공학 연구논문집 제26권 제1호 pp. 327~344, 울산대학교
Journal of Engineering Research Vol. 26, No. 1, pp. 327~344 1995.
University of Ulsan.

POSIX 쓰레드를 이용한 SR 실행 환경 지원에 관한 연구

김영곤 · 정영필* · 박양수 · 이명준
전자계산학과

<요 약>

병행성은 컴퓨터 시스템에서 주요한 추상화중의 하나이다. 이러한 병행성은 하드웨어의 디자인과 오퍼레이션, 운영체제, 다중프로세서, 분산계산, 실시간시스템, 프로그래밍 언어와 시스템의 디자인 수단으로 사용되어왔다. 운영체제 서비스에 대하여 응용 프로그램들의 원시코드 이식성을 제공하기 위하여 최근 몇년동안 운영체제에 대한 응용 프로그램 인터페이스를 위한 일련의 표준화 작업으로 IEEE의 운영 체제 기술 협의회는 POSIX를 개발해오고 있다. POSIX 쓰레드 확장에 관한 표준안은 운영체제 서비스가 하나의 프로세스내에서 제어의 다중 쓰레드를 생성하고 수행하는 것을 지원하는 응용 프로그램 인터페이스를 지원한다.

우리는 이러한 POSIX 쓰레드를 사용하여 SR 언어의 실행지원 시스템을 새로이 설계 및 구현한다. 개발된 실행지원 시스템의 이식성은 POSIX 쓰레드를 지원하는 시스템들 사이에서 자연스럽게 보장된다. 주어진 컴퓨터 시스템에 대하여 이 새로운 실행지원 시스템의 성능은 그 시스템에서의 POSIX 쓰레드 자체의 구현에 대한 질에 좌우된다. 특히 그 시스템이 많은 프로세서를 가지고 이러한 프로세스들을 효과적으로 지원하도록 POSIX 쓰레드가 구현되었을때 이 새로운 실행지원 시스템은 매우 유용하게 된다.

Supporting SR Runtime Environment with POSIX Threads

Young-Gon Kim · Young-Phil Cheung* · Yang-Su Park · Myung-Joon Lee
Dept. of Computer Science

<Abstract>

Concurrency is one of the key abstractions in Computer System. It has relevance to hardware design and operation, operating systems, multiprocessor and distributed

* 부산 전문대 전자계산기과

computation, real-time systems, programming languages and system design methods. To provide source code portability of operating system services for applicaion programs, POSIX is being developed for the recent several years by the IEEE Technical Committee on Operating System as a family of standards for application program interface to an operating system. The draft standard POSIX Threads Extension provides an applicaion program interface to operating system services supporting the creation and execution of multiple threads of control within a single process.

Using this POSIX Threads, we present a new design and implementation of the runtime support system of SR. Naturally, the portability of the runtime support system developed is guaranteed between the systems supporting POSIX Threads. For a given system, the performance of the new runtime support system depends on the quality of the implementation of POSIX Threads for the system. In particular, when the system has a lot of processors and the implementation of POSIX Threads utilizes those processors effectively, the new runtime support system is very useful.

1. 서 론

병행성은 컴퓨터에서 가장 주요한 추상화중의 하나이다[10]. 이러한 병행성은 하드웨어의 디자인과 오퍼레이션, 운영체제, 다중프로세서, 분산계산, 실시간시스템, 프로그래밍 언어와 시스템의 디자인 수단으로 사용되어왔다. 따라서 대부분의 컴퓨터 시스템 환경이 자연히 병렬적으로 수행될 수 있어야 한다. 비록 일반적인 폰 노이만 컴퓨터가 순차적인 관점으로 수행을 계속하지만 컴퓨터 과학자와 소프트웨어 공학자는 이러한 것을 규칙이라기 보다는 예외로 보아야 한다[10]. 최근에 와서 네트웍 발달로 인한 분산처리 시스템의 대두와 멀티프로세서(multiprocessor) 및 병렬 프로세서(parallel processor)의 등장으로 보다 많은 병행성을 지원할 수 있게되어 성능의 진전을 현저하게 가져왔다. 그러나 단순히 하드웨어의 성능이 높아졌다고 해서 소프트웨어의 성능이 그 정수배 만큼 높아진 것은 아니다. 그러한 하드웨어의 환경을 충분히 이용할 수 있는 소프트웨어의 부족 특히 병행성을 요구하는 응용 프로그램을 개발하기 위한 도구인 병행 언어의 부족으로 인해 풍부한 하드웨어 자원을 최대한으로 이용하지 못하고 있다.

따라서 병행성 언어의 도입은 실존하는 문제를 보다 자연스럽게 모델화 할 수 있게 해주며 프로그램의 수행성능 또한 향상할 수 있도록 해주는 수단을 제공해준다. 더 나아가서 이러한 병행성 언어의 도입은 실시간 처리를 위한 개념이나 실시간언어의 설계에 있어서 그 기초가 된다.

이미 외국의 경우는 수십년 전부터 병행성을 지원하는 언어들을 개발해 오고 있으며 실제 많은 언어들이 꾸준히 개발되었다. 하지만 국내에 있어서는 복잡성과 여러가지 제한적인 요소때문에 병행성 언어의 설계에 대한 시도가 미흡한 실정이다.

병행 언어를 개발한다는 것은 단순히 그 언어의 문법을 작성하고 그것의 번역기인 컴파일러만을 설계한다는 의미는 아니다. 이와 아울러 여러 개의 프로세스들이 하나의 프로그램내에서 동시에 실행되도록 지원해 주어야 하며 따라서 운영체제의 프로세스 스케줄러와 같은 기능이 필요하다.

내부적으로는 병행성 제어를 위해 상호배제나 프로세스 통신 기법이 제공되어야 할 것이며 분산 환경의 제어를 위한 네트웍 프로그래밍 기법이 지원되어야할 것이다. 결국 이는 컴파일러 이외에 실행 지원(Run-Time Support:RTS) 시스템이 설계되어야함을 의미한다. RTS의 설계를 위해 프로그래밍 언어 차원이나 커널 차원에서의 프로세스 제어 모델이 필요하다. 현재 개발되었거나 개발된 운영체제들이 실행 단위로 쓰레드 개념을 도입하고 있기 때문에 쓰레드(thread)라는 개념을 통하여 이를 구현하는 것이 바람직 할 것이다[19]. 쓰레드는 문맥교환(context switching)을 할 때 프로세서(processor)의 통제를 받는 프로세스(process:heavyweight process)의 문맥보다는 다소 적은 문맥을 가진다는 의미에서 다소 가벼운 무게의 프로세스(lightweight processes)로 언급되고 있다. 프로세스 제어 모델의 설계는 모델의 특성상 컴퓨터 구조에 의존해야하는 문제점이 있어 특정 시스템에 국한된 설계를 해야하나 다행히도 최근에 쓰레드에 대한 개념을 표준화하기 위한 노력이 성과를 보이고 있다. IEEE 산하의 운영체제를 위한 기술 협의회(TCOS)는 응용 프로그램의 소스코드 이식성을 높이기 위해 다양하게 존재하는 UNIX 운영체제에 대한 표준화된 인터페이스를 확립했다. 그 결과 IEEE는 운영체제의 응용 프로그램 인터페이스를 위한 표준화된 계보를 1003.x의 형태로 POSIX(Portable Operating System Interface Exchange) 표준이라 명시 하였다. 1003.4[23] 인터페이스는 실시간 응용에 주로 필요한 이진 세마포, 프로세스 메모리 잠금, 타이머등과 같은 서비스를 제공하기 위한 것으로서 C 언어 응용 프로그램 인터페이스인 1003.1에 대한 확장이다. 그리고 1003.4a[24] 혹은 Pthreads(POSIX Threads)는 단일 POSIX 프로세스 내에서 제어의 다중 쓰레드 개념을 지원하기 위한 1003.4의 확장이다. 1003.4a에 제공되는 서비스에는 쓰레드 생성, 상호배제, 동기화등이 있다.

