論文 95-6-6-24

객체의 동적 재구성을 지원하는 커널의 구조

正會員 柳 廣 鉉*, 曺 裕 根*

A Kernel Architecture Supporting Dynamic Reconfiguration of Objects

Kwang Hyun Yoo*, Yoo Kun Cho* Regular Members

본 연구는 1994년도 한국 과학 재단의 목적 기초 연구비의 지원에 의하여 연구되었음

要 約

본 논문에서는 운영 체제 커널과 응용 프로그램의 동적 재구성을 효율적으로 지원할 수 있는 객체에 기반한 운영 체제 커널의 구조를 제시한다. 제시한 커널에서는 동적 재구성을 객체 내용에 대한 재구성과 객체의 동작 환경에 대한 재구성으로 구분하여 동작 환경에 대한 재구성을 지원하기 위해서는 반영적 구조를 채택하였고, 객체 내용의 재구성을 지원하기 위해서는 객체와 객체를 실체화(instantiate)할 수 있는 가상 주소 공간을 분리하여 제공하는 기법을 사용하였다. 객체와 가상 주소 공간을 분리하여 제공하는 기법은 객체 내용을 효율적으로 재구성할 수 있게 하고, 반영적 구조상의 메타 객체에 의한 객체의 관리 기법은 객체 수행 환경을 유연하게 제어할 수 있게 한다. 제안된 접근 방법의 타당성을 보이기 위해 객체의 재구성을 지원하는 예와 프로토타입 시스템의 성능 데이타를 보였다.

ABSTRACT

In this paper, we present the architecture of an operating system kernel that provides an efficient reconfiguration mechanism for itself and the applications. In the kernel, the reconfiguration process is divided into the reconfiguration of the environment and the contents of an object. The former is provided by the reflective architecture of objects and the latter is provided by separating the object and its virtual address space in which the objects can be instantiated. The separation of the object and the virtual address space enables an efficient reconfiguration of the contents of an object, and the management of an object by meta object in the reflective architecture can provide the flexibility in controlling the execution environment of an object. The reconfiguration examples and the performance data of the prototype implementation are presented to demonstrate the effectiveness of proposed approaches.

* 서울대학교 컴퓨터공학과
論文番號 : 95143-0413
接受日字 : 1995年 4月 13日

Ⅰ. 서 론

운영 체제는 응용의 요구에 따라 자신의 자원 관리 정책을 동적으로 재구성함으로써 응용 프로그램의 요구를 효율적으로 지원할 수 있다. 그러나, UNIX[2]의 경우와 같이 커널내에 고정된 자원 관리 정책은 운영 체제 자원 관리 정책의 적응성을 떨어뜨리고, 응용으로부터의 다양한 자원에 대한 요구를 만족시키기 힘든 단점이 있다. 따라서, 운영 체제의 자원 관리 정책을 동적으로 재구성할 수 있는 기능을 운영 체제가 제공하고 응용이 자신의 자원 사용 패턴에 따라 자원을 서비스 받을 수 있다면, 보다 효율적인 응용의 동작 환경을 구성할 수 있을 것이다.

일반적 의미의 동적 재구성은 응용의 동작 중에 그 구성 요소 - 프로그램 모듈 혹은 데이타 -를 상황에 따라 변경하는 연산으로 정의할 수 있다[5]. 이러한 동적 재구성의 예로 소프트웨어 모듈을 대치(*replace*), 이동(*move*), 추가(*add*), 제거(*delete*)하는 작업을 들 수 있다. 특히, 객체 지향 운영 체제는 서로 다른 자원을 관리하는 정책을 구현한 객체들의 집합으로 구성되기 때문에 객체에 대한 동적 재구성이 운영 체제를 재구성하는 효과를 낳을 수 있다.

본 논문에서는 응용 프로그램 및 운영 체제 커널의 동적 재구성 기능을 효과적으로 지원하기 위하여 운영 체제의 자원을 객체로 모델링하고 각 객체에 대한 관리가 메타 객체에 의하여 수행되는 반영적 구조를 갖는 운영 체제의 구조를 제시한다. 또한, 객체의 재구성을 용이하게 지원키 위한 기법으로서 객체와 가상 주소 공간의 서비스를 별도로 제공하는 방법을 사용하였다.

논문의 2장에서는 연구 배경을 소개하고, 3장에서는 객체와 반영의 개념을, 4장에서는 SNU ROSE의 구조를, 5장에서는 객체의 동적 재구성 지원을 위한 기능 및 그 예를, 6장에서는 기존 연구와의 비교 분석을, 7장에서는 SNU ROSE의 구현 및 성능에 대한 내용을 기술하고, 마지막으로 8장에서 결론을 맺는다.

Ⅱ. 연구 배경

운영 체제의 주요한 역할중의 하나는 다수의 응용에서 발생하는 운영 체제 자원에 대한 요구를 중재 및 배분하는 것이다. 그러나, 운영 체제, 특히 커널에 자원 관리를 집중한 시스템의 경우 다양한 응용의 운영 체제 자원에 대한 요구를 만족하기 힘든 단점이 있다. 예를 들면, 기존의 운영 체제에 있어서 스케쥴링 정책은 응용 프로그램의 수행 시간에 상당한 영향을 미친다. 일반적인 시분할 스케쥴링 기법은 다수의 쓰레드가 하나의 록(lock)에 대하여 경쟁하는 상황에서 록의 획득 및 반환에 대한 정보의 부족으로 인하여 쓰레드의 병렬성을 최대한 지원하지 못한다고 알려져 있다[3]. 따라서, 응용에 최적화된 수행 환경을 제공하기 위하여는 응용의 자원 사용 패턴에 따라 운영 체제내의 자원 관리 정책을 탄력적으로 조절할 수 있는 기능을 필요로 하며, 응용 프로그램이 자신이 사용하는 운영 체제의 자원 관리 정책에 개입하여 자신에 맞는 관리 기법을 선택할 수 있어야 한다. UNIX System V와 같은 기존 운영 체제의 경우 프로세스 스케쥴링의 경우 실시간 스케쥴링 클래스[11]와 같이 일부 수행 환경에 대한 재구성이 가능하나, 커널에서 제공하는 모든 자원에 대한 재구성 기능을 제공하지 못하며, 각 자원의 재구성에 대하여도 일관적인 기법을 제공하고 있지 못하다.

