POSIX 쓰레드를 이용한 SR 실행지원시스템의 설계 및 구현

김 영 곤[†] · 정 영 필[††] · 박 양 수[†††] · 이 명 준[†††]

요 약

본 연구에서는 POSIX 쓰레드(Pthreads)를 사용하여 구현한 새로운 SR 실행지원시스템인 SR/RTS +를 제시한다. SR은 다중의 병행 제어 프리미티브를 이용해서 다양한 병행 프로그래밍 기법을 제공한다. 원 SR 실행지원시스템이 자신의 고유한 쓰레드 경영 기법을 사용하였기 때문에 다른 컴퓨터 시스템에서 동작하기 위해서는 실행지원시스템의 수정이 필요하다. 더우기 실행지원시스템의 실행 효율성은 이런 각각의 상이한 시스템에 연관되어 개별적으로 고려되어야 한다. 따라서 이식성을 증가시키고 실행 효율성을 예측 가능하도록 하기 위해서 SR/RTS +를 하나의 POSIX 프로세스내에서 다중의 실행의 쓰레드를 제공하는 Pthreads(POSIX 쓰레드) 상에서 설계되었다. 또한, SR 컴파일러가 Pthreads 인터페이스를 위한 코드를 적절히 생성하도록 수정되었다. SR/RTS + 는 Florida State University에서 제공하는 Pthreads 라이브러리를 사용하여 Sparc Workstation 상에서 개발되었으며 University of Arizona에서 제공하는 SR 검증 도구로 테스트하였다.

Design and Implementation of SR Run-Time Support System Using POSIX Threads

Young-Gon Kim[†] · Young-Phil Jeong[††] · Yang-Su Park[†††] · Myung-Joon Lee[†††]

ABSTRACT

In this paper, we present a new run-time support system for SR using POSIX threads(Pthreads), called SR/RTS +. SR supports a rich set of concurrent programming techniques with multiple concurrency control primitives. Since the original RTS for SR uses its own thread management facility, it should be modified to work on different computer systems. Furthermore, the run-time efficiency of the RTS should be considered in association with those different systems. Thus, to enhance the portability and to guarantee the predictable run-time efficiency, SR/RTS + is designed on the top of Pthreads(POSIX Threads) which supports multiple threads of control within a single POSIX process. Also, the SR compiler is modified to generate the codes appropriate for Pthreads interface. SR/RTS + has been developed on Sparc Workstation using Pthreads library announced by Florida State University and tested with the SR verification suite distributed by the University of Arizona.

※본 연구는 1995년도 한국과학재단 연구비지원에 의한 결과임 (과제번호 951-0904-030-2)

† 준 회 원:울산대학교 전자계산학과
†† 정 회 원:부산전문대학 전산정보처리과
††† 정 회 원:울산대학교 전자계산학과

논문접수:1996년 9월 6일, 심사완료:1997년 3월 6일

1. 서 론

병행 프로그램은 동시에 수행될 수 있는 행위를 기술한 프로그램이다. 병행 프로그래밍 언어는 병행 프로그래밍을 기술하고 수행시키기 위한 도구이며, 일반적으로 병행 오퍼레이션을 조작하기 위해서 관련된 실행지원시스템을 필요로 한다. 병행 프로그램을 UNIX등과 같은 일반적인 운영체제하에서 수행시키고자 할 때는 다음과 같은 두가지 구현 전략이 있다. 첫째, 컴파일러가 모든 프로세스의 관리와 스케줄링 특성을 가지고 있는 실행지원시스템을 생성하도록 한다. 실행지원시스템은 사용자 프로그램으로부터 생성된 코드와 함께 링크되어 하나의 프로그램을 구성하게 되며 하나의 운영체제 쓰레드로 동작하게 된다. 둘째, 프로세스와 관리와 스케줄링은 주어진 운영체제에 의해 수행되도록 하며 컴파일러는 필요하다면 이러한 시스템에 대한 호출을 생성한다. 사용자 프로그램에서 각 프로세스는 개별적인 운영체제 쓰레드로 대응될 수 있다[18].

첫 번째 방법은 상대적으로 구현하기가 쉬우며 병행 운영체제를 바탕으로 하지 않을 때 선택할 수 있다. 두 번째 방법은 언어의 병행 모형과 운영체제의 병행 모형이 상호 호환적이라면 매우 유용하다. 성능측면을 고려하지 않고 볼 때, 첫 번째 방법의 주요한 단점은 하나의 프로세스가 I/O 호출을 할 경우에 발생한다. 프로그램의 수행 관점에서 볼 때, 프로그램 프로세스들중 어떤 프로세스가 I/O를 시도한 후 블락되고 다른 수행가능한 병행 프로세스가 계속 수행해야 한다. 그러나 운영체제 관점에서 볼때, 프로그램이 I/O를 요청했으며 따라서 프로그램 자체가 블락되게 된다. 실행지원시스템은 그 자체가 스케줄링되는 것이 아니기 때문에 다른 프로세스가 수행될 수 없다[34].

이러한 실행지원시스템의 구현 모형에 있어서 각 프로세스를 쓰레드로 대응시키는 모형이 적합하다고 볼수 있다. 왜냐하면 실행지원시스템의 설계를 위해 필요한 프로그래밍 언어나 커널 차원에서의 프로세스 제어 모델로는 현재 개발되거나 개발중인 운영체제들이 실행단위로서 쓰레드 개념을 도입하고 있기 때문에 쓰레드 개념을 이용하여 구현하는 것이 바람직하다[10, 16]. 경량의 쓰레드(light-weight threads)라고도 불리는 쓰레드는 일반 POSIX 프로세스내에서 전역 데이타를 공유하면서 자신만의 스택과 지역변수, 프로그램 카운터를 가지는 독립적인 실행의 흐름이다. 쓰레드의 문맥은 일반적인 프로세스 문맥보다 상대적으로 작기때문에 경량의 쓰레드로 불린다. 따라서 쓰레드 사이의 문맥교환은 프로세스 사이의 문맥교환보다 더 적은 시간이 걸리게 되며 시스템의 부하도 감소시킬 수 있다.

본 논문에서는 병행언어 SR의 실행지원시스템을 Pthreads를 이용하여 설계 및 구현한 시스템인 SR/RTS +에 관하여 기술한다. SR[8, 9, 11-15, 29]은 단일 병행모형을 지원하는 병행언어와는 달리 프로시져 호출, 랑데뷰, 메시지 전달, 동적 프로세스 생성등과 같은 다중의 병행 모델을 지원하는 언어로서 외국의 대학이나 연구소에서 병행 프로그래밍과 병렬 알고리즘, 데이타 베이스, 분산 시뮬레이션등을 위한 언어로서 사용되고 있으며 화일 시스템이나 명령 분석기등과 같은 분산 운영체제의 한 부분을 구성하는 모듈을 연구하는데 사용되고있다[15]. Pthreads[24, 26, 27]는 하나의 운영체제 서비스에서 제어의 다중 쓰레드를 생성하여 수행되는 것을 지원하는 응용 프로그램 인터페이스이다. 시스템 인터페이스는 컴퓨터 환경을 이식성있도록 제공하기 위한 운영체제 인터페이스의 확장이며, 이 인터페이스의 목적은 쓰레드라고 불리는 제어의 다중 흐름을 제공하는 표준 인터페이스 함수를 제공하는 것이다[28].

