1999 年度 大韓電子工學會 夏季綜合學術大會 論文集 第 22 卷 第 1 號 99/6

프로시저 단위의 온라인 프로그램 교체

김영진, 이인환
한양대학교 전자공학과
progkim@shira.hanyang.ac.kr, ihlee@email.hanyang.ac.kr

Procedure-Based On-Line Program Replacement

Young-Jin Kim, Inhwan Lee
Dept. of Electronic Engineering, Hanyang University
progkim@shira.hanyang.ac.kr, ihlee@email.hanyang.ac.kr

Abstract

This paper presents a method for procedure-based on-line program replacement for user applications. To deal with the size change in the procedure to replace, the method uses the unmapped area of process address space for mapping a new version of procedure. The method is effective in that and it does not cause large performance loss during the replacement or require any additional software layer for the replacement. The method is illustrated in the SUN Solaris environment.

I. 서론

소프트웨어의 개발에 있어 한번에 완벽한 소프트웨어를 개발하는 것은 현실적으로 불가능하다. 그 주된 이유는 소프트웨어의 개발 단계에서 소프트웨어가 갖추어야 할 모든 기능을 예측하는 것이 어렵고, 개발 과정에서 소프트웨어에 내재된 모든 결함을 제거할 수 없기 때문이다. 따라서 기존 소프트웨어의 기능을 개선하거나 결함을 제거하기 위한 새로운 버전을 필요에 따라 설치하게 되는데, 일반적으로 새로운 버전의 소프트웨어를 설치하기 위해서는 해당 소프트웨어의 기능이 정지되어야 한다. 이러한 업그레이드를 위한 소프트웨어의 일시적인 동작 중단은 연속운전이 요구되는 환경에서는 대단히 심각한 문제가 된다. 따라서 업그레이드에 의한 동작 중단을 최소화하기 위해 동작중인 소프트웨어의 일부를 그 동작에 영향을 주지 않고 교체하고자 하는 온라인 소프트웨어 교체(on-line software replacement)의 필요성이 대두되고 있다.

온라인 소프트웨어 교체가 대단히 중요한 문제임에도 불구하고, 지금까지 소프트웨어 업그레이드에 의한 동작 중단이 당연한 것이라는 인식과 이에 따른 제한적인 연구로 인해 온라인 소프트웨어 교체는 실제 소프트웨어 환경에서는 거의 활용되지 못하고 있다. 온라인 소프트웨어 교체에 대한 대표적인 관련 연구들은 다음과 같다. Segal[1]은 유닉스 환경에서 프로시저 단위의 온라인 교체를 지원하기 위한 새로운 컴파일러와 링커를 포함하는 소프트웨어 설계 환경을 개발하였다. Gupta[2]는 프로세스 상태 전달(process state transfer)을 사용한 온라인 소프트웨어 교체 방법을 제시하였다. 또한 유닉스 환경에서 추가적인 소프트웨어 레이어나 별도의 환경을 필요로 하지 않으며, 프로세스 상태 전달 방법에 비해 교체 중의 성능 저하를 줄일 수 있는 방법도 연구되었다[3]. 이 방법에서는 기존의 방법들과는 달리 유닉스의 기본 서비스만을 이용하여 교체 대상 프로세스의 어드레스 스페이스를 직접 수정하는 방법을 사용하므로 구현이 단순하고 효율적이다. 그러나 교체로 인해 프로시저의 크기가 커지는 경우에 대해서는 DLL의 온라인 교체 방법을 적용하였다[4].

본 연구에서는 Sun Solaris 2.6 환경에서 사용자 응용 프로그램을 대상으로, DLL 교체 방식을 사용하지 않고, 교체 대상 프로시저의 크기 변화에 상관없이 적용 가능한 프로시저 단위의 온라인 소프트웨어 교체 방법을 제시한다. 특히 앞서 제시된 프로시저 단위 교체의 기본적인 방법들을 유지하면서, 교체로 인해 교체 대상 프로시저의 크기가 커지는 경우에 대처하기 위한 방법에 초점을 맞추어, 새로운 프로시저를 위한 공간으로 비매핑 영역(unmapped area)을 사용하는 교체 방법을 제시한다. 이 방법에서는 교체 대상 프로시저의 크기가 커지는 경우에도 교체 대상 프로시저를 DLL로 변환하지 않고, 일관적인 방법으로 교체를 수행할 수 있다.