본 논문에서는 응용에 최적화된 수행 환경을 제공하기 위한 방법의 일환으로서 운영 체제의 자원을 객체로 모델링하고, 객체에 대한 관리 기법으로는 반영적 구조(reflective architecture)를 도입하였다. 반영적 구조하에서는 각 객체의 관리가 해당 객체의 메타 객체에 의하여 수행되는 구조를 갖기 때문에 객체에 대한 관리 정책의 재구성이 쉽게 이루어 질 수 있는 장점이 있다. 반영적 구조를 운영 체제에 적용하려는 기존의 연구는 Apertos[13]에 의하여 시도되었다. 반영적 구조는 객체의 관리 정책에 대한 재구성이 용이하고, 객체의 관리가 메타 연산으로 제공되므로 시스템 전반에 대하여 일관된 정책을 유지할 수 있는 잇점이 있다. 그러나, Apertos의 경우 시스템에서 제공하는 객체의 형태가 객체의 실 구현과 가상 주소 공간, 수행 쓰레드의 집합으로 구성되어 객체의 호출이 항상 메시지에 의하여 수행되는 구조를 갖는다. 이와 같은 객체의 구조는 대형, 혹은 중규모의 객체를 구현하기에는 적합하나 소규모 객체의 지원 기능이 부족하고 객체 내용의 동적 재구성을 지원하기에는 시스템 부담의 증가 및 성능 저하등의 문제가 있다.

이와 같은 문제를 해결하기 위하여 본 논문에서 제시한 시스템에서는 객체를 수행하기 위한 환경을 구성하기 위한 요소인 객체의 실 구현 - 즉, 상태 정보 및 메소드

구현 - 과 객체를 사상할 수 있는 가상 주소 공간을 분리하고, 객체를 수행하는 쓰레드를 별도로 제공하는 기법을 채택하였다.

일반적으로 운영 체제에서 제공하는 객체는 객체의 원래 의미인 캡슐화(encapsulate) 특성을 지원하기 위하여 객체마다 독립적인 보호 공간을 형성하며, 이는 보통 객체마다 독자적인 가상 주소 공간을 통하여 구현된다[1]. 그러나, 메타 객체는 자신이 관리하는 객체의 상태 정보및 메소드 구현에 항상 접근할 수 있어야 하므로, 객체마다 독립적인 보호 공간을 갖는다면, 메타 객체가 관리 연산을 수행하기 위해서는 항상 보호 공간의 경계를 넘어야 하며, 이는 필연적으로 성능의 저하를 유발할 수 있다. 또한, 소규모 객체를 구현하려고 할 때 객체마다 하나의 독립적인 주소 공간을 제공한다면, 객체의 호출시마다 보호 공간의 경계를 넘는 비용을 지불해야만 한다. 따라서, 객체의 효율적인 호출 기법의 구현과 재구성을 포함한 관리 연산의 효율적인 지원을 위해서 객체 보호 공간의 개념을 객체의 개념에서 분리한다면, 다수의 객체를 하나의 보호 공간에 사상할 수 있고 이를 통하여 보호 공간의 경계를 넘는 비용을 절약할 수 있을 것이다. 이와 같이 객체와 객체를 저장하고 사상(mapping)할 수 있는 가상 주소 공간에 대한 서비스를 별도로 제공하는 기법은 Lipto[10], Distributed Shared Repository (DSR)[7]등에서 시도되었다. 그러나, DSR의 경우 프로그램, 데이타및 제어 흐름을 context로 추상화하고, context를 가상 주소 공간으로부터 분리하였다. 그리고, 가상 주소 공간과 영속 저장소(persistent store)사이에서의 context 이동을 자유롭게 제공함으로써 분산 환경에서 객체 공유를 효과적으로 처리할 수 있도록 하였다. 그러나, DSR의 경우 스케쥴링이나 페이징과 같은 객체의 수행 환경에 대한 고려가 이루어지지 않았으며, 따라서 객체 수행 환경의 재구성 기능은 제공하지 못한다. Lipto의 경우 객체는 시스템 설정(configuration)의 단위이며, 설정을 제어하기 위한 구조로서 MOI(module/object infrastructure)를 제시하고 있다. 그러나, MOI구조는 객체의 의존 관계를 기술, 객체를 설정하기 위한 구조만을 갖기 때문에 객체의 동작 환경에 대한 재구성 기능을 효과적으로 지원하기에는 미흡하다.

본 논문에서는 응용에 동적 재구성 기능을 효율적으로 제공하기 위하여 동적 재구성을 객체 수행 환경의 재구성과 객체 내용의 재구성으로 구분하고, 객체 수행 환경의 재구성 지원을 위하여 반영적 구조에 기초한 객체 관리 정책을 사용하고, 객체 내용의 재구성 지원을 위하여 객체와 가상 주소 공간을 분리하는 기법을 채택하였다. 본 논문에서 제시한 운영 체제 구조는 객체 수행 환경의 재구성은 메타 객체의 교환에 의하여, 객체 내용의 재구성은 해당 객체를 특정 가상 주소 공간에 사상함으로써 객체 수행 환경 및 객체의 내용을 손쉽게 재구성할 수 있는 구조를 지원한다.

Ⅲ. 동적 재구성 지원을 위한 객체 모델

1. 객체

본 논문에서 제시한 시스템에서는 객체에 대한 동적 재구성을 효과적으로 지원하기 위한 방법으로서 객체를 본형(template)과 인스턴스(instance)로 구분하여 지원한다. 객체의 본형은 객체의 특성을 나타내는 것으로 시스템에서 객체를 나타내고 제어하기 위한 단위를 형성한다. 본형은 프로그래밍언어 수준의 객체로 볼 때 클래스 혹은 타입에 해당하는 것으로 객체의 인스턴스를 생성하기 위한 기본적인 도구를 제공한다. 객체의 인스턴스는 객체의 메소드 실 구현 및 상태 정보를 갖는 것으로, 특정 가상 주소 공간상에 사상되어 수행 혹은 이용중인 상태를 의미한다.