구현된 SR/RTS +[1, 2, 3, 4, 5, 6]는 다중 병행성 모델을 지원하기 위한 SR의 기반 구조를 면밀히 분석한 후 SR의 내부 프로세스 모형을 대치할 도구로서, 그리고 실행지원시스템의 이식성을 보다 증가시킬 수 있으며 실행 효율을 증진시킬 수 있는 도구로서 Pthreads를 선택하였다. SR은 다중 병행 모델을 지원하기 위해 하나의 통합된 자료 구조를 이용하며 이 구조내에 호출 프로세스와 수신 프로세스의 정보와 아울러 오퍼레이션의 호출 방법과 수신 방법을 담고 있다. 한 오퍼레이션에 대한 호출 프로세스와 수신 프로세스, 그리고 실행에 관련된 정보는 컴파일시에 정적으로 생성되고 있으며 실행시에 이러한 정보를 이용하고 있다. 또한 SR은 리소스 개념을 이용하여 한 리소스내의 모든 프로세스가 내부 변수를 공유할 수 있도록 하고 있으며 글로벌(global) 리소스를 두어

한 가상 기계상의 모든 쓰레드가 이 글로벌 리소스에 있는 자료를 공유할 수 있도록 되어 있다[5].

이러한 SR의 통신 모형을 전반적으로 살펴볼 때, SR 프로세스 사이의 통신은 공유 메모리를 이용한 통신 모형으로 볼 수 있으며 이는 한 POSIX 프로세스내에서 주소 공간을 공유하며 상호 통신할 수 있는 쓰레드 모형으로 구성 가능함을 알 수 있다. 그리고 실행지원시스템의 구현 모형에 비추어 볼때 SR의 실행지원시스템은 컴파일러가 실행지원시스템을 생성하는 모형으로 볼 수 있으며 이는 Pthreads 라이브러리 구현과 일치하는 점이 있다.

SR/RTS +에서는 SR의 프로세스 통신 모형을 Pthreads로 대응하여 Pthreads의 적용 여부를 확인하여 보았다. 그 결과 SR의 내부구조의 변화를 최대한으로 줄이면서 Pthreads로 적절히 대응되었으며 이는 Pthreads를 이용하여 병행언어의 실행지원시스템을 구성할 수 있음을 보여준다. 병행 언어의 기반구조로서 Pthreads를 활용하는 것은 구현된 병행 언어가 Pthreads가 지원하는 시스템에서 구축된다면 언어 자체의 별다른 노력없이 쉽게 다른 시스템으로 이식 가능함을 보여주며 또한 실행지원시스템의 성능과 효율성도 Pthreads의 성능과 효율성의 증대에 따라 함께 증대될 수 있음을 알 수 있다. 그리고 운영체제의 커널에서 직접적으로 Pthreads를 지원하는 시스템에로의 이식은 코드의 수정없이 바로 커널의 Pthreads 인터페이스를 호출함으로써 구현될 수 있다. 이는 쓰레드 자체의 존재를 커널에 알려주는 두번째 실행지원시스템 구현 모형이 되며 비동기적 I/O를 쓰레드 차원에서 지원하게 되므로 보다 높은 성능 향상을 꾀할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 SR 언어와 Pthreads에 대한 소개를 하며 그 특성들에 대해 살펴본다. 3장에서는 쓰레드 관리기법을 사용하는 시스템의 특성과 구현된 SR/RTS + 시스템에 대해 기술한다. 4장에서는 Pthreads로 구현한 SR 실행지원시스템의 병행성 모형에 대해 기술하며 그 적용성 여부에 대해 설명한다. 끝으로 5장에서는 결론에 대해 언급한다.

2. SR 언어와 Pthreads의 소개

이 장에서는 SR 언어와 Pthreads에 대한 개략적인 소개를 한다. 2.1 절에서는 먼저 SR 언어에 대해 살펴보며 그 특성에 대해 기술한다. 2.2 절에서는 Pthreads에 대한 특징과 기능에 대해 언급한다.

2.1 SR 언어의 소개

SR(Synchronizing Resources) 언어는 Arizona 대학의 Gregory R. Andrews등에 의해 15여년간 개발된 병행 프로그래밍을 기술하기 위한 언어이다. 주요한 언어 구성 요소로는 리소스(resource)와 오퍼레이션(operation)이다. 리소스는 공유 프로세스와 변수들을 은닉시키는 역할을 하게 되며, 하나의 리소스내에서는 임의의 프로세스나 함수를 정의할 수 있으며 또한 프로세스도 원하는 만큼 생성할 수 있다. 이와 아울러 오퍼레이션은 프로세스간의 상호 통신을 위한 기반 기법을 제공한다.

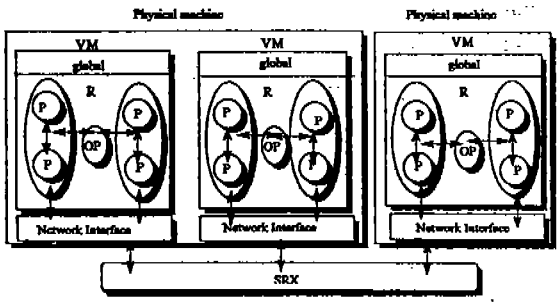
SR은 다중의 오퍼레이션을 발생시키고 유지하기 위한 통합된 기법을 제공한다. 오퍼레이션에 대한 통합된 기법은 프로시져 호출(지역 프로시져 호출 및 원격 프로시져 호출 포함), 랑데뷰, 메시지 전달, 동적 프로세스 생성, 멀티캐스트, 세마포어등과 같은 다중의 병행 프리미티브를 지원하여 보다 다양한 형태의 병행제어를 할 수 있다. 이와 같이 SR은 많은 통신모델을 지원하지만 직교적(orthogonal) 설계를 사용함으로써 분산 및 병렬 프로그래밍 개념의 수를 줄이고 있다. 또한 병행성의 검증을 위하여 CCR, monitor, CSP등을 SR 프로그램으로 바꾸는 세가지 프리프로세서가 지원되고 있으며 그래픽 사용자 인터페이스를 위한 X-Window 인터페이스를 제공하고 있다[9].

• SR 언어의 실행지원시스템

병행 언어는 단순히 그 언어의 문법을 작성하고 번역기인 컴파일러를 작성함으로써 이루어지는 것은 아니다. 이와 아울러 다중의 프로세스를 하나의 프로그램내에서 동시에 실행되도록 지원해 주는 운영체제의 프로세스 스케줄러와 같은 기능이 필요하다. 내부적으로는 병행성 제어를 위해 상호배제나 프로세스 통신 기법이 제공되어야 하며, 분산 환경에서의 효율적인 제어를 위해서는 네트워크 프로그래밍 기법이 지원되어야 한다. 이는 컴파일러 이외에 실행지원시스템(Run-Time Support System:RTS)이 구현되어야 함을 의미한다[5].