본 논문에서는 POSIX 쓰레드를 이용하여 병행언어 SR의 실행 지원 시스템을 설계 및 구현하였다. SR은 C 언어와 유사한 문법을 가지며 다중의 프리미티브를 제공하는 병행언어로서 보다 다양한 형태의 병행 제어를 할수 있다. 이와 같은 SR 언어의 실행지원 시스템을 Pthreads를 사용하여 구현함으로써 보다 이식성이 강한 언어로 만들수 있으며 병행 제어에 필요한 많은 부분이 Pthreads로 흡수되므로 보다 간략한 실행 지원 시스템을 구축할 수 있다. 부가적으로 POSIX 쓰레드의 적용에 대한 안정성 및 신뢰도가 검증될 것이며 다른 응용영역(예" 데이타베이스 관리 시스템, 실시간 처리 시스템등의 설계)에 적용하기 위한 Pthreads의 응용 기술이 확보될수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 병행 언어인 SR의 소개가 기술되며 3장에서는 POSIX의 쓰레드에 대한 특성과 구조에 대하여 논의된다. 4장에서는 POSIX 쓰레드를 이용한 SR 언어의 실행 지원 시스템의 설계 및 구현에 관한 전반적인 내용이 논의 된다. 5장에서는 결론과 본 연구가 가지는 의미와 효과가 논의된다.

2. 병행언어 SR

2.1 SR의 소개

SR은 아리조나 대학의 Gregory R. Andrews등에 의해 만들어진 병행 프로그래밍을 기술하기 위한 언어이다[1,2,3,4,5,6,7,22]. 주요한 언어 구성요소는 공유 프로세스와 변수들을 은닉시키는 리소스(resource)와 프로세스 상호 통신을 위한 기초적인 기법인 오퍼레이션(operation)이다.

SR은 병행 프로그래밍에 관계된 대학과 연구소의 수업 과정이나 연구 과제로 사용되어 왔다. 또한 SR은 병렬 알고리즘,

데이터 베이스, 분산 시뮬레이션등의 연구를 위한 프로젝트의 과정으로 사용되었으며 화일 시스템이나 명령 분석기등과 같은 분산 운영체제의 부분을 연구하는데 사용되었다. SR은 같은 아키텍쳐를 가지는 하나 이상의 네트웍 머신이나 Solaris 2.3 상의 Sun 시스템에서 수행이 된다. SunOS 4.x, HP RISC, DEC Alpha, DECstation, Linux상의 386/486등의 플랫포옴에서는 다중처리가 모사되었다. IBM RS/6000, DGAViiON, VAX등도 같은 방식으로 지원되고 있다. 또 병행성의 검증을 위하여 CCR, monitor, CSP등을 SR 프로그램으로 바꾸는 세 가지의 프리프로세서가 지원되고 있다. 더욱이 그래픽 사용자 인터페이스를 위해 X-윈도우가 지원되도록 X-윈도우 인터페이스를 제공하고 있다. 현재까지 마지막 SR의 구현은 버전 2.3이며 1994년 10월에 발표되었다[6].

2.2 SR 언어 특성

SR은 Hoare의 논리를 바탕으로 하며, 공리 시멘틱스(Axiomatic Semantics)를 확장한 형태와 Dijkstra의 가디드 명령어(Guarded Command)를 결합한 형태이다. 프로그램 문법은 C, Pascal과 논리식을 혼합한 형태이며 순차적인 표기와 병행적인 표기 모두에 대하여 공통적인 표기법을 사용한다. 이것은 공유 메모리를 사용하는 다중 프로세서 환경에서부터 분산 시스템까지 널리 사용될 수 있다. SR 언어는 특정 컴퓨터 구조에 관련되지 않고 프로그래밍할 수 있는 많은 연산을 제공하며, 다른 언어에서 볼 수 없는 뛰어난 기능들을 많이 제공한다. 이처럼 SR은 다양한 기능에 비해 간단한 개념에 기초하여 있기 때문에 배우기가 쉬우며 효율적으로 구현되어 있다.

프로그램은 기본 단위로 리소스(Resource)라는 모듈을 지원한다. 하나의 리소스에는 임의의 함수나 프로세스를 정의할 수 있고 프로세스도 원하는 만큼 생성할 수 있다. 모든 프로세스는 VM이라는 가상 머신 내에서 SR 자체의 프로세스 관리 시스템에 의해 수행이 된다. 하나의 가상 머신은 하나의 실제 프로세스(예" 유닉스 프로세스)와 관련되지만, 한 개의 물리적인 CPU를 가진 컴퓨터에서도 동시에 여러 개의 가상 머신을 생성해 이들을 서로 병행으로 운영하면서 병렬 프로그래밍을 할 수 있다. 또한 여러 개의 CPU를 가진 시스템은 각기 다른 CPU에서 가상 머신을 만들어 이들을 동시에 병렬로 운영할 수 있는 기능이 있다. 네트웍으로 연결된 환경에서는 각각의 컴퓨터 시스템에 다른 가상 머신을 만들어 이들을 동시에 병렬로 운영할 수 있는 기능이 있다. SR은 다중의 주소 공간(multiple address space)내에서 실행할 수 있다. 즉 다중의 물리적인 머신 상에서도 수행할 수 있다. SR은 세마포가 기본적인 연산으로 제공되며, Ada에서 제공되는 랑데부도 자연스럽게 구현할 수 있다. SR은 고 수준 언어로서 컴파일시에 타입을 검사하며 컴퓨터 언어로서 필요한 대부분의 기능이 라이브러리 형태로 제공된다. 즉 SR은 병행 프로그래밍을 위한 많은 기능을 비롯해서 풍부한 수학적인 함수나 X-윈도우를 위한 라이브러리등도 제공한다.

2.2.1 SR 컴파일러

SR 언어의 실행지원 시스템을 Pthreads로 구현하기 위해서는 SR 언어가 병행성을 지원하기 위하여 어떠한 형태의 코드가 생성되어 실행 지원 시스템과 결합하는지 알아야 한다. SR 컴파일러는 SR 프로그램을 컴파일하여 C 코드를 중간 출력으로 내 놓는다. 이를 다시 C컴파일러에 의해 실행 모듈을 만들 수 있으며, 이를 실행 라이브러리와 결합하여 최종적으로 실행 모듈을 생성한다. 최종적인 실행 모듈은 순수 SR 프로그램의 기능을 수행하기 위한 코드 부분(MC)과 프로세스 단위의 제어를 하기 위한 RTS 부분으

로 이루어져 있다. SR 언어의 번역 과정을 그림 1로서 나타낼 수 있다.

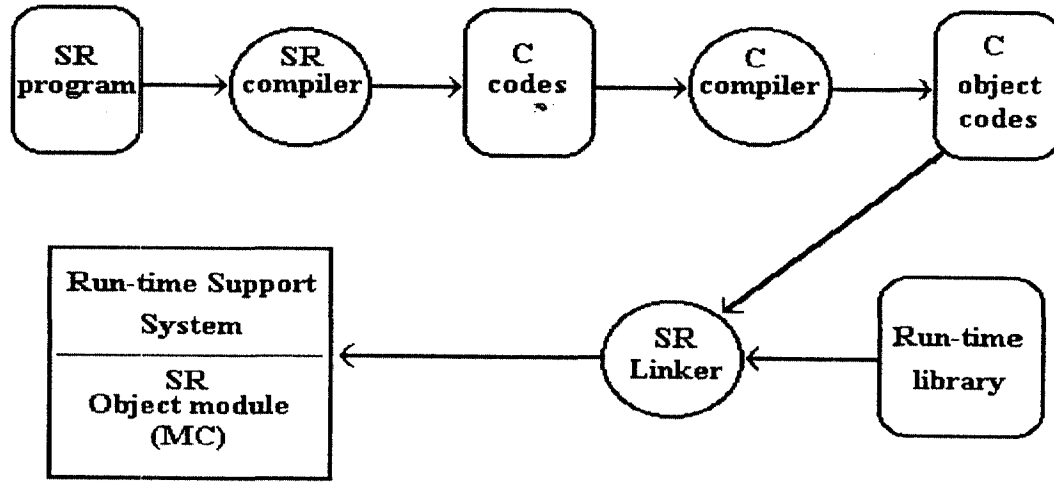

그림 1. SR 언어의 번역 과정

2.2.2 SR 실행 지원 시스템

POSIX 쓰레드를 이용하여 효율적인 SR 실행 지원 시스템을 설계하기 위해서는 현재의 SR 실행 지원 시스템을 정확히 이해하는 것이 필수적이다. 이 부분의 연구는 이미 수행이 된 상태이다[12]. 현재의 실행 지원 시스템은 다음과 같이 개괄적으로 설명될 수 있다.