새로운 객체의 수행은 객체의 본형을 이용하여 객체의 인스턴스를 특정 가상 주소 공간에 생성한 후 객체의 인스턴스를 이용하는 쓰레드 객체를 해당 객체에 부착함으로써 이루어진다. 이와 같은 객체의 본형 및 인스턴스의 분리 구조는 소규모 객체의 경우 하나의 가상 주소 공간에 다수의 객체를 사상할 수 있기 때문에 객체의 호출을 일반적인 프로시쥬어 호출 정도의 비용으로 낮출 수 있어 성능의 이득을 가져 올 수 있다.

2. 객체의 재구성

일반적인 의미의 객체는 상태 정보와 정보를 관리 유지하는 메소드의 집합으로 구성되나, 운영 체제 입장에서의 객체는 객체의 일반적인 의미뿐만 아니라 객체를 수행시키기 위하여 필요한 환경 - 가상 주소 공간, 수행 쓰레드등 - 이 부가된 형태를 지니게 된다. 따라서, 본 논문에서 제시한 시스템에서는 객체의 동적 재구성을 객체 수행 환경의 재구성과 객체 내용의 재구성으로 구분

하였다.

● 객체의 수행 환경에 대한 재구성

객체의 수행 환경은 객체를 수행할 수 있도록 운영 체제 커널이 형성하는 서비스들의 집합이다. 예를 들면, 수행 쓰레드, 가상 주소 공간, 물리적 기억 장치의 페이지 집합, 화일 또는 장치등의 논리적 자원들을 들 수 있다. 본 논문에서 제시한 시스템에서 객체의 수행 환경은 운영 체제가 형성하는 것이 아니라 해당 객체의 관리자인 메타 객체들에 의하여 형성된다. 따라서, 객체 수행 환경의 재구성은 메타 객체를 교환(switching)함으로써 수행되는 방식을 지원한다.

● 객체 내용의 재구성

객체의 실 구현에 대한 재구성으로, 객체의 상태 정보를 재구성하거나 메소드의 실 구현을 변경하는 동작을 의미한다. 객체 내용의 재구성을 위해서는 재구성의 대상이 되는 객체의 상태, 객체의 프로그램 이미지 구조, 객체의 동작 구조에 관한 정보등을 모두 알고 있어야만 가능하다. 따라서, 운영 체제에서 모든 객체의 재구성을 담당하는 기법은 유지해야 할 정보의 양이 방대하고, 다양한 객체의 구조를 모두 운영 체제 에서 인식하고 재구성해야 하므로 거의 불가능하다. 따라서, 운영 체제는 객체 재구성을 효율적으로 수행할 수 있도록 기본적인 기능을 제공하고 객체 내용의 재구성은 해당 객체의 메타 객체가 수행하는 구조를 사용하였다.

3. 객체의 관리 구조 - 반영적 구조

본 논문에서 제시한 운영 체제는 자원에 대한 서비스를 제공하는 다수의 객체로 구성된다. 시스템내의 각 객체에 대한 관리는 반영적 구조에 기반하고 있다.

반영적 구조의 시스템은 객체와 메타 객체 관련성으로 구성된 인과적 접속 관계(causal connection links)[8]에 의하여 객체들의 계층적 구조가 형성된다. 인과적 접속 관계는 객체와 다른 객체 - 보통은 메타 객체 - 와의 관련성을 형성하며, 인과적 접속 관계에 있는 한 객체가 변형될 경우 그 영향이 다른 객체에 미칠때 두 객체는 서로 인과적 접속 관계가 있다고 한다[8]. 따라서, 반영적 구조하에서 프로그램 구조 및 행동 양식은 메타 수준의 시스템에 의하여 정의된다. 또한 메타 수준의 시스템 또한 자신의 메타 수준의 시스템에 의하여 정의된다. 이때 객체의 구조 및 행동 양식에 대한 반영적 동작을 수행할 수 있는 객체를 해당 객체의 메타 객체라 하며, 메타 객체에 의해 객체의 행동 양식을 변화시키기 위하여 수행되는 연산을 메타 연산(meta computing)이라 한다.이와 같은 반영(reflection)은 구조적 반영(structural reflection), 계산적 반영(computational reflection), 자원적 반영(resource reflection)으로 구분할 수 있다[6]. 구조적 반영은 객체의 구조적인 측면, 즉 객체의 상태 정보 및 메소드의 집합에 대한 반영적 연산을 제공한다. 구조적 반영의 예는 객체의 구조적 표현 및 내용의 정의, 생성 및 변경등의 연산을 일컫는다. 계산적 반영은 객체의 계산적 측면에 대한 반영적 동작을 수행하는 것으로, 예를 들면, 객체 호출 메시지의 도착, 메소드의 수행과 같은 동작을 들수 있다[12]. 자원적 반영은 운영 체제의 관점에서 객체의 관리적 측면에 대한 반영이다[6]. 자원적 반영은 객체의 수행 환경 및 자원의 관리 정책과 연관성을 갖으며, 페이징 정책, 스케쥴링 정책등은 대표적인 자원적 반영의 예이다.

본 논문에서 제시한 시스템내의 각 객체는 자신과 인과적 접속 관계를 갖는 메타 객체의 집합을 갖는다. 이때, 하나의 메타 객체는 자신의 관리하에 있는 다수의 객체를 가질 수 있으며, 한 객체는 다수의 메타 객체를 갖는다.

Ⅳ. SNU ROSE 프로토타입 시스템

본 논문에서 제시하고 있는 객체 지향 운영 체제 SNU ROSE(Reconfigurable Operating System Environment) 프로토타입 시스템의 전체 구조 및 시스템내의 객체들은 그림 1에 나타나 있다.