SR은 프로그램을 가상기계(Virtual Machine)라 불리는 하나 이상의 주소공간으로 분리할 수 있도록 한다. 각 가상기계는 하나의 물리적인 기계상에서 주소공간을 정의하며 데이타와 프로세스들을 하나의 가상기계상에서 공유하도록 한다. 실행지원시스템은 이러한 가상기계상에서 병행 프로그램이 수행될 수 있는 환경을 제공해 준다. 또한 리소스의 생성과 소멸, 오퍼레이션의 제공 및 호출, 그리고 필요한 메모리의 할당을 위한 도구를 제공한다. 내부적으로 볼때 실행지원시스템은 개별적인 프로세스와 세마포어를 제공하는 집합체이다. 네크워크에 관련된 오퍼레이션은 SR 네트워크 인터페이스(SRX)를 통하여 이루어 진다. 실행지원시스템이 다른 가상기계상의 오퍼레이션을 요청받으면 호출 블락을 해당 가상기계 상으로 넘겨준다. 해당 가상기계에서는 넘겨받은 호출블락에 따라 해당 오퍼레이션을 수행한다. 이러한 요청의 수행 결과는 호출블락을 넘겨받을 때와 유사한 방법으로 되돌려 진다.

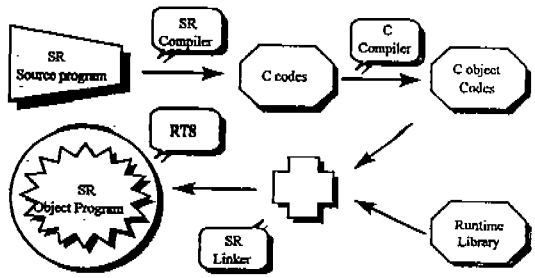
SR 언어의 실행지원시스템 구조는 다음 (그림 1)과 같다.



(그림 1) SR의 계산모델
(Fig. 1) SR computing model

• SR 컴파일러

SR 프로그램이 정상적으로 컴파일되면 SR 컴파일러는 C 코드를 중간 출력으로 내어 놓는다. 이 중간 출력된 C 코드를 시스템의 C 컴파일러로 목적 코드를 생성하며 이를 SR 실행 라이브러리와 결합시켜 최종적인 실행코드를 생성한다. 최종적인 실행코드는 순수한 SR 프로그램의 기능을 수행하기 위한 코드부분(MC)과 프로세스 단위의 제어 및 자체 프로세스 관리를 위한 실행지원시스템부로 나뉘어져 있다. 이러한 관계는 다음 (그림 2)와 같다.



(그림 2) SR 시스템의 구성도
(Fig. 2) SR system architecture

2.2 Pthreads의 소개

쓰레드는 실행제어의 독립적인 순차적 흐름이다 [10, 19, 21, 31]. 쓰레드는 다른 쓰레드와 공통의 주소공간을 공유한다는 점에서 프로세스와 비교될 수 있다. 쓰레드는 단일 프로세서나 다중 프로세서를 위해 공통의 계산구성블락(computational building block)으로 널리 받아들여져 왔다. 단일 프로세서 환경하에서의 쓰레드 모형은 비동기 오퍼레이션에 대한 프로그래밍을 단순하게 하며 다중 프로세서 환경하에서의 쓰레드 모형은 여러개의 프로세서를 이용하여 보다 높은 처리량을 낼 수 있도록 한다.

값싼 병행성을 지원하는 모형은 Mesa 프로그래밍 언어의 코루틴[30], Ada 프로그래밍 언어에서의 다중 태스킹[20, 23]을 포함하여 오랫동안 여러가지 형태로 존재해 왔다. Pthreads는 C-threads[19], Mach threads [32, 33], Brown University threads[21], 그리고 Lynx [24], Sun[31], Chorus[10]와 같은 상용 운영체제의 쓰레드 패키지를 바탕으로 유용한 기능을 제공할 목적으로 제안되었다.

Pthreads는 이러한 시스템을 사용하여 본 경험에 바탕을 두고 도출된 IEEE 표준이며 POSIX.1의 실시간 확장이다. POSIX의 제안은 미국 정부 및 국제 표준(ISO/IEC)으로 채택되어 미래의 실시간 응용 프로그램에 큰 영향을 미칠 것으로 생각되고 있다.

• Pthreads 표준안

Pthreads 표준안은 다중 쓰레드 응용 프로그램을

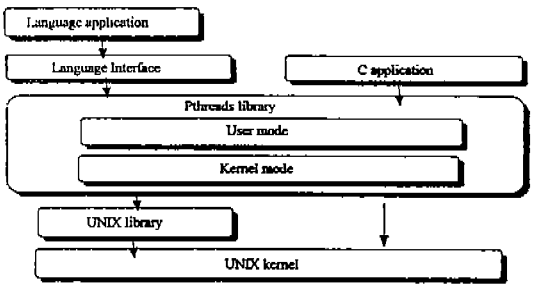
지원하기 위해 제공될 수 있는 다양한 서비스들을 명시한다. 대부분의 인터페이스 명세는 구현상의 세부적인 항목으로 남겨두고 있다. 이러한 Pthreads의 특성은 다음 〈표 1〉과 같다.

〈표 1〉 Pthreads의 특성
〈Table 1〉 Features of pthreads

쓰레드의 관리	쓰레드의 initializing, creating, joining, exiting, destroying
동기화	상호배제, 조건변수
쓰레드당 데이타	쓰레드당 데이타
쓰레드의 우선순위 스케줄링	우선순위 관리, 선점형 우선순위 스케줄링
시그널	시그널 조작기, 비동기적 대기, 시그널의 마스크, 다른 문맥으로의 점프
소멸	cleanup 조작기, 서로다른 인터럽트 상태

• Pthreads의 설계 및 구현

Pthreads 표준은 다중 프로세서상에서의 공유 메모리 응용 프로그램과 실시간 시스템 환경, 단일 프로세서 환경하에서의 다중 쓰레드 프로그램에 대해 단일화된 기본 기능을 제공한다. 현재 Pthreads 구현은 커널 구현과 라이브러리 구현이 있다. 커널 구현인 경우 각 쓰레드는 커널에 그 존재가 알려져 있으며 블락을 야기시키는 I/O 호출을 수행할 때 해당 쓰레드만을 블락시키고 다른 쓰레드는 자신의 수행을 계속해 나갈 수 있다. 하지만 커널 구현인 경우는 시스템의 커널을 재구성해야하는 문제가 있다. 이와는 반대로 라이브러리 구현인 경우 각 쓰레드는 그 존재가 커널에 알려지지 않으므로 I/O 호출을 할 때 전체 프로그램을 블락시키는 결과를 야기시킨다. 본 연구에서 사용한 FSU Pthreads는 라이브러리 구현이며 소프트웨어 계층도는 다음 (그림 3)과 같다.



(그림 3) FSU pthreads 소프트웨어 계층도
(Fig. 3) Software layer of FSU pthreads

인터페이스는 프로그램이 Pthreads 서비스를 이용할 수 있도록 한다. C 언어인 경우 Pthreads의 라이브러리 루틴을 바로 이용할 수 있도록 한다. C 언어가 아닌 언어를 사용할 경우에는 Pthreads가 올바른 인자를 넘겨주도록 해야 하며 타입의 변환, 다른 언어나 컴파일러에 의존적인 정돈을 해야 하므로 해당언어의 인터페이스가 필요하다.