SR의 계산 모델은 프로그램을 가상 머신(VM)이라 불리는 하나 이상의 주소공간으로 분리할 수 있도록 한다. 각 가상 머신은 하나의 물리적인 머신 상에서의 주소 공간을 정의하며 정적으로 생성될 수 있다. 가상 머신은 두 가지 유형의 모듈 구성요소를 가진다. 즉 글로벌(global)과 리소스(resource)이다. 이들 구성요소는 사양(spec)과 실제구현(body)의 두 부분으로 구성되어 있으며 가상 머신은 단지 하나의 글로벌만을 가질 수도 있다. 글로벌이나 리소스의 사양은 타입, 상수, 오퍼레이션등의 선언을 포함한다. 이러한 오브젝트들은 오퍼레이션들을 서비스하는 코드들을 포함한다. 그 코드는 processes와 procs라 불리는 단위들로 나뉘어진다. 프로세스들은 글로벌이나 리소스에 대한 생성이 끝날 때 암시적으로 생성된다. 대신에 proc 들은 그들이 호출될 때 생성되고 독립적인 프로세스로 실행한다. 글로벌이나 리소스내에서 생성된 모든 process들은 그 글로벌이나 리소스가 실행을 종료하면서 모든 프로세스가 동일한 가상 머신 상에서 실행한다.

한 프로그램은 하나의 물리적인 머신상에서 실행되는 하나의 가상 머신으로 구성되며 이는 다중의 물리적인 머신들 상에서 실행되는 다중의 가상 머신들로 구성될 수도 있다. 데이타와 프로세서들은 하나의 가상 머신 내에서 공유하게 된다. 가상 머신상의 프로세스들은 오퍼레이션 호출을 통해서 통신한다. 오퍼레이션들은 오퍼레이션의 선언된 이름이나 resource capability 변수를 통해서 직접적으로 호출될 수도 있다. 이러한 capability 변수들

은 strong typing이 지원되며 구조체적으로 동일한 표시들을 가지는 오퍼레이션들에 대한 포인터이다. 프로세스 사이의 통신은 가상 머신의 위치와는 무관하다. 즉 동일한 리소스들 사이의 메시지 전달이나 서로 다른 가상 머신들 사이에 있는 프로세스 사이의 메시지 전달은 같은 표기법을 가진다. 이 언어는 공유 변수들, 비동기적인 메시지 전달, 랑데부, 리모트 프로시저, 다중 캐스트(multicasting)를 포함해서 다양하고 신뢰성 있는 메시지 전달 구조를 지원한다. 이처럼 SR은 많은 통신모델을 지원하는 것을 목표로 하였으며 언어 구조가 여타의 언어들과 상이한 큰 언어이지만 배우기가 쉽다[9]. SR은 orthogonal 설계를 사용함으로서 분산 및 병렬 프로그래밍의 개념의 수를 줄이고 있다.

RTS의 역할은 다음과 같이 기술될수 있다. RTS는 머신 코드가 실행될 수 있는 환경을 제공해 준다. 또한 리소스의 생성과 파괴, 오퍼레이션의 호출과 제공, 그리고 메모리의 할당을 위한 도구를 제공한다. 내부적으로 RTS는 개별적인 프로세스와 세마포를 제공하는 집합체로 여겨진다. 네트웍에 관련된 수행은 SR 네트웍 인터페이스(SRX) 부분이 담당한다. RTS가 다른 머신에 대한 서비스를 요청받으면 리소스를 생성하거나 오퍼레이션을 수행한다. RTS는 간단히 요청을 호출 블록(invocation block)을 통해 해당 머신으로 넘겨주기만 한다. 해당 머신에 도달했을 때, RTS가 해당 머신 내에서 지역적으로 생성되었다 하더라도 RTS는 그 요청을 처리할 수 있다. 이러한 요청의 수행결과는 이와 유사한 방법으로 되돌려 진다. 이상의 SR RTS의 계산 모델은 그림 2와 같다.

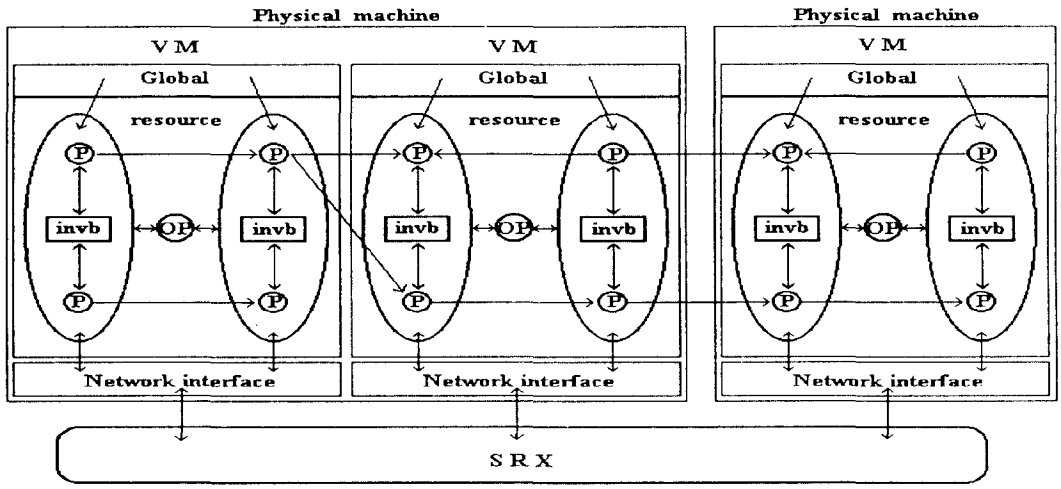

그림 2. SR 실행지원 시스템의 계산 모델

- VM : Virtual Machine
- OP : operation
- SRX : SR network manager
- P : process
- invb : invocaion block
- global and resource

3. POSIX Threads(Pthreads)

3.1 Pthreads의 소개

쓰레드는 실행 제어의 독립적인 순차적 흐름이다[11]. 쓰레드는 다른 쓰레드와 공통의 가상 주소공간을 공유한다는 점에서 프로세스들과는 차이점을 가진다. 쓰레드는 단일 프로세서나 다중 프로세서를 위한 계산 구성 블럭(computational building block)으로서 널리 받아들여져 왔다. 단일 프로세서 환경에 있어서 쓰레드 모델은 비동기 오퍼레이션을 프로그래밍 하는 것이 단순해진다. 다중 프로세서 환경에 있어서 쓰레드는 여러 개의 프로세서를 이용함으로써 보다 높은 처리량을 낼 수 있도록 한다.

값싼 병행성을 지원하는 아이디어는 Mesa 프로그래밍 언어의 코루틴, Ada 프로그래밍 언어에 있어서 다중 테스킹등을 포함해서 오랫동안 여러 가지 형태로 존재해 왔다. Pthreads(POSIX Threads)는 C-언어 프로그램을 위한 유사한 기능을 제공할 의도로 제안되었다. 이 것은 C-threads[13], Mach threads, Brown University threads[14]등 몇몇 가지의 충분한 경험을 바탕으로 한 것이다. 그리고 Lynx[17], Sun[25], Chorus[8]등의 상용 운영체제들은 다중-쓰레드된 프로세스들을 지원한다.

Pthreads(POSIX Threads)는 이러한 시스템을 사용하여 본 경험의 바탕에서 도출된 IEEE 표준이며 POSIX.1의 "Real-Time" 확장이다. POSIX 제안은 미국 정부 및 국제적 표준(ISO/IEC)으로 채택될 전망이어서 미래의 실시간 응용 프로그램에 큰 영향을 미칠 것으로 생각되고 있다. 본 논문에서는 Florida State University에서 구축한 Pthreads 라이브러리(FSU Pthreads Library)[21]를 이용한다.

3.2 Pthreads 구조

Pthreads 표준은 다중 프로세서에서의 공유 메모리 응용 프로그램, 실시간 시스템의 환경, 단일 프로세서 상에서 다중 쓰레드 프로그램에 대하여 모두에 단일화된 기본 기능을 제공한다. Pthreads의 구현은 다음과 같이 할 수 있다.

- 모든 기능이 운영체제의 일부분인 커널로의 구현.
- 모든 기능이 사용자 프로그램의 일부분이거나 링크될 수 있는 라이브러리로의 구현.
- 위 두 가지 경우를 혼합한 구현.