II. 교체 환경 및 방법

일반적으로 온라인 소프트웨어 교체에서는 교체 수행 과정이나 교체가 끝난 후 교체 대상 소프트웨어의 오동작이 발생하지 않아야 한다. 또한 온라인 교체로 인한 대상 소프트웨어의 성능 저하가 최소화되어야 한다. 이를 위해 구 버전의 교체 대상 프로시저를 수정하기 전에 해당 프로시저의 실행 상태를 관찰하여 이 프로시저가 실행 중이 아닐 때 수정을 시작하여야 하며, 교체 대상 프로시저가 수정 도중에 실행되는 것을

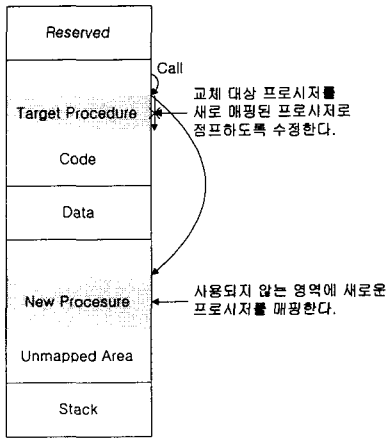

그림 1. 교체의 개념

방지하기 위한 조치를 취하여야 한다. 그리고 하나의 프로세스는 디스크와 메인 메모리, 캐시 메모리의 내용들로 구성되므로 온라인 교체를 위해서는 이들 모두를 새로운 버전의 프로시저로 업데이트하여야 한다[3].

교체하려고 하는 프로시저의 크기가 증가하는 경우에는, 교체에 앞서 새 버전의 프로시저를 위한 공간을 확보하여야 한다. 이를 위하여 본 논문에서는 프로세스 어드레스 스페이스에서 프로세스에 의해 사용되지 않는 공간(비매핑 영역; unmapped area)을 활용한다. 구체적으로 프로세스 어드레스 스페이스에서 프로그램 코드, 데이터, 스택 등을 위해 미리 할당된 영역 이외의 사용되지 않는 공간에 교체 대상 프로시저의 새 버전을 매핑하고, 구 버전의 프로시저의 첫 부분을 수정하여 이 프로시저가 호출될 경우 새 버전의 프로시저로 다시 호출이 일어나도록 한다(그림 1). 이렇게 하면 구 버전 프로시저가 호출되었을 경우 새 버전 프로시저로 이동한 후에 실행이 일어나므로 결과적으로 프로시저 교체가 이루어진다.

이러한 방법으로 교체를 수행하기 위한 환경은 구 버전의 실행 파일과 이 실행 파일이 실행되어 생성된 교체 대상 프로세스, 교체할 새 버전 프로그램의 실행

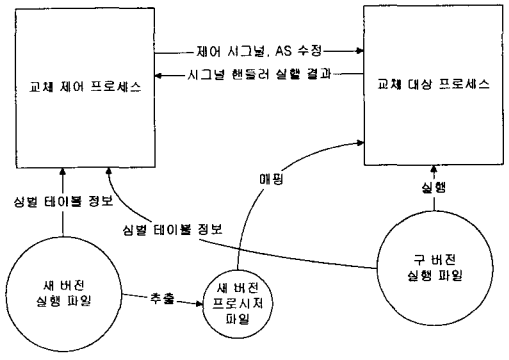

그림 2. 교체 환경

파일과 이로부터 추출된 새 버전의 프로시저 파일, 그리고 전체 교체 과정을 수행하는 교체 제어 프로세스로 구성된다(그림 2).