Pthreads에 의해 할당된 구조체는 비동기적 시그널을 조작하는 동안 일관성을 유지하여야 한다. 이러한 일관성의 유지를 위해 라이브러리 구현은 라이브러리 코드의 임계영역이 한순간에 한 쓰레드에 의해 수행되도록 보장하여야 한다. 이러한 구현에서 상호배제를 제공하기 위해 사용되는 기법을 monolithic monitor라고 한다. 여기서는 단순히 라이브러리 커널 또는 커널이라고 부른다[28].

임계영역에서 쓰레드간의 상호배제를 보장하기 위해서는 쓰레드가 커널모드에서 실행되도록 하며 Pthreads 커널은 커널 플랙을 설정하여 커널모드로 진입한다. 이후의 모든 쓰레드의 내부 데이타 구조에 대한 변경은 다른 쓰레드에 대해 상호배제적으로 수행이 되므로 비정상적인 오퍼레이션이 방지되며 쓰레드 구조의 일관성을 보장하게 된다.

또다른 플랙인 디스패처 플랙은 쓰레드가 Pthreads 커널을 떠날때 디스패처가 수행될지 결정하는 플랙이다. 이 플랙은 새로운 쓰레드가 스케줄링될때나 Pthreads 커널에서 수행되고 있는 동안 시그널이 도착했을 때 설정된다. Pthreads 커널을 빠져나갈 때 디스패처 플랙이 설정되어 있지 않다면 단순히 커널 플랙을 해제한다. 만약 디스패처 플랙이 설정되어 있다면 다른 쓰레드로 문맥교환이 일어날 수 있도록 디스패처가 호출된다.

일반적인 상황에서 디스패처의 호출은 스케줄링 정책에 따라 대기하고 있는 쓰레드들중 수행가능한 다음 쓰레드를 선택한다. 만약 선택된 쓰레드가 현재 수행중인 쓰레드와 다르다면 문맥교환이 발생한다. 시그널에 의해 인터럽트당하지 않은 쓰레드가 선택되었다면 문맥은 새로운 쓰레드의 지역상태(local state)

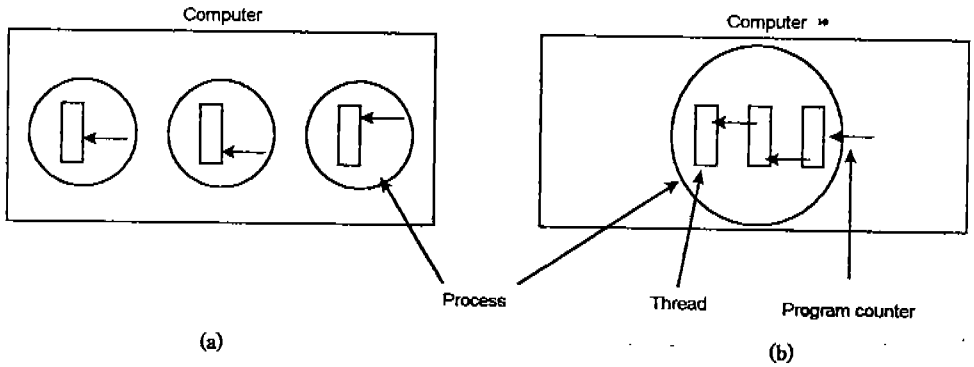
로 바뀐다. 제어가 새로운 쓰레드로 넘어가기 전에 커널과 디스패처 플랙이 초기화되고 커널내에 있는 동안 새로운 시그널이 도착했는지를 검사한다. 새로운 시그널이 없다면 제어가 새로운 쓰레드로 넘어간다. 그렇지 않다면 해당 시그널에 대한 처리를 하고 다음 쓰레드를 선택한다.

3. SR/RTS +

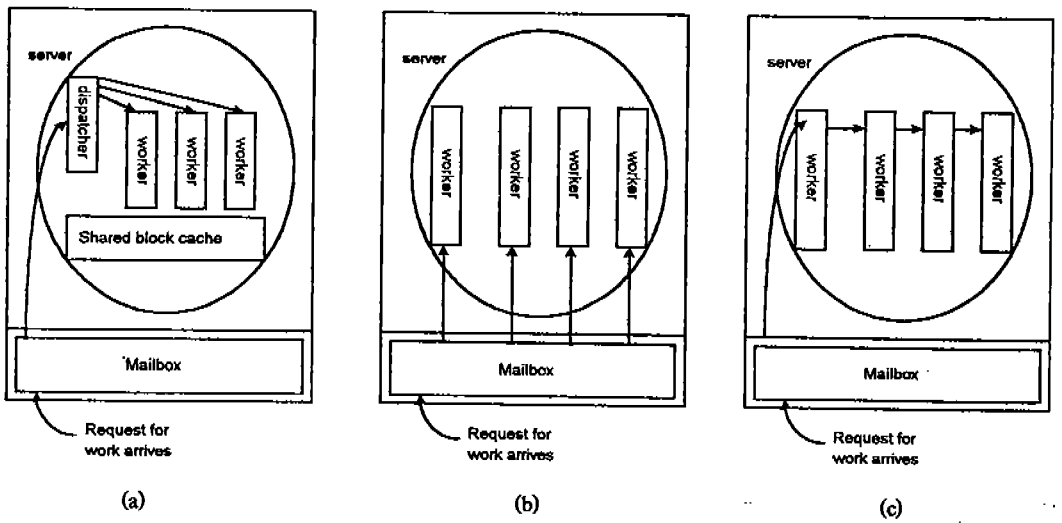
이 장에서는 쓰레드 관리를 이용한 시스템의 특성과 구현한 SR/RTS + 시스템에 대해 기술한다. 먼저 3.1 절에서는 쓰레드 관리를 이용한 시스템의 특성과 잇점에 대해 언급하며 3.2 절에서는 SR/RTS + 시스템의 구조 및 특성에 대해 기술한다.

3.1 쓰레드 패키지를 이용한 시스템

대부분의 전통적인 시스템에서 각 프로세스는 각각의 주소공간을 가지며 단일한 실행의 흐름을 가지고 있다. 그러나 하나의 주소공간을 공유하지만 서로 다른 프로세스같이 거의 병행하게 수행되는 다중 실



(그림 3) 프로세스와 쓰레드
(Fig. 4) Processes and threads



(그림 5) 다중쓰레드 서버 구현 모델
(Fig. 5) Implementation model of a multi-threaded server

행의 흐름이 적합한 경우가 빈번히 발생하고 있다. 이렇게 한 프로세스내에서 제어의 다중 흐름을 쓰레드라고 하며 흔히 경량의 프로세스라고도 한다[34].