커널 구현인 경우 쓰레드 오퍼레이션이나 시그널 조작에 대한 제어를 간단히 할 수 있으나 호출할 때마다 커널에 진입하고 빠져 나와야 하는 부담이 있다. 라이브러리 구현인 경우 운영체제 커널로 진입하는 동작이 없기 때문에 매우 효율적이지만 시그널 조작과 몇몇 쓰레드 오퍼레이션이 매우 복잡해진다. 또한 프로세스들을 위한 커널 수준의 스케줄러와 쓰레드를 위한 라이브러리 수준의 스케줄러등 서로 다른 두 가지 스케줄러를 다루어야 한다. FSU Pthreads는 라이브러리 구현이며 Pthreads 라이브러리의 software layer는 그림 3과 같이 표현될 수 있다.

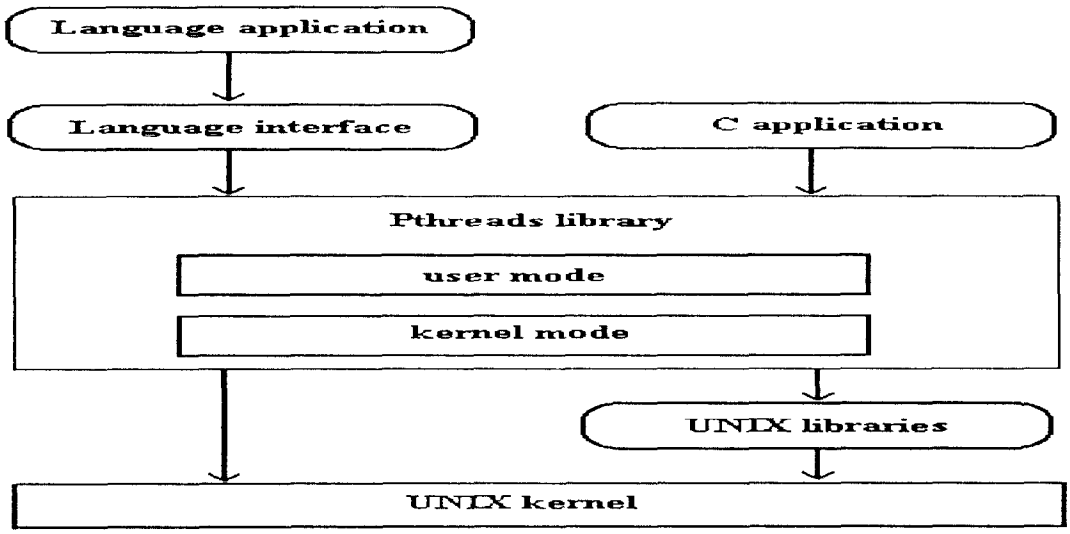

그림 3. Pthreads Software Layer

인터페이스는 프로그램이 Pthreads 서비스를 이용할 수 있도록 한다. C 언어인 경우 Pthreads의 라이브러리 루틴은 인터페이스가 필요없이 이용 가능하다. 다른 언어인 경우는 Pthreads가 인자에 대한 정돈과 타입의 변환, 그리고 다른 언어나 컴파일러에 의존적인 사항때문에 해당 언어의 인터페이스가 필요하다. Pthreads 라이브러리는 POSIX 표준에 의해 정의된 인터페이스와 함수들에 대한 루틴이 포함되어 있다. Pthreads 루틴의 코드는 부분적으로 사용자 코드로 실행이 되며 임계영역에서는 쓰레드간의 상호배제를 보장하기 위해서 Pthreads 커널모드에서 실행이 된다. 이 Pthreads 라이브러리의 구현은 다음과 같은 특징을 수용하도록 설계되어 있다.

- 선점성 : 라운드 로빈 방식과 같은 스케줄링 정책이나 우선순위를 가진 비동기적 이벤트는 선점형 커널 디자인에 의해서만 지원된다.
- 빠른 문맥교환 : 문맥교환은 한 쓰레드에서 다른 쓰레드로 제어가 넘어가는 것을 의미한다. 쓰레드의 가벼움은 문맥교환의 오버헤드를 줄여 준다.
- 작은 임계영역 : 임계영역에서 보내야 할 시간은 가능한한 짧아야 한다. 임계영역으로 진입하고 빠져 나오는 오버헤드도 또한 작아야 한다.
- 무제한적인 스택 증가의 방지 : 인터럽트 핸들러를 수행하고 있는 동안 비동기적인 이벤트가 도착한다면 다른 핸들러는 스택에 이것을 저장할 수 있다.
- 작은 시스템 호출 : 시스템 호출 시간을 매우 소비하는 동작이므로 이것의 사용은 매우 작아야 하며 특히 시그널 조작이나 문맥교환과 같은 시간에 민감한 부분에서는 더욱 줄여야 한다.
- 언어에 독립적인 인터페이스 : 구현은 C 이외의 언어를 위한 동적인 오버헤드가 최대한 작은 Pthreads 인터페이스가 제공되어야 한다.

Pthreads에 의하여 할당된 구조는 비동기적인 이벤트의 처리동안 다른 이벤트에 의해 일관성없이 수정되는 것을 방지해야 한다. 이러한 것을 지원하기 위해서 라이브러리 구현은 라이브러리 코드의 임계영역이 한번에 하나만이 실행되도록 한다.

Pthreads 커널은 커널 플랙을 설정함으로서 진입할 수 있다. 그후에 쓰레드의 내부 데이터 구조에 대한 변경은 다른 쓰레드에 대해서 상호 배제적으로 수행이 되므로 비정상적인 오퍼레이션이 방지될 수 있다. 또 다른 플랙인 디스패처 플랙은 Pthreads 커널을 떠날 때 디스패처가 호출될지를 결정하는 플랙이다. 이 플랙은 새로운 쓰레드가 스케줄링 될 때, 혹은 Pthreads 커널에서 수행되고 있는 동안 시그널이 도착했을 때 설정 된다. Pthreads 커널을 벗어날때 디스패처 플랙이 설정되어 있지 않다면 단순히 커널 플랙을 리셋 시키기만 하면 된다. 그렇지 않다면 다른 쓰레드로 문맥교환이 일어날 수 있도록 디스패처가 호출된다. 일반적인 상황에서 디스패처에 대한 호출은 스케줄링 정책에 따라 수행 가능한 다음 쓰레드를 선택한다. 만약 선택된 쓰레드가 현재 수행중인 쓰레드와 틀리다면 문맥교환이 발생한다.

3.3 Pthreads 인터페이스

3.3.1 쓰레드의 생성

POSIX 쓰레드는 pthread_create() 함수를 호출하여 생성된다. 이때 호출 쓰레드가 수행할 함수에 대한 포인터를 pthread_create()의 인자로 넘겨준다. 이 생성 함수를 호출한 후 해당 스레드는 즉시 수행된다.

3.3.2 쓰레드의 상호배제

상호 배제는 mutex로서 제공된다. 쓰레드가 어떤 공유 리소스에 대해 배타적인 접근을 원한다면 그 쓰레드는 관계된 mutex를 pthread_mutex_lock()이라는 함수를 호출하여 해당 mutex를 소유하게 된다. 만약 다른 쓰레드가 이미 그 mutex를 소유하고 있다면 이 mutex를 요청한 다른 쓰레드는 pthread_mutex_unlock()을 통해 그 mutex를 가지고 있는 쓰레드가 해당 mutex를 풀어줄 때까지 지연을 하게 된다. 같은 mutex에 의해 많은 수의 쓰레드가 지연될 수 있다. 이들 중의 하나는 mutex를 소유하고 있던 쓰레드가 잠금을 풀어주게 되면 해당 mutex를 가지고 수행을 계속하게 된다. mutex는 세마포와 유사하다. 그 근본적인 차이는 mutex를 소유하고 있는 쓰레드만이 그 mutex를 풀 수 있다는 것이다. 하지만 이러한 mutex만을 가지고 쓰레드사이의 일반적인 통신을 하는데는 많은 어려움이 있다.

임의의 한 쓰레드가 현재 잠겨진 mutex를 풀기 위해서 다른 쓰레드에게 신호를 보낼 수가 없다. 이러한 동기화를 위해 조건 변수가 사용된다.