SNU ROSE 프로토타입 시스템은 구현 편의상 5 층으로 구성되었으나 사용 목적에 따라서 가감될 수 있다. 응용 층(application layer)은 응용 프로그램을 위한 사용자 수준의 객체들로 구성된다. 응용 환경 층(application environment layer)은 일반 운영 체제의 라이브러리 시스템이나 시스템 프로세스에 해당하는 부분으로 운영 체제 환경 층과 기본 객체 층의 서비스를 이용하여 응용 층의 객체를 지원한다. 운영 체제 환경 층(operating system environment layer)은 일반적인 운영 체제의 서비스를 제공하는 층으로 화일 시스템, 장치 구동기, 그리고 응용 환경 층의 메타 객체로서의 역할을 담당한다. 기본 객체 층(base object abstraction layer)은 시스템 전반에 대하여 객체의

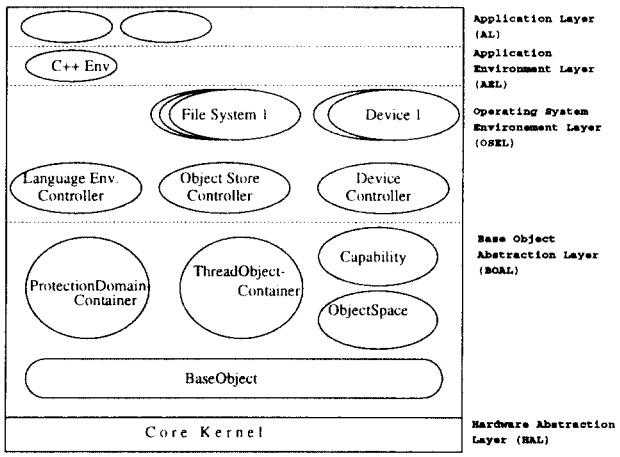

그림 1. SNU ROSE 프로토타입 시스템의 구조

기능을 제공하고 시스템내의 각 객체들에 대한 기본적인 프리미티브를 제공하여 상위 시스템에서 응용 객체를 효과적으로 재구성할 수 있도록 지원하는 기능을 제공한다. 이 층의 가장 기본적인 객체는 BaseObject이다. BaseObject는 객체의 본형 및 인스턴스에 대한 정보를 유지 관리하며 각 객체에 대한 메타 계층 구조(meta hierarchy)를 유지한다. 또한, 각 객체에 대한 메타 연산의 요청시 해당 메타 객체로 하여금 메타 연산을 수행하도록 지시한다. ProtectionDomainContainer객체는 가상 주소 공간을 나타내는 ProtectionDomain 객체를 제공하며, 페이저의 역할을 담당한다. ThreadObjectContainer객체는 스케쥴러의 역할을 담당하고 스케쥴링의 단위체인 ThreadObject를 제공한다. 하드웨어 추상화 층(hardware abstraction layer)[4]은 상부 층에서 하드웨어와 무관한 정책을 구현할 수 있도록 하드웨어의 기능을 객체화된 인터페이스를 통하여 제공한다.SNU ROSE의 기본적인 서비스인 응용 프로그램 및 데이타를 표현하기 위한 객체는 BaseObject에 의하여, 객체를 사상할 수 있는 가상 주소 공간은 ProtectionDomainContainer에 의하여, 그리고 객체를 수행하기 위한 쓰레드는 ThreadObjectContainer에 의하여 각각 제공된다.

Ⅴ. 객체의 동적 재구성 지원 구조

본 절에서는 객체의 환경 및 내용의 재구성을 지원하기 위하여 SNU ROSE에서 채택한 객체의 구조 및 재구성 지원 기법을 기술한다.

1. 객체와 객체의 환경

(1) 객체

SNU ROSE의 객체는 BaseObject에 의하여 유지 관리된다. BaseObject는 객체의 본형과 인스턴스의 관리를 위하여 ProtoObj 클래스를 사용한다. ProtoObj 클래스는 객체의 관리를 위한 정보 및 메타 객체와의 인과적 접속 관계 정보를 유지하며 그 내용은 그림 2에 나타나 있다.

ProtoObj는 자신의 타입, 인과적 접속 관계, 접근 자격, 가상 주소 공간등의 정보를 갖는다. 가상 주소 공간 정보는 본형과 인스턴스를 구별하는 정보로서, 본형의 경우 해당 객체가 사용할 수 있는 가상 주소 공간의 범위만이 설정되나, 인스턴스의 경우는 상태 정보를 저장할 수 있는 물리적 페이지 정보가 추가된다. 인과적 접속 정보는 메타 객체에 대한 정보로서, 본형의 경우 객체의 최초 생성시 설정되고, 인스턴스의 경우 새로운 인스턴스의 생성시 본형으로부터 상속받거나 재구성을 위하여 자기 자신 혹은 다른 객체에 의하여 동적으로 설

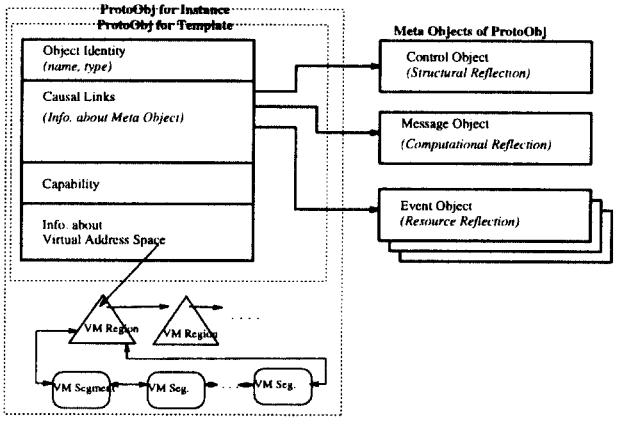

그림 2. ProtoObj의 구조및 환경

정된다.

(2)메타 객체

그림 2에서 보는 바와 같이 ProtoObj는 자신의 메타 객체로서 5개의 객체 인스턴스와 인과적 접속 관계를 갖는다. 각 메타 객체는 특정 메타 연산을 해당 객체에 수행할 수 있으며, 해당 메타 연산을 통하여 객체의 수행 환경을 구성한다.