(그림 4(a))에는 세개의 프로세스가 있다. 각 프로세스는 각각의 프로그램 카운터, 고유의 스택, 레지스터 집합, 고유의 주소공간을 가지고 있다. 이 프로세스들은 세마포어나 모니터, 메시지등과 같은 시스템의 프로세스간 통신 기법을 통하지 않고는 서로 작업을 공유할 수가 없다. (그림 4(b))에는 세개의 쓰레드를 가진 한개의 프로세스가 있다. 각 쓰레드는 자신의 프로그램 카운터와 스택을 가지고 수행이 된다. 이들 쓰레드사이에는 하나의 주소공간을 가지므로 같은 공유 변수를 공유하게 되고 따라서 쓰레드간의 자료 공유를 위한 별다른 통신 기법이 필요하지 않다. 모든 쓰레드가 모든 가상주소에 접근할 수 있기 때문에 한 쓰레드가 다른 쓰레드의 스택을 읽거나 쓰거나 또는 완전히 제거시킬 수도 있다. 이를 방지하는 것은 거의 불가능하고 또한 필요하지 않기때문에 아무런 제약이 없이 행해질 수 있다. 주소공간을 공유하는 외에도 모든 쓰레드는 열려진 화일에 대한 같은 집합과 자식 프로세스, 타이머, 시그널등을 공유할 수 있다. 따라서 (그림 4(a))는 세개의 프로세스가 본질적으로 아무런 관계가 없을 때 사용되며 (그림 4(b))는 세개의 쓰레드가 같은 작업의 한 부분이며 서로 밀접하게 상호협력하는 모형에 적합하다. 전통적인 프로세스와 마찬가지로 쓰레드는 수행(running), 블락(blocked), 대기(ready), 종료(terminate) 상태를 가질 수 있다.

• 쓰레드의 사용

쓰레드는 순차적인 수행과 블락 시스템 호출을 서로 연관지어 병행성을 강조하기 위해 고안되었다. 다음의 세가지 서버 모형을 살펴보자.

(그림 5(a))에서 분배기(dispatcher)는 시스템의 메일박스로부터 작업을 위한 요구를 읽는다. 그후 요구를 분석하여 유휴(idle) 작업 쓰레드를 선택하여 요구를 넘겨준다. 작업 쓰레드가 깨어나면 공유블락캐쉬를 조사하여 요구가 만족되었는지를 살펴보고 만족될때까지 수면상태(sleep)가 된다. 그리고 스케줄러는 다른 쓰레드를 깨워서 수행을 계속하도록 한다. (그림 5(b))의 팀 모형에서는 모든 쓰레드가 동등하며 각 쓰레드가 자신만의 요구를 선택하게 된다. (그림 5(c))의 파이프라인 모형은 첫 쓰레드가 데이타를 생성하고 이 데이타를 다음 쓰레드로 넘겨준다.

이러한 쓰레드와 관계되어 사용자에게 유용한 특성들을 쓰레드 패키지라고 한다. 이 특성중에서 쓰레드 관리에 관계된 가장 일반적인 방법은 수행시에 동적인 쓰레드의 생성을 하는 것이다. 쓰레드 생성 호출은 일반적으로 쓰레드의 메인 프로그램, 스택, 우선순위등을 가지고 생성된다. 이 모형은 초기에 한 쓰레드로 부터 출발하며 필요시에 하나이상의 쓰레드를 생성하게 된다. 쓰레드는 메모리를 공유하기때문에 여러 쓰레드가 자료를 공유하게 된다. 공유 데이타에 대한 접근은 여러개의 쓰레드가 한 자료에 대해 동시에 접근하지 못하도록 임계영역을 설정하게 된다. 이를 위해 쓰레드 패키지에서는 뮤텍스(mutex)를 두어 상호배제 문제를 해결한다. 이 뮤텍스에 대한 동작은 이진 세마포어와 같다. 또다른 동기화 특성으로는 조건변수(conditional variable)가 있다. 각 조건변수는 일반적으로 생성될때 한 뮤텍스와 연관되어 있다. 뮤텍스는 임계영역에의 진입을 위해 짧은 시간동안의 잠금(lock)을 가지는 것이며 조건변수는 상대적으로 긴 시간동안 리소스가 이용가능할 때까지 기다리는데 사용된다.

• 쓰레드 사용의 장단점

쓰레드 패키지를 사용하는 장점은 다음과 같다. 첫째, 몇몇 응용 프로그램은 근본적으로 병행성을 가지고 있다. 쓰레드는 이러한 응용 프로그램을 위해서 가장 자연스러운 모형을 제시해 준다. 둘째, 다중프로세서 공유 메모리 환경하에서 각 쓰레드는 서로 다른 프로세서상에서 수행될 수 있다. 따라서 쓰레드는 프로그래머가 다중 프로세서환경을 적절히 사용할 수 있도록 하며 하나의 응용 프로그램내에서 진정한 병행성을 이룰 수 있다.

단점은 다음과 같다. 첫째, 쓰레드와 같은 상위단계의 리소스 공유를 병행성과 함께 지원하는 것은 많은 UNIX 프로그래머에게는 새로운 개념이다. 따라서 신뢰성있는 다중 쓰레드 응용 프로그램을 작성하는 것은 많은 노력과 주의가 필요한 작업이다. 둘째, UNIX의 fork() 시스템 호출등과 같은 오퍼레이션들을 효율적으로 다중 쓰레드환경으로 확장하는 것이

어렵다는 것이다.

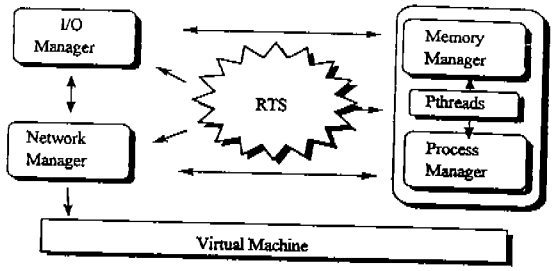
3.2 SR/RTS +

이 장에서는 Pthreads를 이용하여 구현된 SR/RTS + 시스템의 기반 구조 및 특성에 대해 기술을 한다. SR의 실행지원시스템을 Pthreads로 구현하기 위한 방법으로는 기본적으로 다음의 두가지 구현 모형을 생각해 볼 수 있다. 첫째로는 가상기계 수준의 구현 모형이다. 이 모형은 SR의 가상기계를 하나의 쓰레드로 대응시키는 방법으로써 가상기계가 생성될 때마다 하나의 쓰레드를 생성하여 간단히 해결할 수 있는 방법이다. 실제 SR 실행지원시스템에서는 Solaris와 같은 다중 프로세서 환경을 지원하는 운영체제에 대해서 다중 쓰레드 커널을 효과적으로 사용할 수 있도록 가상 기계 단위로 쓰레드를 할당하고 있다. 그러나 가상기계라는 관점이 병행단위로써는 너무 큰 개념이므로 실제적으로 효율적인 병행성을 제공하지 못하고 있다. 그리고 이 구현 모형은 주소공간을 공유하면서 여러 쓰레드가 독립적으로 실행되는 Pthreads의 개념과 그 근본적인 개념이 대치된다.

SR의 프로그래밍 모형에서 병행 행위가 주로 발생하는 단위는 SR의 프로세스 단위이다. 따라서 가상기계보다는 좀더 세밀한 병행 단위에 쓰레드 개념을 적용하는 것이 병행성을 극대화시킬 수 있는 방법이 된다. 이는 SR의 프로세스 하나를 한 쓰레드에 대응시키는 방법으로 SR의 프로세스는 쓰레드와 근본적으로 유사하므로 가상기계단위의 병행성 모형보다는 좀더 자연스러운 모형이라 할 수 있으며 SR/RTS + 는 이 방법을 이용하여 설계되었다[5].