3.3.3 쓰레드의 동기화

공유 자원에 대한 접근을 할때는 주로 mutex를 이용한다. 하지만 임계 영역에서 쓰레드가 오랜 시간을 중지하기 위해서는 mutex만으로 구현 될수 없다. 따라서 이러한 것을 허용하기 위해서 조건 변수가 제공된다. 쓰레드는 조건 변수에 대해서 pthread_cond_wait()를 호출하여 조건이 참이 되는 상태를 기다린다. 다른 쓰레드는 pthread_cond_signal()을 통해 조건 변수에 신호를 보내어 상태를 참이 되게 한다. 이때 두 쓰레드 사이의 동기화를 위해 주로 mutex와 조건변수를 조합하여 사용된다. mutex는 이전에 언급한 바와 마찬가지로 해당 쓰레드의 임계 코드를 수행하기 위해서 사용된다. 즉 기다리는 쓰레드가 mutex를 소유하고 해당 mutex에 블락이 되게한다. 이러한 mutex는 호출되기 전에 잠겨져야 하며 호출과 함께 잠금이 풀리며 호출이 되돌려질 때 다시 잠겨져야 한다. 이것은 쓰레드가 대기해야 하는 상황을 방지해 준다. pthread_cond_signal() 호출은 적어도 하나의 대기중인 쓰레드를 깨우도록 보장해주지만 대기중인 쓰레드 중에서 하나 이상을 되돌려주

도록 하는 것이 더 효율적일 수 있다.

3.3.4 쓰레드당 데이타

Pthreads는 pthread_key_create()에 의해 생성된 키에 기초를 둔, 쓰레드에 관계된 데이터를 위한 인터페이스를 제공한다. 이 함수는 프로세스 내의 모든 쓰레드에 보여질 수 있는 쓰레드에 특정화된 데이터(thread specific data)를 명시한다. 각각의 쓰레드는 pthread_setspecific() 함수를 호출하여 각 키에 쓰레드 특정 데이타를 연관시켜준다. 서로 다른 쓰레드 같은 키 값을 공유할 수 있으며 주어진 키에 관계된 데이타는 각 쓰레드마다 유일하다. 쓰레드는 pthread_getspecific() 함수를 호출하여 제공되는 키에 대한 이 포인터를 찾아낼 수 있다. 이 함수는 호출된 쓰레드에 의해 명시된 키에 관계된 값을 되돌려준다. 이것은 프로그램 전반적이며 각 쓰레드마다의 전역 데이터를 제공해준다.

3.3.5 쓰레드의 스케줄링

Pthreads는 쓰레드의 우선순위 스케줄링을 지원한다. 스케줄링 정책은 프로세스를 위한 1003.4에 정의된 것과 같다. 즉 세 가지의 스케줄링 정책이 지원되고 있다. 이와 관련된 플랙은 SCHED_FIFO, SCHED_RR, ECHED_OTHER이다. SCHED_FIFO는 각 우선순위마다 하나의 FIFO 큐를 가지는 것으로 개념적인 설명이 된다. 실행하고자 하는 쓰레드들은 자신의 우선순위 큐에 들어가야 한다. 가장 높은 우선순위 큐 상의 쓰레드가 실행 우선권을 가진다. 만약 어떤 쓰레드가 더 높은 우선순위의 쓰레드에 의해 선점 당한다면 그 쓰레드는 큐 상의 헤드에 위치하게 된다. 만약 쓰레드가 지연된다면 큐의 맨 끝으로 가게 된다. SCHED_RR은 SCHED_FIFO와 같지만 실행중인 쓰레드가 선점 당할 수 있으며 일정시간 동안 실행한 쓰레드는 큐의 맨 마지막으로 가게 된다는 점이 틀리다. SCHED_OTHER은 구현에 정의된 스케줄링 방법을 사용화는 것이다. 즉 특정 스케줄링 정책이 구현에 의해 제공될 수 있다. 정책과 우선순위는 각 쓰레드 기초 하에 동적으로 변경될 수도 있다.

4. 실행 지원 시스템(RTS) 구현

SR의 주요 구성 요소가 리소스와 오퍼레이션이므로 RTS에 있어서 이들의 개념을 지원하기 위해 연관된 자료 구조를 유지하여 프로세스 제어 기법을 구현하여야 한다. 따라서 본 장에서는 리소스와 오퍼레이션에 대한 구조 및 관리 방식을 언급하며, 이를 근거로 SR에서 다양한 프로세스 동기화 기법이 어떻게 구현되어 있는지 소개한다.

4.1 SR의 Resources와 Operations

SR에서 리소스들은 오브젝트들을 위한 틀이고, 오퍼레이션들은 오브젝트 상에서의 행위에 대한 틀이다. 각 VM 상에서 RTS는 활성화된 리소스들의 예들의 한 테이블(Res table)을 유지한다. 하나의 리소스는 resource capability에 의해서 나타내며, 이것은 VM 인식자, Res table 내에서의 그 엔트리에 대한 포인터, 그리고 그 리소스의 명세 부분에 선언된 각 오퍼레이션에 대한 한 operation capability(Op cap)로 구성되어 있다. VM, RTS, Resource, Operation 그리고 사용자 프로그램 생성 코드(MC)와의 관계를 그림으로 나타내면 그림 4와 같다.

MC가 리소스 구조를 생성하기 위해 "create"문장을 호출하면, 생성될 리소스의 구분자, 생성될 VM, 실제 파라메타들을 포함하는 생성 블록(creation block)을 구축한다. 이 블록은 명시된 VM으로 전달되어 그곳의 RTS로 넘겨진다. 생성 블록

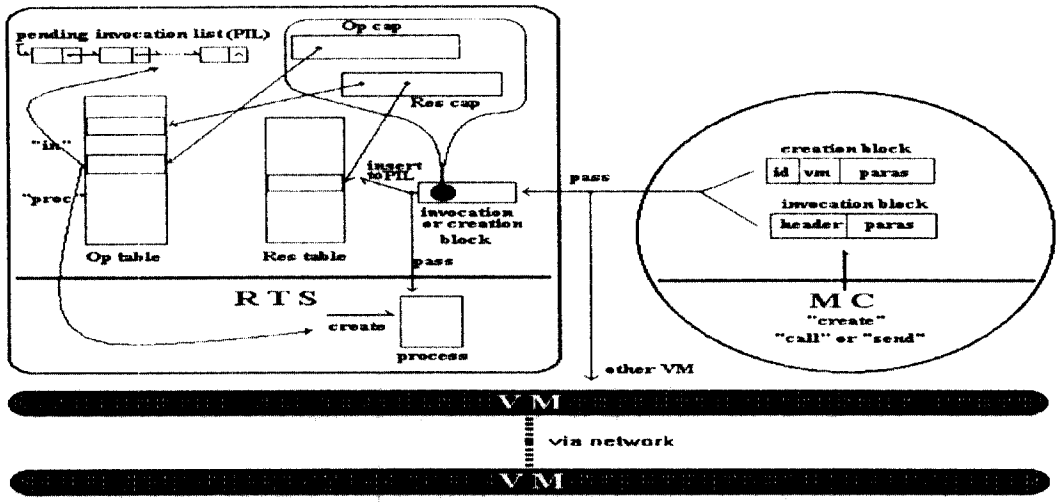

그림 4. Resource와 Operation

이 지명된 VM에 도착했을 때, 그곳의 RTS는 Res table의 한 엔트리를 할당하고 Res cap의 첫 부분을 채운다. 그리고 그때 리소스 초기화를 수행하기 위하여 하나의 프로세스를 생성 시킨다. 모든 리소스에 대하여 MC는 초기화 코드를 가지고 있으며, 이 코드의 주된 역할은 리소스 변수들을 위한 메모리를 할당하고 선언부에 표현된 변수 초기화 작업을 하는 것이다. 그리고 SR 프로그램에서의 리소스 명세와 몸체의 바깥 부분에 표현된 오퍼레이션들을 생성시키는 것이다. 각 오퍼레이션을 생성하기 위해 MC는 RTS와 상호작용한다. 생성될 각 오퍼레이션들을 위해서 RTS는 오퍼레이션 테이블의 한 엔트리을 할당하고 초기화 한다(그림 4 참조). 만약 그 오퍼레이션이 그 리소스의 명세내에 있다면 RTS는 "create"명령으로부터 반환될 resource capability(Res cap)내의 해당 필드의 정보를 채운다. 그 다음으로 만약 사용자가 명시한 초기화 코드가 있다면 초기화를 위한 프로세스를 생성시켜 실행 시킨다.