제어 객체(*control object*)는 인스턴스 생성 및 제거,인스턴스의 공유 및 재구성등의 구조적 반영 기능을 제공하는 객체이다. 메시지 객체(*message object*)는 계산적 반영 기능을 제공한다. 메시지 객체는 객체에 대한 간접 호출의 제어, 객체의 재구성된 메소드의 수행, 대리자 객체로의 메시지 전송등의 역할을 담당한다. 사건 객체(*event object*)는 해당 객체에 대하여 발생하는 사건의 처리를 담당하는 객체이다. 본 논문에서 제시한 시스템에서는 자원 관리 정책에 대한 보다 탄력적인 재구성 기능을 제공하기 위하여 자원적 반영을 제공하는 사건 객체를, 메모리 공간에 대한 사건 객체(event object for space), 수행에 대한 사건 객체(event object for activity), 운영 체제 환경에 대한 사건 객체(event object for operating system environment)로 세분하였다. 메모리 공간에 대한 사건 객체는 주소 폴트(address fault)에 대한 해결을 담당한다. 메모리 공간에 대한 사건 객체의 대표적인 구현은 그림 1의 ProtectionDomainContainer 객체이다. 수행에 대한 사건 객체는 일반적인 운영 체제의 스케쥴러 기능을 제공한다. 그림 1의 ThreadObjectContainer 객체는 수행에 대한 사건 객체의 대표적인 예이다. 운영 체제 환경에 대한 사건 객체는 일반적인 운영 체제의 서비스를 제공하는 사건 객체로서 응용 객체에 대한 사건 객체의 경우 화일 시스템, 네트웍 기능등의 서비스를 제공하는 객체이다.

2. 객체 내용의 재구성 지원

객체 내용의 동적 재구성은 객체의 인스턴스에 대하여 상태 정보 및 메소드의 실 구현을 변경하는 동작을 의미한다. SNU ROSE에서는 객체 내용의 재구성 기능을 운영 체제 커널에서 제공하는 것이 아니라 객체의 관리자인 메타 객체, 특히 제어 객체에 의하여 제공되는 구조이다.

제어 객체는 객체 인스턴스의 생성과 객체 내용의 동적 재구성을 담당한다. 따라서, 제어 객체에 의하여 동적 재구성을 용이하게 지원하기 위하여 SNU ROSE의 BaseObject는 객체의 인스턴스를 특정 가상 주소 공간 - ProtectionDomain - 내에 국한하지 않고, 상황에 따라서 다른 주소 공간에 사상할 수 있는 서비스를 제공한다. 이 기능은 객체의 관리자인 제어 객체로 하여금 재구성의 대상이 되는 목표 객체의 인스턴스를 다른 가상 주소 공간으로부터 자신이 사상되어 있는 가상 주소

공간으로 사상하여 해당 객체를 자신의 가상 주소 공간 내에서 조작할 수 있도록 지원한다.

상기한 객체의 가상 주소 공간에 대한 제어 기능은 다음과 같은 BaseObject의 메소드들에 의하여 제공되며, 제어 객체에 의하여만 이용될 수 있다.

```
kern_ret_t MapObject(ObjInstanceID_t, ProtectionDomainID_t,
                     VaddrHint_t);
kern_ret_t UnMapObject(ObjectInstanceID_t, ProtectionDomainID_t);
```

MapObject() 메소드는 주어진 객체의 인스턴스를 특정 가상 주소 공간의 특정 주소에 사상하는 기능을 수행한다. MapObject()의 인자들은 먼저 목표 객체의 인스턴스를 지정하는 ObjectInstanceID_t형의 객체 식별자(identifier)와 목표 가상 주소 공간을 지정하는 ProtectionDomainID_t형의 객체 식별자, 그리고 가상 주소 공간내에 객체를 사상할 가상 주소 - VaddrHint_t형 - 로 구성된다. 단, 지정된 가상 주소 공간은 반드시 MapObject()를 호출한 객체가 위치한 가상 주소 공간이어야 하며, 이 때, MapObject() 메소드는 지정된 가상 주소 공간의 페이지 테이블에 해당 객체의 페이지 정보를 등록하여 목표 객체를 접근할 수 있도록 한다. UnMapObject() 메소드는 MapObject()연산의 반대 연산으로, 주어진 가상 주소 공간내의 객체 인스턴스에 대한 가상 메모리 사상 정보를 삭제한다. 그림 3은 특정 객체의 관리자인 제어 객체가 대상 객체의 재구성을 위하여 MapObject()와 UnMapObject()메소드를 사용하여 객체 인스턴스를 자신의 가상 주소 공간으로 사상한 다음 그 내용을 변경하는 예를 보이고 있다.

따라서, 특정 객체의 재구성을 원하는 제어 객체는 다음의 의사 코드에서 보인바와 유사한 형태의 단계로 작업을 수행하게 된다.

```
retval = MapObject(TargetObjectID,
         MyProtectionDomainID, SOME_ADDRESS);
// reconfigure the object's contents
. . . . . .
//
retval = UnMapObject(TargetObjectID, MyProtectionDomainID);
```

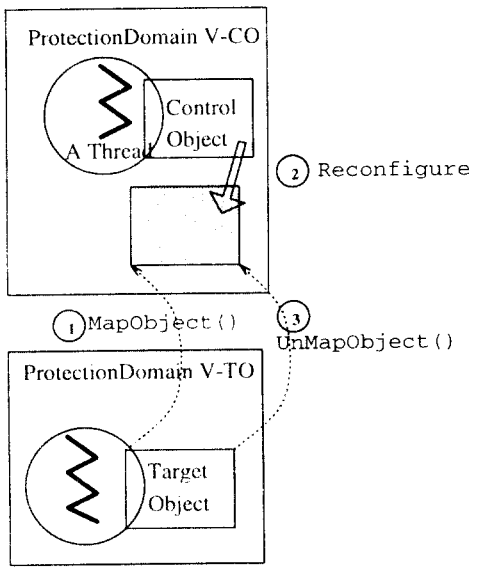

그림 3. 제어 객체에 의한 객체 내용의 재 구성

3. 객체 수행 환경의 재구성 지원

객체의 수행 환경에 대한 재구성은 객체의 관리 정책을 상황에 맞도록 변화시키는 작업이다. 예를 들면, 페이징 기법 혹은 스케쥴링 기법의 변경이 대표적인 예가 될 수 있다. SNU ROSE에서는 사건을 해석하여 해당 객체에 대한 메타 연산으로 적용하는 방법을 사용하고 있다. SNU ROSE에서는 객체에 대한 수행 환경을 객체에 대한 메타 환경으로 정의하고 객체 수행 환경에 대한 재구성이 메타 객체의 교환(switching)에 의하여 수행되는 구조를 제공한다.