• SR/RTS +의 실행지원시스템

SR/RTS +의 실행지원시스템 구조는 다음 (그림 6)과 같다.



(그림 6) SR/RTS + 시스템 구성도
(Fig. 6) SR/RTS + system architecture

이 구조는 근본적으로 원래의 SR 실행지원시스템과 유사하다. 다만 원래의 SR 실행지원시스템의 메모리 관리 모듈과 프로세스 관리 모듈이 Pthreads 라이브러리 안으로 포함되어 있으며 쓰레드간 동기화도 Pthreads내의 뮤텍스와 조건변수를 통하여 이루어지게 되었다. SR 프로세스는 Pthreads의 한 쓰레드로 대치되었으며 프로세스간의 동기화가 필요한 부분에서는 뮤텍스와 조건변수를 조합하여 생성된 구조인 세마포어를 이용하여 구현되었다. 세마포어의 연산에 대한 개략적인 코드는 다음과 같다.

이러한 세마포어 구조를 이용하여 SR 프로세스 사이의 동기화를 이루었다. P 연산은 세마포어 구조체의 value를 검사하여 값이 0 보다 클때는 단지 그값을 감소시키고 0일 경우에는 조건변수가 만족될때까지 블락되도록 한다. 이때 pthread_cond_wait 호출은 내부적으로 mutex를 해제시키고 조건변수내의 뮤텍스에 의해 블락이 되도록 한다. V 연산은 세마포어의 큐에 대기하고 있는 쓰레드가 없다면 값을 증가시키고 대기하고 있는 쓰레드가 있다면 조건변수에 블락되어 있는 쓰레드를 깨워주게 된다. 이러한 모든 연산은 상호배제를 보장하기 위해 뮤텍스에 의해 잠금(lock)이 되어 있다.

```
struct sem_st {
    int value;
    pthread_mutex_t mutex;
    pthread_cond_t  cond;
}

struct sem_st *sp;
```

```
P Operation:
pthread_mutex_lock(&sp->mutex);

if (sp->value) sp->value--;
else pthread_cond_wait(&sp->cond,
                       &sp->mutex);

pthread_mutex_unlock(&sp->mutex);
```

```
V Operation:
pthread_mutex_lock(&sp->mutex);

if (!sp->cond.queue->head)
    sp->value++;
else  pthread_cond_signal(
                       &sp->cond);

pthread_mutex_unlock(&sp->mutex);
```

실행지원시스템에서 Pthreads로 병행 프리미티브를 효율적으로 지원하기 위한 모형은 4장에서 자세히 언급한다.

• SR 컴파일러의 수정

수정된 SR 컴파일러는 SR 컴파일러가 생성하는 코드 이외에도 Pthreads를 위한 코드가 생성되어야 한다. SR 문법의 변동은 없으므로 필요한 Pthreads 호출을 위한 코드를 생성하도록 코드 생성부분을 새로이 작성하게 된다. 그리고 SR 프로세스와 쓰레드를 생성시키는 호출에서 상이한 인자를 서로 맞추기 위한 인자정리블락(parameter mashalling block)을 두어 인자를 정돈하게 된다. 이 인자정리블락에는 SR 프로세스가 생성될때 인자로써 넘어가는 내부 자료구조뿐만 아니라 프로세스간의 동기화를 이루기 위한 세마포어 구조도 같이 저장한다. 인자정리블락은 쓰레드가 실제 수행될 때 분석되어 각 쓰레드당 데이타(thread-specific data) 그 값을 저장하도록 한다. 쓰레드당 데이타는 모든 데이타를 공유하도록 되어 있는 쓰레드 구조내에서 특정 쓰레드만이 사용하는 값을 자신의 키로 저장하도록 한다. 따라서 쓰레드가 자신의 데이타를 읽어오고자 할때는 해당 키를 사용하여 정보를 얻어내게 된다. 현재 SR/RTS +에서 사용하고 있는 키는 다음 〈표 2〉와 같다.

〈표 2〉 쓰레드당 데이타를 위한 키 정의
〈Table 2〉 Definitions of thread-specific data keys

키의 정의	키의 역할
TYPE	쓰레드의 타입을 저장
SEM	쓰레드의 동기화를 위한 세마포어 저장
PROCQ	자신이 블락되어 있는 쓰레드 큐에 대한 포인터 저장
PROC	프로세스 리스트에 대한 포인터 저장
RINST	리소스에 대한 포인터 저장
COB	CO 루틴을 위한 블락을 저장
INVB	다음 호출 블락을 위한 포인터 저장
BOOL	호출에 대한 선택사항을 저장

이 키 구조를 사용하여 서로 상이한 SR 프로세스 구조체를 Pthreads 구조체로 연관시켜 효율적인 쓰레드 대응이 되도록 하였다.

4. Pthreads로 구현한 SR 병행성 모델 및 성능평가

이 장에서는 Pthreads로 구현된 SR의 병행성 모델과 구현된 RTS에 대한 성능을 평가한다.

4.1 Pthreads로 구현한 SR 병행성 모델

SR에서 주요한 구성요소는 리소스와 오퍼레이션이라고 2.1절에서 언급한 바가 있다. 리소스는 공유 프로세스와 변수들을 은닉시키는 역할을 하게 되며, 하나의 리소스내에서는 임의의 프로세스나 함수를 정의할 수 있으며 또한 프로세스도 원하는 만큼 생성할 수 있다. 이와 아울러 오퍼레이션은 프로세스간의 상호 통신을 위한 기반 기법을 제공한다. 오퍼레이션은 op 선언문으로 선언된다. 이러한 선언문은 리소스 명세부나 리소스, 또는 프로세스내에서 나타날 수 있다. 오퍼레이션은 call이나 send 문에 의해 호출된다. 이것은 또한 proc이나 in 문에 의해 서비스될 수 있다.

이를 이용한 네가지 오퍼레이션의 효과는 다음 〈표 3〉과 같다.

〈표 3〉 SR의 병행 프리미티브
〈Table 3〉 SR concurrency primitives

호출(invocation)	서비스(service)	효 과
call	proc	procedure call(local or remote)
call	in	rendezvous
send	proc	dynamic process creation
send	in	message passing

이러한 호출을 위해 실행지원시스템은 오퍼레이션 테이블을 가상기계에 유지한다. 이 테이블은 가상기계 내에서 서비스되고 현재 수행중인 각 오퍼레이션의 엔트리를 가지고 있다. 이 엔트리는 오퍼레이션이 proc에 의해 서비스될지 in 문장에 의해 서비스될지 결정하게 된다. proc에 의해 서비스되는 오퍼레이션에 대해 엔트리는 proc에 대한 주소를 가진다. in 문장에 의해 서비스되는 오퍼레이션에 대해서는 엔트리가 호출 리스트를 가리키도록 한다.