RTS는 각 VM 상에서 하나의 오퍼레이션 테이블을 유지하고 있다. 이 테이블은 현재 활성화 되어있고 그 VM 상에서 서비스될 각 오퍼레이션들을 위한 엔트리들을 포함하고 있다. 그 엔트리는 해당 오퍼레이션이 "proc"에 의해 서비스 되는지 "in"명령으로 서비스 되는지를 나타낸다. "proc"에 의해서 서비스되는 오퍼레이션의 경우, 그 엔트리는 "proc"을 위한 코드의 주소를 가지고 있다. "in"명령으로 서비스되는 오퍼레이션의 경우, 그 엔트리는 지연된 호출 리스트(pending invocation list:PIL)를 가르키고 있다. 어떤 operation capability(Op cap)는 VM 식별자, 오퍼레이션 테이블 속으로의 인덱스로 구성된다.

어떤 오퍼레이션에 대한 호출은 SR 프로그램에서 "call"이나 "send"로 발생이 되는데, MC는 호출 블록(invocation block)을 구축한다. 이것은 헤드 정보와 실 파라메타 값들로 이루어 진다. MC는 먼저 헤드 내에다 호출의 종류(call, send, concurrent call, concurrent send)를 채운다. 그리고 호출될 오퍼레이션에 대한 capability를 채운다. 그때 MC는 그 호출 블록을 RTS로 전달한다. 만약 필요하다면

그 오퍼레이션이 위치해 있는 다른 VM으로 호출 블록을 전송한다. RTS는 전달된 호출 블록 내의 operation capability를 사용해서 오퍼레이션 테이블 내의 해당 오퍼레이션에 대한 엔트리를 위치시키기 위해 인덱스로 한다. 그러면 그 오퍼레이션이 어떻게 서비스 받아야 할 지 결정하게 된다. 만약 어떤 오퍼레이션이 "proc"에 의해서 서비스 되어야 한다면, RTS는 하나의 프로세스를 생성하고 그 호출 블록을 그 프로세스로 전달한다. 오퍼레이션이 "in" 문장에 의해서 서비스 받아야 한다면, RTS는 그 호출 블록을 그 오퍼레이션을 위한 호출들의 리스트(PIL) 속에 위치시킨다. 이때 어떤 프로세스가 해당 호출을 기다리고 있다면 그 프로세스를 하나 깨운다. 어떤 경우이든 간에 호출하는 프로세스는 블록 되는데, 그 오퍼레이션이 서비스 되어졌을 때 그 프로세스는 깨어나게 되고 호출 블록으로부터 결과를 참조할 수 있게 된다.

이상과 같은 리소스와 오퍼레이션의 관리하에서 어떻게 호출하고 어떻게 서비스 되는가에 따라 SR은 다중의 병행 프리미티브를 지원하고 있다. 즉 호출하는 방법이 "call"인가 "send"인가 그리고 각각에 대하여 서비스 받는 방법이 "proc"인가 "in" 문장 인가에 따라서 아래의 조합이 가능하며 그 조합에 따라 프로세스 동기화 기법에 대한 효과가 결정된다. 부가적으로 세마포는 "send"와 파라메타 없는 "in" 문장을 사용함으로써 모사 될 수 있다.

호출	서비스	효과
call	proc	procedure call (remote procedure call)
call	in	rendezvous
send	proc	dynamic process creation
send	in	asynchronous message passing

4.2 Pthreads에 의한 SR RTS 설계

2장에서 설명한 SR 실행 지원 시스템은 그림 2의 계산 모델을 지원하고 있으며 Pthreads를 이용한 실행 지원 시스템 역시 동일한 계산 모델을 지원하도록 한다. 또한 SR의 구문이나 개념에 변화가 있는 것이 아니므로 리소스나 오퍼레이션 등과 같은 근본적인 개념들의 지원을 그대로 유지하도록 해야 한다.

전형적인 프로세스는 어떤 보호된 주소공간 내에서 화일 명세들이나 신호 벡터들과 같은 어떤 부가적인 속성들을 가진 제어의 단일 흐름으로 구성된다. 이들 프로세스 사이의 통신은 프로세스간 통신(IPC)기법에 제한된다. 통신폭은 운영체제의 능력에따라 불필요하게 제한된다(메시지 큐의 갯수, 공유메모리의 양등의 제한). 또한 프로세스들 사이의 문맥교환은 시스템 트랩을 야기시키므로 그에 따른 부담이 크다. 따라서 실질적인 병행성이 떨어지고 시스템의 자원(프로세스)을 충분히 활용 못하는 단점이 있다.

병행성을 향상할 수 있는 방법으로, 먼저 SR-VM 단위의 구현 모델을 생각할 수 있다. 이 수준은 SR의 VM을 하나의 쓰레드에 대응시키는 기법으로 VM 생성시마다 하나의 쓰레드와 대응 시키는 간단히 해결할 수 있는 방법이다. 실제 기존 SR RTS에서는 Solaris 2.3과 같은 다중 프로세서 환경을 위해서 다중 쓰레드 커널을 효과적으로 사용할 수 있도록 가상머신 단위로 쓰레드를 할당하고 있으나 VM이라는 관점은 병행 단위로서는 너무 큰 개념이기 때문에 실제적으로 효율적인 병행성을 지원하지 못한다. 주소 공간을 공유하면서 여러 쓰레드가 독립적으로 실행되는 Pthreads의 개념과 그 근본 개념이 대치된다. 단지 물리적인 프로세스(예" 유닉스 프로세스) 보다는 작은 관점에서 VM이라는 것을 쓰레드의 개념으로 적용할 수 있다는 의미로 소개된 것 같다.

SR의 프로그래밍 모델에서 병행 행위가 주로 발생하는 단위는 SR의 프로세스 단위이다. 따라서 VM보다는 좀더 세밀한 병행 단위에 쓰레드 개념을 적용하는 것이 병행성을 극대화 시킬 수 있는 방법이라 하겠다. 즉 SR-Process 단위의 구현 모델을 생각할 수 있다. 이는 하나의 SR 프로세스를 하나의 쓰레드에 대응 시키는 기법으로 SR의 프로세스는 쓰레드와 근본적으로 유사하므로 자연스러운 모델이라 할 수 있다. 쓰레드들은 동일 주소 공간을 공유하고 있으므로 보다 쉽고 빠르게 많은 양의 데이터를 공유할 수 있다. 또한 병행 행위들을 보다 자영스럽게 지원할 수 있다. SR의 프로세스 경우도 마찬가지이다.

본 연구에서는 하나의 가상 머신 상에서 생성되는 SR 프로세스들 각각에 하나의 쓰레드를 할당한다. 이러한 쓰레드들은 소속된 가상 머신 상의 전역 자료를 동일한 주소 공간으로 공유하도록 하여 양자간의 근본적인 계산 모델로 일치시켜 효율적으로 병행성을 지원하도록 하였다.

이상의 방법을 기초로 SR의 실행 지원 시스템을 구현한다. 구현 언어는 원래의 실행 지원 시스템 구현 언어인 C 언어를 사용하였다. Pthreads에 의해 SR RTS를 구현할 경우, 리소스와 오퍼레이션을 유지하기 위한 구조는 그림 5와 같다.

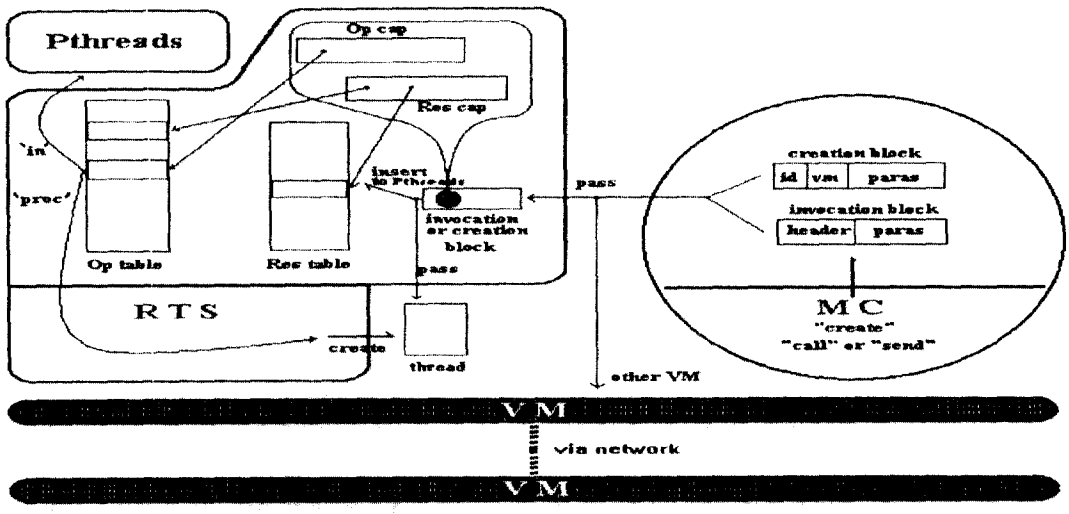

그림 5. 간략화 된 SR RTS 구조

그림 5에서 보듯이 그림 4와 근본적인 차이는 아직 서비스되지 않은 호출에 대한 리스트를 유지하는 pending invocation list라는 자료구조가 빠져있음을 알 수 있다. 이는 기존 SR RTS의 상당부분이 Pthreads 라이브러리로 흡수되어 간략화 되어짐을 보여준다. 이외에도 기존 RTS로부터 흡수되는 부분으로는 프로세스 관리(쓰레드를 관리하는 Pthreads 라이브러리의 자체 기능으로 흡수), 세마포 관리(Pthreads의 mutex와 조건변수로 세마포로 흡수), 메모리관리(Pthreads 자체에서 쓰레드를 위한 메모리 할당)등이 있다. 그리고 원 SR과 마찬가지로 그 쓰레드에 대하여 호출 혹은 생성 블록(invocation or creation block)을 전달 하도록 되어 있다.

