 대해서 읽기와 쓰기 모두 가능하다. 이와 같이, 같은 주소 공간에 대해서도 프로텍션 도메인을 어떻게 설정하는지에 따라서 쓰레드 마다 접근 권한을 다르게 줄 수가 있다.

MMP는 하드웨어 부분과 소프트웨어 부분으로 구성된다. MMP 하드웨어는 명령 반입와 데이터 읽기, 쓰기 때마다 접근 권한을 퍼미션 테이블(Permission table)로부터 찾아서 올바른 접근을 하는지 검사를 한다. MMU가 가상 주소를 물리 주소로 변환하는 동안, 동시에 가상 주소에 대하여 접근 권한 검사를 하고, 최종적으로 메모리 접근에 대해서 승인 또는 비승인을 한다. MMP 하드웨어의 구조는 기본적으로 그림 2와 같이 구성되어 있다.

퍼미션 테이블은 접근 권한의 정보들을 가지고 있는 자료구조이고, 프로텍션 도메인마다 생성이 되어 할당이 된다. MMP 하드웨어는 프로세서가 접근을 요구하는 주소를 참조로 하여, 퍼미션 테이블에서 해당 주소에 대한 권한을 찾아, 접근에 대한 승인 또는 비승인을 처리한다. 퍼미션 테이블에는 표 1과 같이 2 비트로 구성된 값들이 접근 권한으로 저장된다.

퍼미션 룩어사이드 버퍼(Permission Lookaside Buffer, PLB)는 최근 사용한 권한 정보들을 저장해 두어, MMP 하드웨어가 권한 검사 때마다 메모리의 퍼미션 테이블에 접근하는 시간을 감소 시켜준다. 사이드카 레지스터(Sidecars register)는 프로세서에 있는 모든 주소 레지스터와 쌍을 이루고, 가장 최근에 사용한 세그먼트에 대한 권한 정보를 가지고 있어, MMP 하드웨어가 PLB에 접근하는 시간을 줄여준다. 또한 사이드카 레지스터는 세그먼트의 정보를 2의 멱승의 크기에 맞추어서 저장하고 있는 PLB와

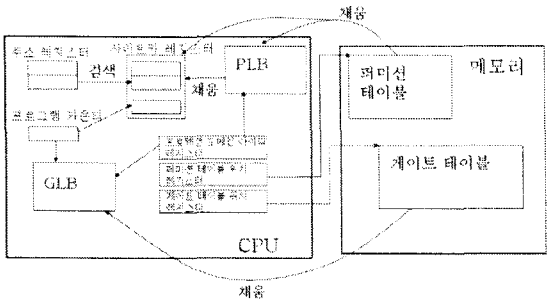

그림 2 MMP 하드웨어의 구조[6]

는 다르게, 기본 주소와 세그먼트의 크기의 형태로 정보를 가지고 있기 때문에 PLB 미스를 줄여준다.

MMP는 일반적인 접근 권한 검사의 기능뿐만이 아니라, 프로시저나 함수 호출에 대한 보호 기능도 제공한다. 악의적이거나 실수에 의한 실행을 방지하기 위하여, 진입 권한이 설정되어 있는 곳에서만 프로시저나 함수에 진입을 할 수가 있다. 이 기능을 제공하기 위하여 MMP는 게이트 테이블(Gate table)이라고 불리는 진입점(Entry point)이 저장된 테이블을 사용하고, 매번 콜이나 리턴 명령이 반입 될 때, 게이트 테이블을 검색하여 진입이 허용된 위치인지 검사한다. 이 때, 진입이 허용된 위치라면 크로스 도메인 콜링(Cross domain calling)이라고 불리는 프로텍션 도메인의 환경 전환을 실행한다. 또한 MMP는 매 검사 때마다 메모리의 게이트 테이블에 접근하는 오버헤드를 줄이기 위하여, PLB의 기능과 마찬가지로 게이트 룩어사이드 버퍼(Gate Lookaside Buffer, GLB)라는 캐시를 가진다.