교체 대상 프로세스가 실행되고 있는 상태에서 교체가 시작되면 교체 제어 프로세스는 교체 대상 프로세스 내의 구 버전 프로시저를 새 버전 실행 파일 내의 새 버전 프로시저로 교체한다. 그림 2에서 알 수 있듯이 교체 과정에서 교체 제어 프로세스는 구 버전과 새 버전 실행 파일의 심벌 테이블 정보를 이용하여, 새 버전 실행 파일에서 새 버전 프로시저를 추출한 다음, 독립된 파일(새 버전 프로시저 파일)로 생성시키고, 이를 교체 대상 프로세스의 어드레스 스페이스에 매핑한다. 또한, 교체 제어 프로세스는 교체 대상 프로세스의 어드레스 스페이스에서 필요한 부분을 수정한다. 이러한 과정에서 교체 제어 프로세스와 교체 대상 프로세스 사이의 정보 전달은 시그널을 사용하여 이루어진다.

교체 과정에서 새 버전 프로시저를 교체 대상 프로세스의 어드레스 스페이스에 매핑함으로 인해 새 버전 프로그램 코드의 일부인 새 버전 프로시저가 구 버전의 프로세스 어드레스 스페이스에 존재하게 되는데, 이 경우 중요한 것은, 교체 대상 프로시저의 크기가 새 버전과 구 버전 사이에 서로 다르므로, 새 버전의 실행 파일 내의 심벌들의 위치가 구 버전과는 다르게 된다는 것이다. 즉, 새 버전의 교체 대상 프로시저가 이 프로시저 외부의 심벌들을 액세스할 때 이 심벌들의 위치는 구 버전의 그것들과는 다르게 되며, 따라서 새 버전의 프로시저가 구 버전을 이용하여 설정된 프로세스 어드레스 스페이스 내에서 정상적으로 동작하게 하기 위해서는 새 버전 프로시저 내의 이러한 심벌

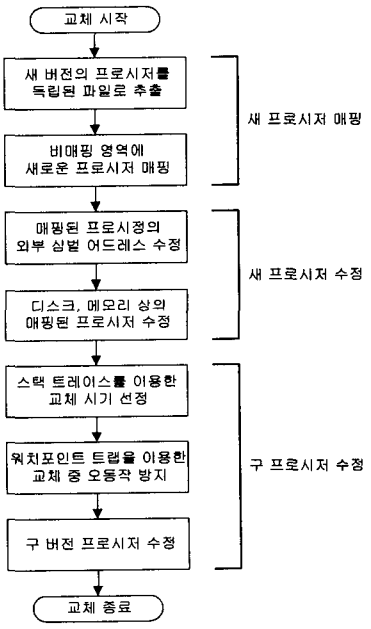

그림 3. 교체 절차

들의 위치를 구 버전에 맞게 수정해 주어야 한다.

이러한 사항들을 고려하여 본 논문에서는 새 버전 프로시저 매핑, 새 버전 프로시저 수정, 구 버전 프로시저 수정의 세 단계로 교체를 수행한다(그림 3).

III. 새 버전 프로시저 매핑

새 버전의 프로시저를 구 버전 프로세스의 비매핑 영역에 매핑하기 위해 먼저 새 버전 실행 파일의 심벌 테이블로부터 새 버전의 교체 대상 프로시저의 위치와 크기를 알아낸 후, 새 버전 실행 파일로부터 새 버전의 프로시저를 추출하여 독립된 파일로 생성시킨다.

다음으로 추출된 새 버전 프로시저 파일을 구 버전 프로세스 어드레스 스페이스의 비매핑 영역으로 매핑한다. 새 버전 프로시저의 매핑은 mmap(2) 시스템 콜을 사용하여 수행한다[5]. 그런데 이 시스템 콜은 자신을 호출한 프로세스의 어드레스 스페이스에 대해서만 그 기능을 수행할 수 있기 때문에, 매핑을 수행하는 프로시저가 교체 대상 프로그램의 내부에 포함되어 있어야 한다. 이러한 기능은 시그널(signal)을 이용하여 실현한다.