4.3 Pthreads에 의한 SR RTS 구현

쓰레드에 기초한 SR 실행 지원 시스템의 원시 코드는 Pthreads의 라이브러리를 사용하여 구현되었으며, SR 원시 코드의 상당부분이 재사용되었다. 또한 SR RTS 코드의 많은 부분이 Pthreads 라이브러리로 흡수되었으며, 어떤 부분은 상이한 차이를 보인다. 본 절을 통하여 그 구현된 접근 기법을 서술한다. 가장 근본적인 아이디어는 SR의 각 프로세스를 하나의 POSIX 쓰레드로 대치시키는 것이다.

컴파일러

실제 구현에 있어 무엇보다도 중요한 것은 SR 프로세스를 Pthreads상의 쓰레드로 가능한한 하나씩 대치시키는 것이다. 그러나 SR의 프로세스와 POSIX의 쓰레드사이에는 인터페이스 구조에서 오는 미세한 차이가 있다. 즉 SR 프로세스 생성시와 쓰레드 생성시에 넘겨지는 파라메타의 갯수가 상이하게 되어 있다. SR 프로세스 구축시에 필요한 파라메타는 4개까지 설정할 수 있도록 되어 있는데 비해 쓰레드 생성에 필요한 파라메타는 하나만 가능하다. 이를 해결하기 위해 SR 컴파일러가 코드를 생성할 때 파라메타 정리 블록(parameter marshalling block:PMB)기법[19]을 적용하여 수정을 가하였다. 하지만 SR의 문법이나 의미에 있어서는 원 개발자의 의도를 그대로 유지시켰다. PMB 기법의 대략적인 모형은 그림 6과 같다.

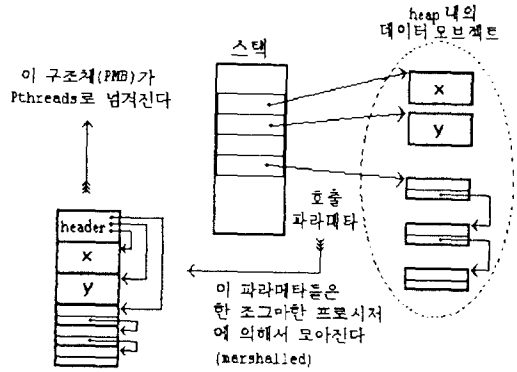

그림 6. PMB 기법에 의한 파라메타 전달 방법

컴파일러의 역할은 PMB 구조로서 파라메타들을 소스코드로 출력만 하며 실제 PMB의 풀이(demarshalling)는 RTS에서 실행시에 행한다.

SR의 프로세스와 POSIX 쓰레드

SR의 프로세스 구조는 일반적인 프로세스의 개념에 능동적인 코루틴의 개념으로 되어있다[1]. 즉 프로세스들은 주종관계를 가진다던가, 파이프라인을 형성한다던가, 코루틴처럼 상호작용 하면서 독립적으로 수행될 수 있다.

프로세스 그룹은 몇몇 다른 프로세스가 병행적으로 실행하는 오퍼레이션들의 동일한 집합을 공유할 수 있는 방법을 제공

해준다. 하나의 리소스 내에 있는 프로세스들은 리소스 변수들을 공유할 수 있기 때문에, 프로세스 그룹은 동일한 오퍼레이션들이 병행으로 공유 자원에 접근할 수 있도록 하기 위한 수단 또한 제공한다. 프로세스 그룹 내의 오퍼레이션은 프로시저처럼 사용될 수 있다. 그러나 DP나 Ada와는 다르게 SR은 프로시저를 가지고 있지 않다. 그 이유는 분산 환경 하에서는 비지역적인 프로시저를 가지는 것이 의미가 없기 때문이다. 따라서 이것은 프로세스 혹은 그와 유사한 구조 형태로 구현되어야 한다. 간단히 말해서 프로세스의 개념은 분산 계산 모델에서 반드시 존재해야 한다.[1]

리소스와 프로세스를 합친 개념은 모니터, 클래스, 모듈의 개념을 일반화시킨다. 리소스가 하나의 프로세스만을 가진다면 그 리소스 변수들은 하나의 프로세스만이 접근할 수 있으므로 모니터와 같이 된다. 만약 리소스가 영구 변수(permanent variable)를 가지고 있고, 하나의 프로세스 그룹만을 가진다면 변수를 공유하는 프로세스들의 객체가 병행하게 실행될 수 있기 때문에 정규 Modula의 모듈과 같다. 만약 리소스가 하나의 프로세스 그룹만을 가지지만 영구 변수가 없다면 여러 개의 프로세스 객체가 있지만 각각은 자신의 지역 변수만 접근할 수 있기 때문에 Concurrent Pascal의 클래스와 같다.

Pthreads인 경우 각 쓰레드는 자신의 속성 구조를 가지고 있어서 원하는 속성을 설정하고 쓰레드를 하나 형성하게된다. 생성된 쓰레드는 즉시 스케줄링되어 수행을 하게 된다. 이때 스케줄링은 Pthreads 내의 스케줄러가 호출되어 스케줄링을 해주게 된다. 원 SR의 경우, 실행 지원 시스템 내에 스케줄러를 둠으로서 해당 실행 지원 시스템의 복잡성과 코드의 부피가 커지고 각 부분에서 프로세스들의 상태를 알고 적절히 스케줄링 해야하는 부담이 있다. 하지만 Pthreads 라이브러리를 사용하여 구현할 경우, 실행 지원 시스템이 일일이 스케줄링 해야하는 부담이 줄어들게 된다. 이러한 특성은 새로운 실행 지원 시스템을 만들 경우에 일일이 스케줄링 방식을 정해야 하고 스케줄링을 해주어야 하는 부담을 줄일 수 있다. 아울러 표준화된 라이브러리를 사용함에 따라(어셈블리 코드도 Pthreads로 흡수되어 없어짐) 프로그램의 이식성이 매우 높아지게 된다.

세마포와 조건 변수

SR에서의 세마포는 상호배제 및 조건 동기화 모두를 이루기 위해서 사용되었다. 조건 동기화를 위해 사용될 경우, P 오퍼레이션이 수행되면 현재 할당받은 세마포에 자신은 블락되고 다른 프로세스가 스케줄링되도록 한다. 이때 블락되는 프로세스는 특정 프로세스로부터 시그널을 받을 때까지 블락이 지속된다. 이 블락을 풀어주는 호출은 V 오퍼레이션이다. V 오퍼레이션이 수행되면 현재 그 세마포에 의해 블락되어 있는 프로세스들을 깨우게 된다. 이처럼 SR은 특정 조건을 만족되기를 기다리는 프로세스들 사이의 동기화 기법에 세마포를 사용하여 구현 하도록 되어있다. 즉 상호배제나 조건 동기화 모두에 세마포를 사용 하도록 되어 있다. 이것은 어떤 세마포에 대하여 상호 배제의 목적으로 사용되었는지 아니면 조건 동기화를 위한 세마포로 사용되었는지를 판별하기 어렵게 만든다. 이러한 구현상의 혼돈을 피하기 위해 근본적으로 상호배제와 조건 동기화는 분리된 프리미티브로 제공하는 것이 바람직하다[16]. 다행히도 Pthreads의 경우는 상호 배제를 위한 기능(mutex를 통해 지원)과 조건 동기화를 위한 기능(조건 변수를 통해 지원)이 엄격히 구분되어서 지원된다. 따라서 쓰레드간 동기화는 Pthreads의 mutex와 조건변수를 도입함으로써 매우 효율적으로 구현 할 수 있다. 이들 구현의 효율성은 이진 세마포등과 같이 사용하기가 까다로운 유형의 다른 동기화 프리미티브(예" 시퀀스, 이벤트 카