Mondriaan Memory Protection의 소프트웨어 부분인 MMP 수퍼바이저(MMP Supervisor)는 각종 API를 사용자나 OS에 제공하여 MMP를 관리할 수 있게 해준다. API를 이용하여 MMP 하드웨어를 제어하고, 프로텍션 도메인의 생성 및 삭제, 퍼미션 테이블과 게이트 테이블의 관리, 퍼미션 테이블 내의 권한 값 변경, 게이트 테이블의 진입점의 설정과 같은 작업을 할 수 있다. MMP 수퍼바이저는 여러 프로텍션 도메인의 관리에 있어서 충돌을 방지하기 위하여 MMP 하드웨어에서 이용하는 정보들과는 개별적으로 프로텍션 도메인에 대한 정보와 정책을 가지고 있다.

3. 구현 이슈

3장에서는 MMP를 실제로 구현할 때 나타나는 고려 사항과 문제점에 대하여 언급하고 대안 및 해결책을 제시한다.

3.1 퍼미션 벡터 테이블(Permission vector table)

퍼미션 벡터 테이블은 트라이(Trie) 구조를 이용한 다중 레벨 테이블이다. 프로세서가 접근하려는 주소를 인덱스로 사용하여 테이블을 검색하고, 권한 값이 저장되어 있는 벡터 또는 서브 블록(Sub block)에 접근하여 해당 주소에 대한 권한 값을 찾는다. 테이블의 엔트리는 일반적으로는 하위 레벨의 테이블이나 벡터의 시작 주소를 가지고 있지만, 하위 레벨 테이블에서 담당하고 있는 주소 영역에 대한 권한 값을 8등분으로 나눌 수 있을 때, 사용 공간을 줄이기 위하여 서브 블록을 가지고 있을 수 있다. 따라서 하위 레벨 테이블에 대한 공간의 할당이 필요 없이 권한 값을 설정할 수 있다. 퍼미션 벡터 테이블은 권한 값의 최대 검색 시간이 일정하다는 장점을 가지고 있지만, MMP 수퍼바이저가 테이블을 관리하기 위한 부가적인 오버헤드가 크다는 단점이 있다. 예를 들어, 서브 블록으로 권한 값의 압축을 위하여, 테이블 내용의 변경 때마다 하위 테이블 엔트리들의 서브 블록에 대한 검사를 한다. 따라서 매 번 중복되는 검사를 줄이기 위하여, 기존의 정보 외에도 한 엔트리의 서브 블록이 모두 같은 권한 값을 가졌는지에 대한 정보를 저장하는 공간을 할당해야 한다. 각각의 엔트리를 갱신할 때에는 엔트리의 서브 블록이 모두 같은 권한인지 아닌지에

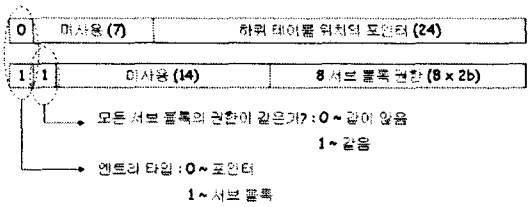

그림 3 서브 블록의 권한 상태 정보에 대한 공간이 할당된 퍼미션 벡터 테이블의 엔트리의 구조

대해서 저장을 하고, 전체적인 테이블을 검사할 때에 이 정보만 검사하게 되므로, 상대적으로 검사 오버헤드가 많이 줄어든다. 그림 3은 이를 위해 수정된 퍼미션 벡터 테이블의 엔트리 구조이다.

트라이 인덱스의 비트 할당에 대한 변경도 필요하다. 구현이 될 MMP 시스템에서 메인 프로세서는 최대 64킬로바이트의 외부 메모리를 지원한다. 또한 8 비트 환경이기 때문에 바이트 단위까지의 권한 설정을 지원해야 한다. 그림 4는 변경된 퍼미션 벡터 테이블의 트라이 인덱스이다. 기존의 MMP는 벡터 내의 권한 값의 개수가 16개이고 워드 단위까지 권한 설정을 지원하기 때문에 벡터 오프셋이 6 비트였지만, 구현될 MMP는 벡터내의 권한 개수는 같지만 바이트 단위까지 권한 설정을 지원하기 때문에 벡터 오프셋에 4 비트를 할당한다.