객체의 메타 객체 정보는 객체의 본형 생성시 설정되고, 객체 인스턴스는 본형으로 부터 메타 객체 정보를 상속(inherit)받는다. 메타 객체의 변경은 BaseObject에 의하여 처리되며, 이를 위하여 BaseObject는 응용 수준에 다음과 같은 인터페이스를 제공한다.

```
kern_ret_t SetMetaInfo(ObjectID_t, MetaObjSelector_t,
ObjInstanceID_t);
```

응용이 메타 객체의 변경을 요구하면, BaseObject는 자신이 관리하는 객체의 메타 정보를 갱신하며, 동시에

BaseObject는 변경 이전의 메타 객체와 새로운 메타 객체에 해당 객체의 관리가 이전된다는 사실을 통보하여 메타 객체들로 하여금 자신이 관리하는 객체들에 대한 정보를 갱신하도록 한다.

그림 4는 SNU ROSE에서 쓰레드 서비스를 제공하는 ThreadObject의 수행에 대한 사건 객체를 변경할 시 BaseObject에 의하여 수행되는 작업을 보이고 있다. 그림 4에서는 스케쥴링 기능을 제공하는 ThreadObjectContainer형의 객체 A와 B가 시스템에 존재하고, A에 의하여 관리되는 ThreadObject형의 객체 C의 메타 객체를 A에서 B로 변경하는 예를 보이고 있다. 이때, BaseObject는 'SetMetaInfo(C의 ID, EVENTOBJ_TIMER, B의 ID)'의 형태로 객체 C의메타 객체 변경이 요구되면, 현 메타 객체인 A에 C 객체의 메타 객체 변경 사실을 통보한다. 객체 A는 자신의 스케쥴링 큐에서 C를 제거하고, 객체의 스케쥴링 정보를 Hint정보로 변환하여, BaseObject에 해당 객체를 복귀한다. BaseObject는 객체 C와 Hint정보를 객체 B에 전달하고, 객체 B는 자신의 스케쥴링 큐에 객체 C를 적절히 삽입하여, 스케쥴링 될 수 있도록 한다.

Ⅵ. 관련 연구와의 비교

SNU ROSE의 객체 개념은 Lipto[9,10], Distributed Shared Repository(DSR)[7]에서 채택한 것과 유사하다. DSR의 경우 객체를 상태 정보, 메소드의 실 구현 및 이를 수행하는 쓰레드를 모두 포함하고 있으며, 객체의 수행 환경은 커널에 의하여 제공되는 구조이다. 그러나, SNU ROSE의 객체는 상태 정보와 메소드의 실 구현만으로 구성되며, 객체의 수행 환경은 다수의 메타 객체로 구성되어, 메타 객체의 교환에 의하여 객체의 수행 환경이 쉽게 재구성될 수 있는 구조를 제공한다. Lipto에서 제공하는 객체는 SNU ROSE의 객체와 매우 유사하나, 객체의 수행 환경에 대한 고려를 하지 않았다. 따라서, 5.3절의 예와 같이 객체 수행 환경의 재구성을 지원하는 것은 불가능하다. SNU ROSE의 반영적 구조에 의한 객체의 관리 정책은 객체의 수행 환경에 대한 효과적인 재구성 기능을 제공할 수 있으며, 이를 통하여 응용에 최적화된 수행 환경을 제공할 수 있다.

Apertos[13]에서 제공하는 객체는 객체의 실 구현과 객체를 수행하는 쓰레드, 그리고 객체가 사상된 가상 주소 공간으로 구성된다. 특히, 운영 체제에서 인식하는 각 객체마다 객체를 수행키 위한 쓰레드가 할당된 형태를 갖는다. 따라서, 객체에 대한 호출은 항상 프로세스간 통신을 이용해야만 한다. 이와 같은 메시지 통신 구조는 객체의 호출시 메시지에 의한 부담 및 문맥 교환(context switch)을 수반하여 성능의 저하를 유발할 수 있다. 객체와 수행 쓰레드가 항상 부가된 형태는 운영 체제 수준의 객체를 재구성하는 경우 다음과 같은 문제점을 낳을 수 있다. 운영 체제에 새로운 스케쥴링 기법을 제공하는 스케쥴러 객체를 첨가한다고 가정하면, 스케쥴러 객체도 하나의 객체이므로 스케쥴러 객체에 대한 호출 - 예를 들면, 새로운 쓰레드로의 스케쥴링 - 이 발생할 때마다 스케쥴러 객체의 쓰레드로 문맥 교환을 하여야만 한다. 이는 스케쥴링이 요구될 때 마다 추가의 문맥 교환이 수반되므로 성능의 저하를 가져오게 된다. SNU ROSE에서는 객체의 인스턴스와 객체를 수행하는 쓰레드 서비스가 별도로 제공되므로, 새로운 스케쥴러 객체를 첨가한 경우 스케쥴링이 요구되었을 때 현재 수행중인 쓰레드의 문맥에서 스케쥴링 함수를 직접 수행할 수 있으므로 상기한 문제점이 발생하지 않는다.

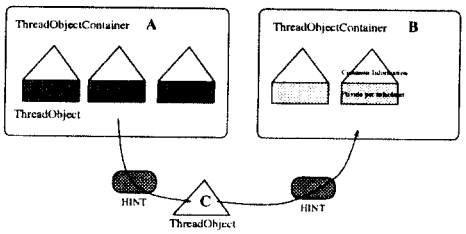

그림 4. 스케쥴러 메타 객체의 변경

표 1. 타 시스템과의 비교

	객체 내용 재 구성 지원 구조	자원 관리 정책 재 구성 지원 구조
Mach	No	일부 서비스 재구성 가능
Apertos	Possible	메타 객체의 교환
DSR	Yes	No(embedded in kernel)
Lipto	Yes	No(embedded in kernel)
ROSE	Yes	메타 객체의 교환

SNU ROSE 시스템과 기존의 객체 지향 운영 체제의 재구성 기능에 대한 비교를 표 1에 보였다.