운트등)보다 나은 성능을 보일 수 있다. 이들 방식을 적용한 예가 Ada RTS 구현에서 성공적으로 이루어졌다[15]. 사실상 mutex와 조건 변수가 쓰레드간의 동기화 기법에 있어서 적절하고도 효율적인 도구 집합을 이룬다.[18]

5. 결론 및 추후 연구계획

본 연구에서 SR이라는 병행언어의 실행 지원 시스템을 앞으로 국제 표준이 될 Pthreads 라이브러리를 사용하여 구현하였다. 이로 인해 SR 언어의 이식성 및 RTS 구현의 효율성이 더욱 증대되며 설계의 간단화와 정형화가 이루어지게 된다. 구현된 시스템은 SPARC 워크스테이션상에서 개발되었으며 Florida State University에서 구축한 Pthreads 라이브러리[21]를 이용하여 그 동작이 검증되었다. 본 연구를 통하여 얻을 수 있었던 효과를 요약하면 다음과 같다.

첫째, SR 언어와 관련된 기계 의존도가 Pthreads로 흡수됨으로써 Pthreads를 지원하는 모든 시스템에 SR 언어가 용이하게 이식 될 수 있다.

둘째, SR RTS 구현 도구로 Pthreads 라이브러리를 사용함으로써 가지는 강점 중에 하나는 SR 언어를 위한 인터페이스가 불필요하다. 즉 SR언어가 Pthreads 라이브러리로 접근할 경우 SR 컴파일러가 중간 코드로 C-언어로 된 모듈을 생성하기 때문에 인터페이스가 필요하지 않다. 참고로 Ada의 경우는 언어 인터페이스가 필요하다[15].

세째, SR 프로세스 관련된 구조 및 관리 부분이 Pthreads 라이브러리로 흡수됨으로써 SR의 RTS의 복잡도가 감소된다. 이는 앞으로 새로운 병행언어에 대한 RTS 설계시 Pthreads를 도입함으로써 설계가 간단해짐은 물론 정형화(prototyping)가 용이해진다.

네째, 각 컴퓨터 시스템들이 Pthreads의 병렬 실행을 극대화 할 것으로 전망되기 때문에 SR 언어의 실행 효율을 컴퓨터 시스템의 종류에 따라 극대화 하기 위한 노력이 불필요하게 되며 각 기계의 Pthreads의 실행 효율의 향상에 따라 자동적으로 SR 언어의 실행 효율은 향상하게 된다.

다섯째, 본 시스템의 구현을 통해서 병행 프로그래밍 언어를 비롯해서 실시간 프로그래밍 언어의 핵심 기술인 실행 지원 시스템(Runtime Support System)의 구축에 관한 기반 지식을 확보함으로써 차후에 개발할 병행프로그래밍 언어 및 실시간 프로그래밍 언어의 개발이 용이하게 된다.

여섯째, Pthreads의 응용 기법이 확보됨에 따라 병행 언어나 실시간 프로그래밍 언어가 아닌 Pthreads를 직접 이용하여 병행 프로그래밍 및 실시간 프로그래밍을 지원하려고 하는 경우에 신뢰성 있는 프로그래밍이 가능하다.

이상에서 열거한바와 같이 병행 언어인 SR RTS를 Pthreads를 이용하여 구현해 본 연구 결과와 더불어 다음의 잠재적 의미를 지닌다. Pthreads의 적용성 여부에 대한 신뢰성 검증을 Florida State University에서 Pthreads를 이용하여 구현한 Ada RTS 구현에 이어 또 한번 확인이 되었다. 그리고 Pthreads의 응용 기술에 대한 지금까지 축적된 기반지식을 바탕으로 SR이 실시간 기능을 가질 수 있도록 확장 하는 것에 Pthreads를 도입할 수 있을 것으로 기대되며 실제 이를 위한 연구를 계속하고 있다.

참고문헌

[1] Andrews, G.R., Synchronizing resources, ACM Trans. on Prog. Languages and Systems, 3:4, pp.405-430, Oct.

1981.

[2] Andrews, G.R., The distributed programming language SR-mechanisms, de sign and implementation, Software-Practice and Experience, 12:8, pp.719- 754, Aug. 1982.

[3] Andrews, G.R., Concurrent Programming: Principles and Practice, Benjamin/ Cummings Publishing Company, 1991.

[4] Andrews, G.R., and Olsson, R.A., The evolution of the SR language, Di stributed Computing, 1:3, pp.133-149, July 1986.

[5] Andrews, G.R., and Olsson, R.A., The SR programming Language: Con currency in Practice, Benjamin/ Cumm ings Publishing Company, 1992.

[6] Andrews, G.R., and Olsson, R.A., Report on the Programming Language Version 2.3, Dept. of CS at The University of Arizona, Oct 1994.

[7] Andrews, G.R., Olsson, R.A., Coffin, M., Elshoff, I., Nilsen, K., Purdin, T., and Townsend, G., An overview of the SR language and implementation, ACM Trans. on Prog. Lang. and Systems, 10:1, pp.51-86, Jan. 1988.

[8] F. Armand, F. Herrmann, J. Lipkis, and M. Rozier, Multi-threaded Processes in CHORUS/MIX, Proceedings of EEUG Conference, pp.1-13, Spring 1990.

[9] H.E. Bal, "Parallel Programming in SR," Proc. IEEE CS 1992 Int'l Conf. on Computer Languages, Oakland, CA, pp.310-319 (April 1992).

[10] Alan Burns, Geoff Davies, Concurrent Programming, Addison-Wesley, 1993.

[11] Cheung Y.P., Lee M.J., Park Y.S., Repairing Omitted Special Symbols, Journal of the KISS, 21:6, pp.1127-1135, Jun 1994.

[12] Cheung, Y.P., Kim, Y.G. and Lee, M.J., Issues in improving SR runtime support system, to publish in Ann. Bull. of PJC, 18, 1994.

[13] E. Cooper and R. Draves, "C threads", TR CMU-CS-88-154, Carnegie Mellon University, Dept. of CS, 1988.

[14] T. Doeppner Jr., A threads tutorial, TR CS-87-06, Brown University, Dept. of CS, 1987

[15] E.W. Giering III, and T.P. Baker, The Gnu Ada Runtime Library (GNARL): Design and Implementation, Flori da State University, Dept. of CS, ACM Copyright 0-89791-684-0/94/0006 3.50, May 1994.

[16] Narain Gehani, Andrew D. McGettrick, Concurrent Programming: Surveys, Addison-Wesley, 1988

[17] Bill O. Gallmeister and Chris Lanier. "Early experience with POSIX 1003.4 and POSIX 1003.4a", IEEE Symposium on Real-Time Systems, IEEE Computer Society, pp.190-198, 1991.

[18] IEEE, Draft Standard for Information Technology -- Portable Operating System Interface (POSIX) -- Part 1: System Application Program Interface (API) -- Amendment 2: Threads Extension [C Language], IEEE Std P1003.4a/D8, Oct 1993.

[19] Jean Bacon, Concurrent Systems: An Integrated Approach to Operating Systems, Database, and Distributed Systems, Addision-Wesley Publishing co., 1993.

[20] Lee, M.J., and Choe, K.M., Boundedly LR(k)-conflictable grammars, Acta Informatica, 31:2, pp. 261-283, 1994.

[21] Frank Mueller,A library implementation of POSIX threads under UNIX, Proceedings of the USENIX Conference, pp.29-41, Jan. 1993.

[22] Olsson,R.A.Issues in Distributed programming Languages: The Evolution of SR, TR 86-21 (Ph.D. Dissertation), The University of Arizona, Dept. of CS, August 1986.

[23] IEEE Portable Applications Standards Committee, P1003.4: Real-time Extension for Portable Operating Systems(Draft 9), IEEE, 1989.

[24] IEEE Portable Applications Standards Committee,P1003.4a: Threads Extension for Portable Operating Systems(Draft 8), IEEE, Oct 1993.

[25] D. Stein and D. Shah, "Implementing lightweight threads", Proceedings of the USENIX Conference, pp.1-10, Summer 1992.