3.2 크로스 도메인 콜링

크로스 도메인 콜링은 진입점에 접근 하였을 때, MMP 하드웨어가 자동적으로 새로운 프로텍션 도메인의 환경으로 바꾸어주는 것을 말한다. 이때에는 프로텍션 도메인 아이디와 퍼미션 테이블 위치, 게이트 테이블 위치 레지스터를 갱신한다.

RISC 프로세서는 모든 명령이 일정한 크기로 되어 있기 때문에 연산자(Opcode)를 찾아서 명령의 종류를 알아내기 쉽다. 하지만 CISC 프로세서는 특별한 처리를 하지 않으면 어느 것이 연산자이고 피연산자(Operand)인지 판단할 수가 없다. 연산자를 알아내는 방법이 두 가지가 있다. 첫째, 프로세서의 코어를 변형시켜 콜이나 리턴 명령을 디코드하였을 때 MMP 하드웨어에게 신호를 주는 것이다. 따라서 MMP는 이 신호를 감지한 후, 진입점에 대한 검사를 실행한다. 하지만 이 방법은 코어의 변형이 필요하므로 MMP가 주변장치의 개념으로 추가된다고 가정하였을 때 불가능한 방법이다. 둘째, MMP 하드웨어는 프로세서의 명령 반입을 디어셈블리하여 연산자를 알아내는 것이다. 프로세서의 기동과 동시에 디어셈블리를 실행하여 연산자가 콜이나 리턴인지 검사한다. 모든 디어셈블리 기능이 필요 없이, 연산자에 따라서 연산자에 대한 검사를 건너뛰는 방법을 사용한다.

일반적인 프로세서는 명령을 실행할 때, 다음 명령을 반

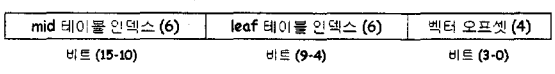

그림 4 비트 할당이 변경된 트라이 인덱스

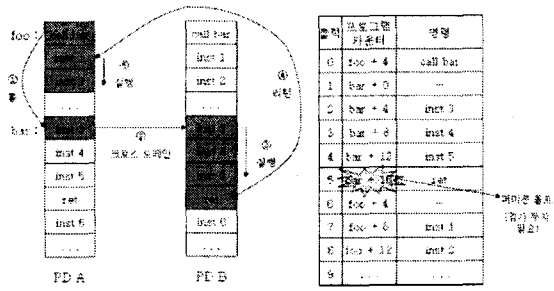

그림 5 크로스 도메인 콜링에서 권한 충돌 상황

입하게 된다. MMP의 권한 검사는 반입 과정에서 실행하기 때문에, 만약 콜이나 리턴 명령의 다음 주소의 권한 값이 실행 읽기 전용이 아니라면 퍼미션 폴트가 발생한다. 예를 들어 그림 5와 같은 상황을 보면, PD B에 속해 있는 'ret' 명령이 실행이 되는 순간에 프로세서는 다음 명령에 대해서 메모리로부터 반입을 한다. 'inst 6'은 PD B에 속하여 있지 않으므로, 비록 프로세서가 'inst 6'에 대하여 명령을 실행하지 않을 것이라도 MMP는 퍼미션 폴트를 발생시킨다. 따라서 크로스 도메인 콜링이 승인된 후에는 다음 명령의 반입에 대해서는 권한 검사를 무시하는 것이 필요하다.