우선 signal(3C) 기능을 이용하여 교체 대상 프로그램 내에 시그널을 설정하고, 이 시그널을 받았을 때 실제의 매핑을 수행할 시그널 핸들러(signal handler)를 정의한다. 그리고 교체 제어 프로세스는 매핑이 필요할 경우 교체 대상 프로세스에 시그널을 보낸다. 새 버전 프로시저의 매핑이 완료되면 시그널 핸들러는 다시 시그널을 이용하여 제어 프로세스에 이 사실과 새 버전 프로시저가 매핑된 위치를 알린다. 이러한 시그널 핸들러는 교체 대상 프로그램 내에 존재해야 하므로 기존 프로그램에 이를 추가하여야 하는데, 이에 필요한 시그널의 초기화와 실제 매핑을 수행하는 과정은 라이브러리의 형태로 만들어 재사용 가능하도록 하였다. 이렇게 시그널 핸들러를 이용하여 교체를 수행하면, 시그널 핸들러가 실행되는 동안 이 프로세스는 다른 기능을 제공할 수 없으며, 이에 따른 성능 저하가 일어날 수 있다. 따라서 시그널 핸들러의 기능을 최소화하는 것이 중요하다.

IV. 새 버전 프로시저 수정

새 버전 프로시저의 매핑이 완료되어 그 위치가 결정되면, 새 버전의 프로시저가 구 버전의 프로세스 어드레스 스페이스에서 올바르게 동작하도록 외부 심벌을 액세스하는 부분을 수정한다.

외부 심벌 어드레스를 수정하기 위해서는 어떤 명령어들이 수정의 대상이 되는지 알아야 하고, 또한 수정 대상 명령어의 심벌 필드 부분을 어떻게 수정할 것인지를 결정하여야 한다. SPARC 머신에서 수정의 대상이 되는 명령어 즉, 외부 심벌을 액세스하는 명령어를 살펴보면 데이터 엑세스와 관련되는 로드/스토어 명령어에서 그 대상이 로컬 스태틱 데이터이거나 전역 데이터인 경우, 프로시저 호출을 수행하는 명령어, 그리고 간접 레지스터 주소 지정 방식을 사용하여 점프를 수행하는 명령어가 이에 해당된다[6]. 이러한 명령어에 대하여 심벌 필드를 수정하는 방법은 명령어의 주소 지정 방식에 따라 달라진다. SPARC 머신에서는 레지스터, 레지스터 간접, 즉치, PC 상대의 네 가지 주소 지정 방식이 있으며, 이 중 수정 대상이 되는 명령어들이 사용하는 것은 레지스터 간접과 PC 상대의 두 가지 주소 지정 방식뿐이다.

먼저 레지스터 간접 주소 지정 방식인 경우에는 수정 대상 명령어가 주소 지정에 사용할 레지스터의 값을 수정해야 한다. SPARC 어셈블러에서는 레지스터 간접 주소 지정 방식을 사용하는 명령어의 바로 앞에 항상 "set"이라는 명령어로 주소 지정에 사용할 레지스터의 값을 정한다. 따라서 새 버전과 구 버전 실행 파일의 심벌 테이블 정보를 이용하여, 수정 대상 명령어 앞에 위치한 "set" 명령어의 오퍼랜드(operand)를 구 버전의 프로세스 어드레스 스페이스 상의 심벌 어드레스로 수정한다. (SPARC 에서 "set" 명령어는 의사 명령어로서 실제로 기계어 코드가 생성될 때는 "or"나 "sethi" 또는 이 두 명령어의 조합으로 바뀐다.)

SPARC 에서 수정의 대상이 되는 명령어 중 PC 상대 주소 지정 방식을 사용하는 것은 "call" 명령어뿐이며, 이 명령어는 2비트의 "opcode"와 30비트의 "displacement"로 구성된다. 이 때 타겟 어드레스는 "displacement"의 값을 왼쪽으로 2비트 로지컬 쉬프트한 값과 PC값을 더해서 구해진다. 일반적으로 이 명령어의 위치(PC값)가 구 버전과 새 버전에서 다르므로 "displacement" 부분을 수정함으로써 구 버전 프로세스 어드레스 스페이스 상의 심벌 어드레스를 액세스하도록 한다.

이러한 방법을 사용하여 새 버전 프로시저의 심벌 어드레스 수정이 완료되면 구 버전 프로세스 어드레스 스페이스에 매핑된 새 버전 프로시저의 실제 수정이 이루어진다. 이러한 수정은 디스크, 메인 메모리, 캐시 메모리의 순서로 수행된다.