• 호출 부분

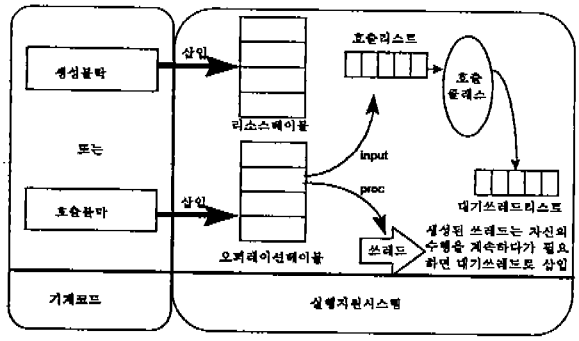
오퍼레이션을 호출하기 위해서 기계코드는 먼저 헤드정보와 실제인자로 구성된 호출블락을 만든다. 기계코드는 호출의 종류와 호출될 특성에 따라 헤드를 채운다. 이후 기계코드는 실행지원시스템으로 호출블락을 넘긴다. 실행지원시스템은 필요에 따라 호출블락을 오퍼레이션이 위치해 있는 가상기계로 보낸다. 이후 실행지원시스템은 오퍼레이션 포인터에 있는 인덱스를 오퍼레이션 테이블에 있는 엔트리로 위치시키고 오퍼레이션이 어떻게 서비스될 지 결정한다. proc에 의해 서비스되는 오퍼레이션에 대해 실행지원시스템은 프로세스를 생성하고 호출블락으로 넘겨준다. in에 의해 서비스되는 오퍼레이션에 대해 실행지원시스템은 호출블락을 오퍼레이션에 대한 호출리스트로 위치시킨다. 그리고 어떠한 프로세스가 호출을 기다리고 있는지 판별한 후 대기하고 있는 프로세스가 있다면 그 프로세스를 깨운다. 두 오퍼레이션 모두 호출을 수행하게 되면 실행지원시스템은 호출하는 프로세스를 블락시킨다. 오퍼레이션이 서비스되었을 때 그 프로세스가 깨어나고 호출블락으로부터 결과를 얻게된다.

• 서비스 부분

서비스 부분을 위한 문장 구조는 매우 복잡하며 구현이 까다롭다. SR의 실행지원시스템에서는 서비스를 지원하기 위한 기본적인 자료구조로서 호출 클래스 구조체를 사용한다. 이 호출 클래스 구조체는 같은 서비스를 제공하는 프로세스 사이의 충돌을 식별하고 제어하기 위해 사용된다. 컴파일러는 서비스 문장의 호출을 바탕으로 해당 오퍼레이션별로 정적인 그룹을 구성한다. 이 그룹을 이용하여 실행지원시스템은 실행시에 실제적인 상호 관계를 파악하여 호출 클래스를 구성하게 되며, 호출 클래스내의 실제 상호 관계는 그룹내에 존재하는 오퍼레이션에 의존적이 된다. 이때 실행지원시스템은 호출 클래스 구조로 호출 클래스 내의 오퍼레이션의 호출 리스트, 호출 클래스에 접근하고자 하는 프로세스의 유무를 지시하는 플랙, 호출 클래스에 접근하고자 하는 프로세스들에 대한 리스트로 유지한다. 각 오퍼레이션 테이블의 엔트리는 자신의 오퍼레이션 구조를 지시한다.

한순간에 한 프로세스만이 주어진 호출 클래스 구조에서 오퍼레이션 호출 블락의 리스트에 접근할 수 있다. 따라서 주어진 호출 클래스에서 한순간에 한 프로세스만이 서비스받기 위해서 호출을 선택하거나 새로운 호출을 추가할 수 있다. 프로세스는 선입선출(first come first out) 순서로 클래스 구조내에 주어진 호출을 접근할 수 있다. 따라서 호출을 기다리는 프로세스는 input 문장내의 동기화 함수나 스케줄링 표현식이 끝나면 그 호출에 접근할 수 있다.

이러한 구조는 다음 (그림 7)과 같이 표현된다. (그림 7)에서 호출리스트는 서비스를 요청하는 쓰레드가 보낸 호출블락을 가지고 있으며 대기쓰레드는 서비스를 제공하는 쓰레드가 원하는 호출블락을 받기위해 대기하는 리스트이다.



(그림 7) 오퍼레이션 처리를 위한 시스템 구조
(Fig. 7) System architecture for handling operations

오퍼레이션의 조합이 send-proc으로 이루어지는 동적 프로세스 생성을 구현하기 위해서는 SR 프로세스 생성 루틴을 Pthreads 생성 루틴으로 대치시킨다. 그리고 호출 쓰레드는 생성한 쓰레드가 수행되기를 기다리지 않고 자신의 수행을 계속하도록 한다. 즉 쓰레드간의 동기화를 위한 세마포어가 필요하지 않다. 생성된 쓰레드는 자신이 제어를 받을 때까지 대기리스트에서 대기하게 된다.

오퍼레이션의 조합이 send-in으로 이루어지는 비동기적 메시지 전달의 구현은 다음과 같다. 먼저 호출 쓰레드는 해당 호출블락을 생성시킨 다음 이 호출블락을 해당 오퍼레이션의 클래스 구조로 보낸다. 이 클래스에는 해당 호출블락을 받고자 하는 쓰레드의 호출블락 리스트에 이 호출블락을 삽입한다. 이 오퍼레이션도 호출 쓰레드가 블락되지 않고 자신의 수행

```
호출블럭을 할당한다.

if (호출자의 형이 == CALL_IN)
    reply = discard = FALSE;
else
    reply = discard = TRUE;

호출블락의 오퍼레이션을 구한다.

if (오퍼레이션의 형이 == INPUT_OP) {
    switch (호출자의 형) {
        case SEND_IN : 오퍼레이션 호출; break;
        case CALL_IN : 세마포어 할당;
                       오퍼레이션 호출;
                       현 쓰레드 블락;
                       break;
    }
}
else {
    switch (호출자의 형) {
        case SEND_IN : 새로운 쓰레드 생성; break;
        case CALL_IN : if (프로시져가 새로운 문맥을 필요로 하는가 == YES) {
                           세마포어 할당;
                           오퍼레이션 호출;
                           현재 쓰레드 블락;
                       } else {
                           프로시져 호출;
                       }
                       break;
    }
```

(그림 8) 오퍼레이션의 호출자 측면에서의 알고리즘
(Fig. 8) Algorithm for sender-side operation

```
if-(클래스가 사용중) {
    현 쓰레드를 블락시킨다.
} else {
    클래스를 사용중으로 바꾼다.
    현 쓰레드의 next_inv를 현 클래스중에서 이용가능한 호출 블락들의
        헤드를 가리키도록 한다.

    if (next_inv != NULL) {
        호출 블락을 얻는다.
    } else {
        if (현 쓰레드를 위한 호출 블락이 있는가) {      /* 있다면 */
            클래스가 사용중일때 삽입되어 있던 호출블락(new_inv)을 모두 이용가능한 호출블락(old_inv)으로 바꾼다.
            if (자신의 차례인가) {                      /* 그렇다면 */
                호출블락을 얻는다.
            } else {
                호출블락을 old_inv로 옮긴다.
                대기하고 있는 쓰레드(old_pr)를 클래스에 접근할 수 있는 쓰레드(new_pr)로 바꾼다.
                new_pr로 호출 블락을 옮긴다.
                해당 쓰레드를 깨운다.
            }
        } else {
            현 쓰레드를 블락시킨다.
            if (새로운 쓰레드가 대기중인가) {           /* 그렇다면 */
                대기중인 쓰레드의 next_inv가 이용가능한 호출블락의 헤드를 가리키도록 포인터를 조정한다.
                대기중인 쓰레드를 깨운다.
            } else {
```

```
                클래스 사용을 False로 한다.
            }
        }
    }
}
```

(그림 9) 오퍼레이션의 수신자 측면에서의 알고리듬
(Fig. 9) Algorithm for receiver-side operation

을 계속하게 된다.