Ⅶ. SNU ROSE의 구현 현황

현재 SNU ROSE 시스템은 IBM PC호환 기종에서 약 40,000 라인의 C++와 1,500라인의 어셈블리언어를 사용하여 하드웨어 추상화 층과 기본 객체 층은 구현이 완료되었으나, 기타 계층의 객체는 현재 구현 중에 있으며, 커널내 객체들과 메타 객체들의 관련성은 그림 5와 같다. SNU ROSE 시스템은 구현의 편의상 운영 체제 환경 층, 기본 객체 층, 하드웨어 추상화 층을 커널의 가상 주소 공간에 전부 포함하였으며, 응용 환경 층의 객체는 커널 객체에 대한 proxy객체들과 응용 프로그램이 함께 링킹되어 수행되는 것을 가정하고 있다.

현재, 운영 체제 환경 층, 기본 객체층의 객체들에 대한 제어 객체 및 운영 체제 환경에 대한 사건 객체는 BaseObject가 담당한다. 스케쥴링 기능은 ThreadObjectContainer객체의 인스턴스 중의 하나인 stdThreadContainer에 의하여 제공되고, 커널 가상 주소 공간은 ProtectionDomain객체의 인스턴스인 stdKernProtDomain에 의하여 유지되고, stdKernProtDomain객체의 메모리 공간에 대한 사건 객체는 ProtectionDomainContainer객체의 인스턴스인 KernelProtDomContainer객체에 의하여 유지된다. 또한, 객체에 대한 메타 연산의 요구는 사용자 수준의 요구의 경우는 시스템 호출의 형태로, 그리고 인터럽트나 트랩등의 사건의 경우는 BaseObject가 사건을 해석하여 해당 객체에 메타 연산을 요구하는 기법을 사용하였으며, 그 방식은 그림 6에 나타나 있다.

SNU ROSE 기본 연산의 성능은 표 2에 나타나 있다. 측정 대상 하드웨어는 20MHz의 인텔 80386 처리기를 장착한 IBM PC호환 기종이며, GNU C++ 컴파일러를 사용하였다. 표 2의 사건 전달 시간은 발생한 사건을 기본 객체에서 해석하여 메타 객체로 전달하기 전까지 소요된 시간이며, 메타 연산 전달 시간은 사건 발생으로부터 해당 메타 객체로 메타 연산이 전달되기까지 소요된 시간으로 사건 전달 시간에 메타 객체로의 제어 전이 시간을 합한 값이다.

객체 수행 환경의 재구성 및 객체 내용의 재구성 시간의 성능 측정 결과는 표 3에 나타나 있으며, 각 결과는 5.3절의 예와 새로운 객체를 커널에 첨가하는 경우에

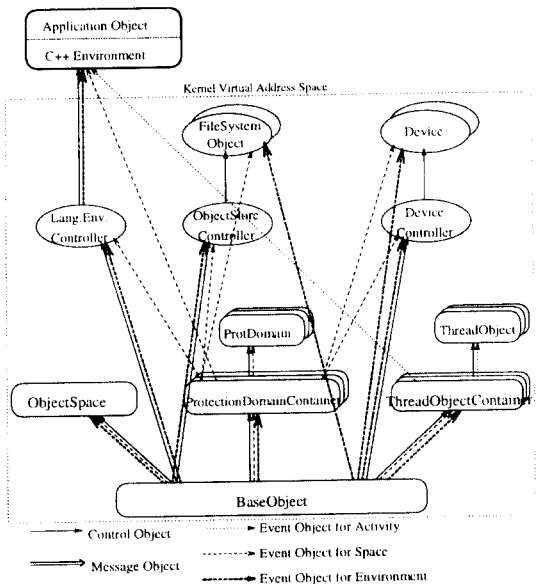

그림 5. SNU ROSE 프로토타입 시스템내 객체의 계층 관계

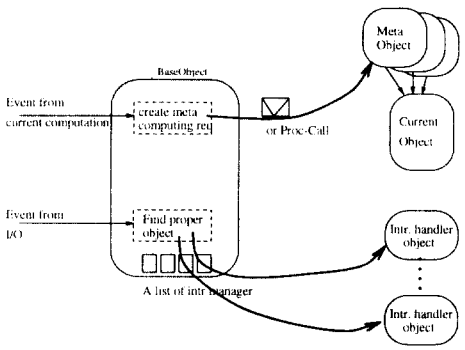

그림 6. SNU ROSE의 사건 처리 기법

표 2. 사건 전달 및 메타 연산 전달 시간

측정 내용		사건(μ sec)
사건 전달 시간		75-80
메타 연산	Intra ProtectionDomain	200-250
전달 시간	Inter ProtectionDomain	520-550
객체 본형 생성 시간		460-800
객체 실체화 시간†		830-1100

† BaseObject에서 해당 연산 수행 시간

표 3. 객체 재 구성 시간

측정 내용	사건(μ sec)
객체 수행 환경 재 구성	540-620
객체 내용 재 구성†	2400-3000

† NILL객체를 추가한 경우의 시간임

대한 성능 측정의 결과이다. 5.3절의 예는 객체 환경 재구성의 예로서, 표 3의 시간(540-620)은 ThreadObject에 대한 메타 객체를 변경할 때 소요되는 시간을 표시한다. 표 3의 시간(2400-3000)은 객체 내용의 재구성시 소요된 시간으로서 커널내에 새로운 객체를 첨가하는 데 소요되는 시간이다. 단, 동적 재구성을 위하여 운영 체제에서 소모되는 시간만을 측정하기 위하여 객체의 내용이 없는 NULL객체를 추가하는 경우에 대하여 측정하였다. 따라서, 제시된 값은 보조 기억 장치 입출력 및 동적 링킹등에 소요되는 시간은 제외된 값이다.