3.3 프로텍션 도메인 정보 테이블

크로스 도메인 콜링을 할 때, 타입이 스위치이면, 게이트 테이블 엔트리에 있는 목적지 PD-ID(Destination PD-ID) 항목에 해당하는 프로텍션 도메인의 환경으로 전환이 된다. 프로텍션 도메인 환경의 전환은 프로텍션 도메인 아이디와 퍼미션 테이블 위치, 게이트 테이블 위치 레지스터의 내용을 프로텍션 도메인의 정보에 해당하는 값으로 갱신한다. 그러나 게이트 테이블의 엔트리나 크로스 도메인 콜 스택에는 프로텍션 도메인 아이디 값만 들어있기 때문에 퍼미션 테이블 위치와 게이트 테이블 위치 레지스터의 갱신을 위하여, 프로텍션 도메인에 대한 정보들을 알려주는 방법이 필요하다.

프로텍션 도메인 정보 테이블은 생성된 모든 프로텍션 도메인에 대한 퍼미션 테이블 위치와 게이트 테이블 위치의 값을 저장하고 있다. 빠른 접근을 위하여 정적으로 테이블의 공간을 할당한다. 또한, MMP 하드웨어는 프로텍션 도메인 정보 테이블의 위치를 가리키는 레지스터를 가지고 있어야 한다. 이 레지스터의 값은 MMP 수퍼바이저의 초기화 과정에서 프로텍션 도메인 정보 테이블에 대한 공간을 할당한 후, 기록이 되기 때문에 그 뒤로는 변경할 필요가

표 2 MMP 시스템의 제원

소 자	제 품	종 류
메인 프로세서	AT89C52	MCS-51(CISC), 8-bit
MMP 하드웨어	ATMEGA128	AVR(RISC), 8-bit
롬	AT29C256	플래시 타입
램	UM61M256	스태틱 타입
주소 디코더	GAL22V10D	PLD

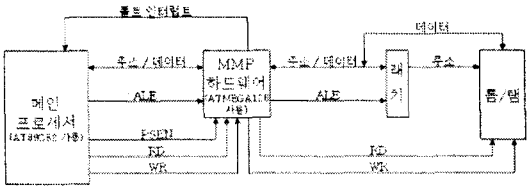

그림 6 MMP 시스템의 구성

없다. 크로스 도메인 콜링을 할 때에 목적지 PD-ID를 인덱스로 사용하여, 프로텍션 도메인 정보 테이블을 검색하고 해당 프로텍션 도메인에 대한 환경으로 전환을 한다.

4. 구현 결과

이번 장에서는 Emmett Witchel의 MMP 논문들[3, 5, 6]을 바탕으로, 3장에서 언급하였던 내용들을 종합적으로 적용하여 실제로 구현이 된 결과를 보여준다.

MMP의 기본 개념에 따르면, MMP 하드웨어는 프로세서의 코어와 연동이 되어야 한다. 사이드카 레지스터와 같은 것은 프로세서의 코어 내부에 있는 주소 레지스터와 결합이 되어야 하기 때문이다. 본 논문에서는 프로세서의 외부에 주변장치의 개념으로 MMP를 구현하였다. 따라서 MMP 하드웨어는 기존의 프로세서의 구조를 전혀 변경하지 않으면서, 프로세서에 대하여 메모리 접근 보호를 제공할 수 있다.

표 2는 MMP 시스템 구현에 사용된 주요 부품에 대한 리스트이고, 구현된 시스템은 그림 6과 같이 구성되어 있다. 또한 그림 7은 구현된 MMP 하드웨어의 기본적인 내부 구조이며, 그림 8은 실제 사진과 각 부품에 대한 설명이다.

5. 기능 평가

MMP가 제공하는 메모리 보호 기능은 일반적인 메모리 접근에 대한 권한 검사와 콜 또는 리턴에 대한 진입점 검사이다. 5장에서는 구현된 MMP에서 위의 두 가지 기능이 올바르게 동작하는지 확인한다.