디스크 수정의 경우 write(2) 시스템 콜을 사용하여 수행한다. 메인 메모리의 수정은 두 가지 방법을 생각할 수 있다. 첫 번째는 어드레스 스페이스를 직접 수정하는 방법으로, Solaris에서는 프로세스 파일 시스템(process file system, "/proc")의 "as"라는 파일을 이용하여 프로세스의 어드레스 스페이스를 수정할 수 있다[5]. 두 번째는 새 버전의 프로시저가 매핑된 페이지를 무효화시키는 방법이다. 무효화된 페이지는 다음에 참조될 때 디스크로부터 다시 읽혀지게 되므로 디스크 수정 후 페이지를 무효화하면 메인 메모리의 내용도 새 버전으로 교체되는 효과를 갖는다[7]. 이를 위해 프로세스 어드레스 스페이스에 대한 여러 가지 제어 기능을 지원하는 memcntl(2) 시스템 콜을 사용한다[5].

SPARC 시스템은 명령어 캐시와 데이터 캐시가 분리되어 있고, 명령어 캐시에 대해서는 어드레스 스페이스의 수정 내용이 캐시 메모리에 반영되지 않는다. 따라서 메인 메모리의 수정이 이루어진 다음에 캐시 메모리의 내용도 수정해 주어야 프로세서가 수정된 명령어를 바로 실행할 수 있다. 이를 위하여 Solaris에서 제공하는 sync_instruction_memory(3C) C 라이브러리 함수를 사용한다[5]. 이 함수는 주어진 어드레스 범위에 해당하는 명령어 캐시의 엔트리를 무효화하므로, 이후 프로세서에서 명령어들을 실행할 때 캐시의 내용도 새로운 버전의 것으로 변경된다.

페이지 무효화를 수행하는 memcntl(2) 시스템 콜과

캐시 동기화를 수행하는 sync_instruction_memory(3C) 함수는 mmap(2) 시스템 콜과 마찬가지로, 자신을 호출한 프로세스의 어드레스 스페이스에 대해서만 그 기능을 수행할 수 있으므로 mmap(2) 시스템 콜과 같이 시그널을 이용해 구현된다.

V. 구 버전 프로시저 수정

교체의 마지막 단계로 구 버전 프로시저를 새 버전 프로시저를 호출하는 동작을 수행하도록 수정한다. 이 때 구 버전 프로시저가 실행 중일 때 수정을 시작하거나, 수정 도중에 구 버전 프로시저가 호출되어 실행되면 프로그램의 정상적인 동작을 보장할 수 없으므로, 이에 대한 조치를 취해 주어야 한다.

먼저 구 버전 프로시저가 실행 중이 아닐 때 수정을 시작하기 위하여 스택 트레이스(stack trace)를 통해 프로시저의 실행 상태를 확인한다. 일반적으로 런타임 스택에는 모든 실행 중인 프로시저에 관한 정보가 호출된 순서대로 저장되므로 스택 트레이스 정보를 이용하면 교체 대상 프로시저가 실행 중인지를 정확히 판단할 수 있다[8]. 실제로 Solaris에서는 pstack(1) 을 이용하여 런타임 스택의 내용을 알아낼 수 있다[5].

이렇게 구 버전 프로시저가 실행 중이 아닐 때 교체를 시작하더라도, 교체가 수행되는 도중에 구 버전 프로시저가 새로 호출되어 실행되면, 이 프로시저는 그 기능상의 일관성이 유지되지 않은 상태에서 실행이 되고 이에 따라 오동작이 발생할 수 있다. 본 논문에서는 성능 저하를 최소화하면서 이러한 오동작을 막기 위해, 워치포인트 트랩(watchpoint trap)을 사용하여 구 버전 프로시저가 수정 중에 호출되는 경우에만 해당 프로세스가 정지되도록 하였다[5]. 워치포인트 트랩은 프로세스 어드레스 스페이스의 연속된 일부 영역을 감시 영역(watched area)으로 설정해 두면 그 영역이 참조되었을 때 트랩에 의해 프로세스가 정지되는 기능으로, 이를 이용하여 구 버전 프로시저의 영역에 대하여 감시 영역을 설정해 놓으면 구 버전 프로시저가 실행되려고 할 때 트랩이 발생하고 해당 프로세스가 정지하게 되어 수정 도중의 오동작을 방지할 수 있다.