Ⅷ. 결 론

본 논문에서는 객체와 가상 주소 공간 서비스를 별도로 제공하는 기법과 메타 객체를 이용하여 객체의 수행 환경을 제어하는 기법을 결합하는 방식을 통해 객체 수행 환경의 재구성과 객체 내용의 재구성을 효율적으로 제공할 수 있도록 지원하는 시스템의 구조를 제시하였다.

본 논문에서 채택한 반영적 구조는 객체의 수행 환경이 메타 객체에 의하여 의하여 형성되기 때문에, 응용에 최적화된 수행 환경을 형성하는 작업이 응용의 목적에 맞는 메타 객체의 설정을 통하여 수행될 수 있다. 그리고 자원 관리를 담당하는 객체의 재구성을 통하여 운영 체제 자원 관리 정책의 탄력성 및 적응성을 증가시킬 수 있다. 또한, 객체는 자신의 메타 객체에 의하여 생성되므로 특정 언어에 구애될 필요 없이 다양한 구조 및 특성을 갖는 객체를 구현할 수 있는 장점을 갖는다.

또한, 본 논문에서 제시한 운영 체제의 기본 서비스는 객체와 객체를 실체화할 수 있는 가상 주소 공간, 그리고 객체를 수행키 위한 쓰레드 서비스이며, 이들의 조합을 통하여 다양한 수행 모델을 생성할 수 있다. 특히, 가상 주소 공간과 객체를 별도의 서비스로 제공함으로써 메타 객체가 객체 내용을 재구성할 시 관리 대상이 되는 객체의 상태 정보 및 실 구현에 쉽게 접근할 수 있도록 한다. 본 논문에서 제시한 기법을 사용한다면, 상황에 따른 객체 호출의 최적화가 쉽게 가능하다. 예를 들면, 객체의 개발시 객체의 동작은 외부 인터페이스에 의하여 주어지므로 객체의 호출을 적절히 조절하여 객체에 대한 호출 동작을 추적(trace)해야 할 필요가 있다. 이와 같은 호출의 추적 동작은 객체를 독립적인 가상 주소 공간에 할당하고 호출 기법으로서 IPC기법을 사용한다면, 객체의 호출에 대한 정보를 쉽게 얻을 수 있다. 반면에, 최종 결과물의 경우 각 모듈을 구성하는 객체를 같은 가상 주소 공간에 할당하고 객체의 호출을 가상 주소를 사용하는 형태로 설정한다면 성능의 이득을 얻을 수 있다.

본 논문에서 제시한 운영 체제의 구조는 분산 시스템 환경이나 특정 응용을 위해 특정 구조를 갖는 워크스테이션을 위한 운영 체제로서 적합하게 쓰일 것이라 기대된다. 앞으로의 연구는 특정 응용 환경에 적합한 재구성 기능을 효과적으로 제공하는 메타 객체들의 설계 및 구현과 프로그래밍 모델의 제시에 대한 연구가 이루어져야 할 것이라고 생각된다.

참고문헌

1. A. Armand, M. Gien, M. Guillemont, P.

Leonard, "Toward a Distributed UNIX System-The Chorus Approach", Proc. EUUG, pp.413-431, 1986.
2. M. Bach, The Design of UNIX Operating System, Prentice Hall, 1986.
3. P. Barton, D. McNamee, R. Vaswani, and E. Lazowska, "Adding Scheduler Activation to Mach 3.0", Proc. USENIX MACH III Symp., pp.119-136, Apr., 1993.
4. 유 광현, 김 철민, 박 민규, 조 유근, "객체 지향 기법에 기초한 마이크로 커널의 설계및 구현", 정보과학회 논문지, 20권 10호, pp.1530-1538, Oct., 1993
5. C. Hofmeister and J. Purtilo, "Dynamic Reconfiguration in Distributed Systems: Adapting Software Modules for Replacement", Proc. 13th Int'l Conf. on DCS, pp.101-110, May, 1993.
6. Y. Ishikawa, "Reflection Facilities and Realistic Programming", ACM SIGPLAN Notices, Vol. 26, no 8, Aug., 1991.
7. K. Kato, A. Narita, S. Inohara, and T. Masuda, "Distributed Shared Repository: A Unified Approach to Distribution and Persistency", Proc. 13th Int'l Conf. on DCS, pp.20-29, May, 1993.
8. P. Maes, "Concepts and Experiments in Computational Reflection", Proc. OOPSLA, pp.147-155, Oct., 1987.
9. N. Hutchinson P. Druschel, L. Peterson, "Beyond Micro-Kernel Design: Decoupling Modulariry and Protection in Lipto", Proc. 13th Int'l Conf. on DCS, pp.512-520, Jun., 1992.
10. N. Hutchinson P. Druschel, L. Peterson, "Service Composition in Lipto", Proc. of Int'l Workshop on Object Orientation in Operating System, pp.108-111, Oct, 1991.
11. AT&T, *UNIX System V Release 4: Programmer's Guide: System Services and Application Packaging Tools*, UNIX Software Operation, 1990.
12. T. Watanabe and A. Yonezawa, "Reflection in an Object-Oriented Concurrent Language", Proc. OOPSLA, pp.306-315, Sep., 1988.
13. Y. Yokote, "The Apertos Reflective Operating System: The Concepts and its Implementation", Proc. OOPSLA, pp.414-434, Oct., 1992.

柳 廣 鉉(Kwang Hyun Yoo) 정회원

1985년 : 서울대학교 컴퓨터공학과 졸업
1987년 : 서울대학교 컴퓨터공학과 석사
1987년~현재 : 서울대학교 컴퓨터공학과 박사과정
※주관심 분야 : 운영 체제 및 시스템 소프트웨어

曺 裕 根(Yoo-Kun Cho) 정회원

1971년 : 서울대학교 공과대학 졸업
1978년 : 미국 미네소타대학 전산학과 박사
1979년~현재 : 서울대학교 컴퓨터공학과 교수
※주관심 분야 : 알고리즘, 운영 체제