5.1 접근 권한 검사

프로그래머가 가장 흔하게 실수하는 문제 중의 하나가 버퍼 오버플로우에 의한 변수 공간의 파괴이다. MMP를 이용

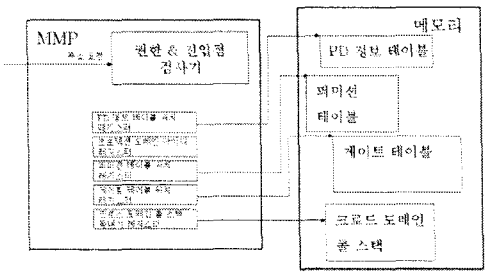

그림 7 구현된 MMP 하드웨어의 구조

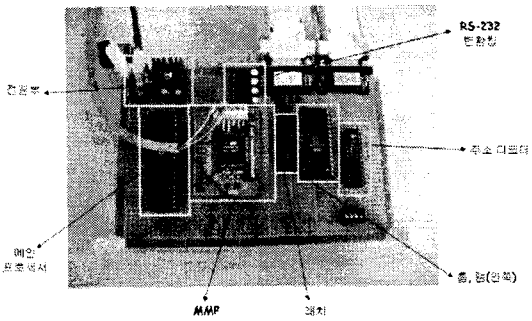

그림 8 구현된 MMP 하드웨어

하여, 버퍼 공간 이외에 접근 권한을 주지 않으면 오버플로우로 인하여 변수 공간이 파괴되는 상황을 방지한다. 그림 9는 접근 권한 검사에 쓰이는 예제 소스이다.

검사를 위해서 몇 가지 가정을 한다. 첫째, strcpy() 함수 자체에서는 버퍼의 경계 검사를 하지 않는다. 둘째, MMP를 사용하였을 때, buf[0]에서 buf[4] 변수 공간까지는 읽기/쓰기 권한을, cnt 변수 공간에는 읽기 전용, 이외에 공간에는 아무런 권한을 주지 않는다. 셋째, 퍼미션 폴트가 발생하였을 때에는 폴트 핸들러를 동작시키지 않고 접근에 대해서만 무시를 한다.

그림 10은 MMP가 동작이 되어 접근 권한 검사를 하는 상태에서 실행을 하고난 후 함수 프레임의 메모리 내용을 본 것이다. 복사를 하려는 스트링이 모두 복사 되지 못하고 buf 변수의 공간 내에서만 복사된 것을 볼 수 있다. cnt 변수는 10의 값을 그대로 가지고 있다. 따라서 MMP가 버퍼 오버플로우에 대해서 보호를 제공한 것을 볼 수 있다.

이 실험 결과를 통하여 MMP는 프로텍션 도메인의 접근 권한 설정에 따라서, 잘못된 접근에 대한 메모리 보호를 제공하는 것을 확인할 수 있다.

5.2 크로스 도메인 콜링

진입이 허용된 위치에서의 프로시저나 함수에 진입을 위하여 MMP는 크로스 도메인 콜링을 한다. 반대로 허용이 되지 않은 위치로 진입을 시도할 경우 게이트 퍼미션 폴트가 발생하고 프로텍션 도메인 환경의 전환과 프로시저나 함수가 실행 되지 않는다. 그림 11은 크로스 도메인 콜링을 테스트하기 위한 소스이다.

테스트를 하기 위한 조건은 다음과 같다. 첫째, pd1_func() 함수는 프로텍션 도메인 1로 설정하고 pd2_func() 함수는 프로텍션 도메인 2로 설정한다. 둘째, 프로텍션 도메인 1에는 프로텍션 도메인 2의 pd2_func()

```
void main(void) {
    char buf[5];
    unsigned char cnt = 10;

    strcpy(buf, "hi, there?");
}
```

그림 9 버퍼 오버플로우 소스

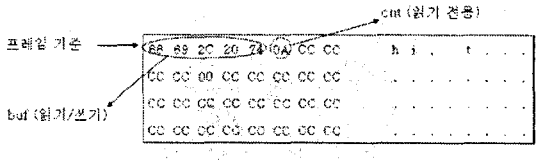

그림 10 MMP에 의하여 보호된 함수 프레임 내용

함수를 호출하기 위한 진입점을 설정 한다. 셋째, 게이트 퍼미션 폴트가 발생하였을 때에는 폴트 핸들러를 동작시키지 않고 크로스 도메인 콜링를 취소하고 자동적으로 리턴한다.