워치포인트 트랩이 설정되면 구 버전 프로시저의 시작 부분을 새 버전의 프로시저가 매핑된 위치로 점프하는 동작을 수행하도록 수정한다. 이러한 동작은 "sethi"와 "jmpl"명령어의 조합으로 이루어지며 실제 수정은 매핑된 새 버전 프로시저를 수정하는 것과 같은 과정을 따라 수행한다. 마지막으로 설정된 워치포인트 트랩을 해제하는 것으로 전체 프로시저 교체 과정이 완료된다.

VI. 교체 방법의 구현

지금까지의 연구 결과를 바탕으로 하여 Sun Solaris 2.6 에서 C를 이용하여 실제로 온라인 프로그램 교체를 지원하는 소프트웨어를 제작하고, 간단한 예제 프로그램을 대상으로 온라인 교체를 수행해 보아 그 기능을 확인하였다. 제작한 교체 소프트웨어는 교체 제어 프로그램과 교체 대상 프로그램에 포함시키는 라이브러리로 구성되어 있으며, 그림 2와 같은 환경으로 동작한다.

VII. 결론

본 논문에서는 유닉스 환경에서 사용자 응용 프로그램을 대상으로 프로시저 단위의 온라인 프로그램 교체를 수행하기 위한 방법과 절차를 제시하고 그것을 SUN Solaris 2.6 환경에 적용하여 검증하였다.

본 논문에서 제안한 교체 방법은 교체 대상 프로시저의 크기 변화에 상관없이 교체를 수행할 수 있으며, 최소한의 교체 단위인 프로시저 단위의 온라인 교체를 지원하므로, 교체 중의 성능 저하가 적다. 특히, 제안한 방법은, 기존의 프로시저 단위의 온라인 프로그램 교체에 대한 연구와는 달리, 온라인 교체를 지원하기 위한 별도의 환경을 사용하지 않고, 유닉스에서 기본적으로 제공하는 서비스들을 이용하여 온라인 교체를 수행한다. 또한, 새로운 프로시저를 프로세스 어드레스 스페이스의 비매핑 영역에 매핑하는 방법을 사용하므로, 교체 대상 프로시저의 크기가 커지는 경우에도 기존 프로시저를 DLL로 변환하는 과정 없이 일관적인 방법으로 교체를 수행한다. 따라서 제안한 방법은 보다 효율적으로 교체를 수행할 수 있으며, 기존의 유닉스 환경에서 동작하는 소프트웨어에 간단한 시그널 핸들링을 추가함으로써 이에 대한 온라인 프로시저 교체를 가능하게 한다.

앞으로 본 연구에서 제안한 방법을 실제의 대규모 소프트웨어에 적용하고, 온라인 교체를 지원하기 위한 버전 관리와 사용자 인터페이스 등을 포함하는 교체 도구의 개발, 데이터의 온라인 교체 등에 대한 연구가 이루어질 것이다.

참고문헌

[1] O. Frieder and M. E. Segal, "On dynamically updating a computer program: from concept to prototype," *J. Systems and Software*, pp. 111-128, Feb. 1991.

[2] D. Gupta and P. Jalote, "On-Line Software Version Change Using State Transfer Between Processes," *Software Practice and Experience*, pp. 949-964, Sep. 1993.

[3] 김형곤, 이인환, "유닉스 환경에서의 프로시저 단위의 온라인 소프트웨어 교체," 1998 정보과학회 가을 학술대회 논문집, pp. 514-516, 1998. 10.

[4] 김화준, 이인환, "유닉스 환경에서의 DLL의 동적 업그레이드," 1998 정보과학회 가을 학술대회 논문집, pp. 600-602, 1998. 10.

[5] *"UNIX Reference Manual,"* SUN Microsystems, Inc., 1997.

[6] Richard P. Paul, *"Sparc Architecture Assembly Language Programming, &C,"* Prentice Hall, 1994.

[7] *"System Interface Guide,"* SUN Microsystems, Inc., 1997.

[8] W. Richard Stevens, *"Advanced Programing in the Unix Environment,"* Addison Wesley, 1992.