그림 12는 위의 조건에서의 실행 결과에 대한 메모리 내용을 보인 것이다. 크로스 도메인 콜링이 일어나서 pd2_func() 함수의 내용이 실행되어 foo 변수의 값이 1로 바뀐 것을 확인할 수 있다.

이 실험 결과를 통하여 특정 프로시저나 함수는 호출 권한을 가지고 있는 프로텍션 도메인에서만 호출이 가능하다는 것을 확인할 수 있다.

6. 결론

몬드리안 메모리 프로텍션은 워드 단위까지의 접근 권한 설정을 제공하는 메모리 보호 개념이다. 기존 프로세서의 명령어 셋의 추가 또는 변경과 프로그램 소스의 변경이 필요 없으면서, 프로그래머 또는 사용자에 의해서 생길 수 있는 잘못된 메모리 접근을 하드웨어적인 검사를 통하여 원천적인 방법으로 방지한다.

본 논문에서는 MMP의 구현을 하기 위하여 고려해야 할 점들에 대해서 분석을 하였다. MMP를 처음 제안한 Emmett Witchel의 논문들을 바탕으로 직접 구현을 하면서 발생하였던 여러 고려 사항과 문제점을 짚었고, 다음과 같은 대안 및 해결책을 제시하였다. 첫째, 8 비트 마이크로프로세서 환경에 맞게 바이트 단위까지 권한 설정이 가능하도록 퍼미션 벡터 테이블의 엔트리를 수정하였고, 트라이 인덱스를 변경하였다. 둘째, 퍼미션 테이블의 서브 블록 검사 오버헤드를 줄이기 위하여 퍼미션 벡터 테이블 엔트리의 구조를 변경하였다. 셋째, 크로스 도메인 콜링이 일어날 때, 접근 권한 검사를 무시해야 하는 상황에 대하여 언급하였다. 넷째, 프로텍션 도메인의 환경 전환을 위하여 프로텍션 도메인 정보를 가지고 있는 프로텍션 도메인 정보

```
void pd1_func(void) {
    int foo = 5;

    pd2_func(&foo);
}
```

```
void pd2_func(int *num) {
    *num = 1;
}
```

그림 11 크로스 도메인 콜링 소스

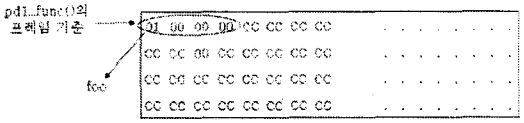

그림 12 크로스 도메인 콜링을 승인 하였을 때의 프레임 내용

테이블과 그 테이블의 위치를 가지고 있는 레지스터에 대한 개념을 제안하였다.

본 논문에서는 8 비트 프로세서를 기반으로 하고, 프로세서와의 결합이 아닌 주변장치의 개념으로 구현이 되었기 때문에 기존의 MMP와 많은 부분에서 차이가 발생하였다. 프로세서와 구현 환경으로 인하여, PLB와 GLB, 사이드카 레지스터, 주소 변환(Address translation)과 같은 MMP의 개념을 모두 구현하지 못하였지만, 차후에는 고성능의 프로세서를 사용하여 구현을 하고, 그 때 나타나는 고려 사항과 문제점을 더욱 분석하여 MMP가 실용화가 되기 위해서 필요한 점들을 찾을 것이다.

참 고 문 헌

[1] Matthew Simpson, Segment Protection for Embedded Systems Using Run-time Checks, CASES, San Francisco, CA, USA, September 2005

[2] Butler Lampson, Protection. In Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems, pages 437-443, Princeton University, 1971

[3] Emmett Witchel, Mondrian Memory Protection, ASPLOS, San Jose, CA, USA, October 2002

[4] Wikipedia, Page fault, August 2006, http://en.wikipedia.org/wiki/Page_fault

[5] Emmett Witchel, Mondrix : Memory Isolation for Linux using Mondriaan Memory Protection, SOSP, Brighton, United Kingdom, October 2005

[6] Emmett Witchel, Mondriaan Memory Protection, PhD thesis, Massachusetts Institute of Technology, February 2004