오퍼레이션의 조합이 call-proc인 프로시져 호출을 구현하기 위해서는 동적 프로세스 생성과 마찬가지로 SR 프로세스를 하나의 쓰레드로 대응시킨다. 다만 두 쓰레드간의 동기화를 위해 호출 쓰레드는 서비스하는 쓰레드가 종료되고 결과를 되돌려줄 때까지 블락을 하게 된다. 이때 3.2절에서 언급된 세마포어를 이용하여 동기화를 이루었다.

오퍼레이션의 조합이 call-in인 랑데뷰는 위의 비동기적 메시지 전달과 유사하다. 다만 호출 블락을 받고자 하는 쓰레드가 요청을 받은 다음 요청에 대한 서비스를 종료한 다음 호출 쓰레드가 수행을 계속해 나가게 된다.

이상에서 설명한 SR에서 제공하는 오퍼레이션을 구현하기 위한 알고리즘은 (그림 8)과 (9)와 같다.

4.2 성능평가

본 연구에서 구현된 SR/RTS +는 Sparc Workstation상에서 구현이 되었으며 University of Arizona에서 제공하는 SR 검증도구로 테스트하였다. 특히 성능평가를 위한 예제로서 주어진 자료를 정렬하기 위한 quick sort 프로그램을 이용하였다. 성능평가를 위해 이용한 quick sort 프로그램은 주어진 자료를 정렬하기 위하여 다수의 프로세스를 생성시키며 자료의 정렬이 완전히 완료되었을 때 정렬된 내용을 출력해 준다. 이 프로그램을 이용함으로서 SR 실행지원시스템과의 성능비교뿐만 아니라 프로세스 동기화에 대한 적절한 구현을 시험해 볼 수 있다. 테스트는 예제 자료를 3가지로 나누어 수행되었으며 그 결과는 〈표 4〉와 같다.

테스트된 결과를 바탕으로 SR/RTS +의 성능을 평가해 볼 때 기존의 시스템에 비하여 그 성능이 약간 향상된 것을 볼 수 있다. 이는 기존의 시스템이 시스템의 스케줄링을 위하여 다수의 프로세스를 먼저 생성하여 이 프로세스를 동시에 사용자 프로세스와 스케줄링시킴으로서 성능의 저하를 가져온 것으로 추정되며 Pthreads를 이용하여 쓰레드를 관리하는 SR/RTS +가 자신 고유의 쓰레드 관리기법을 이용하는 종래의 SR/RTS보다 구조적으로 향상되었을 뿐 아니라 성능면에서도 향상되었음을 보여준다.

5. 결 론

본 연구에서 SR이라는 병행언어의 실행 지원 시스템을 국제 표준(ISO/IEC)으로 채택된 POSIX 쓰레드를 사용하여 설계하고 구현하였다. 이로 인해 SR 언어의 이식성 및 RTS 구현의 효율성이 더욱 증대되며 설계의 간단화와 정형화가 이루어지게 된다. 구현된 시스템

〈표 4〉 SR/RTS와 SR/RTS+의 성능비교표
〈Table 4〉 Performance comparison between SR/RTS and SR/RTS+

자료수	SR/RTS			SR/RTS +		
	Real Time	User Time	System Time	Real Time	User Time	System Time
10,000	2.2	1.82	0.32	2.12	1.66	0.36
20,000	4.48	3.78	0.62	4.38	3.64	0.62
40,000	9.14	7.86	1.18	8.72	7.50	1.06

은 SPARC 워크스테이션상에서 개발되었으며 Florida University에서 구축한 Pthreads 라이브러리[28]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[23].

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 병행 프로그래밍 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 가지도록 확장하는 것에 Pthreads를 도입할 수 있을 것으로 기대되며 실제 이를 위한 연구를 수행하였다[6].

그리고 실제로 구현된 SR/RTS +를 평가해본 결과 기존의 시스템보다 다소 우수한 성능을 가지는 것으로 테스트되었다. 나아가 다수의 프로세서를 활용하는 Pthreads 시스템을 기반으로 SR/RTS +가 수행되는 경우에는 그 성능이 더욱 향상될 것이다.

참 고 문 헌

[1] 김영곤, 정영필, 박양수, 이명준, "SR 실행지원 시스템의 새로운 구현에 관한 연구", 제2회 정보과학회 영남지역 학술발표논문집, pp.29-32, 1995년 2월

[2] 김영곤, 정영필, 박양수, 이명준, "SR/RTS +: POSIX 쓰레드를 이용한 SR 실행지원 시스템", 제22회 정보과학회 춘계 학술발표논문집, pp. 321-324, 1995년 4월

[3] 김영곤, 정영필, 박양수, 이명준, "POSIX 쓰레드를 이용한 SR 실행환경 지원에 관한 연구", 울산대학교 공학 연구 논문집, 제26권, 제1호, pp. 327-344, 1995년

[4] 김영곤, 정영필, 박양수, 이명준, "SR의 병행성 제어를 위한 실행지원 시스템의 구조 및 구현에 관한 연구", 제22회 정보과학회 추계 학술발표논문집, pp.915-918, 1995년 10월

[5] 김영곤, "SR/RTS +: POSIX 쓰레드를 이용한 SR 실행지원 시스템", 울산대학교 공학석사학위 논문, 1995.

[6] 정영필, 김영곤, 박양수, 이명준, "실시간 스케줄링을 지원하기 위한 SR의 확장", 제22회 정보과학회 추계 학술발표논문집, pp.845-848, 1995년 10월

[7] 정영필, "ESR : POSIX 쓰레드를 이용한 실시간 동기화를 지원하는 확장된 SR 언어 시스템", 울산대학교 공학박사학위 논문, 1995.

[8] Andrews, G.R., and Olsson, R.A., The SR programming Language : Concurrency in Practice,

Benjamin/Cummings Publishing Company, 1992.

[9] Andrews, G.R., Olsson. R.A., Coffin, M., Elshoff, 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.

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

[11] Andrews, G.R., "Synchronizing resources", ACM Trans. on Prog. Languages and Systems, 3:4, pp. 405-430, Oct. 1981.

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

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

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

[15] 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.

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

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

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

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

[20] U.S. Department of Defense, Military Standard Ada programming Language, ANSI/MIL-STD-1815A, Ada Joing Program Office, Jan 1983.

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

[22] Narain Gehani, Andrew D. McGettrick, Concurrent Programming:Syrveys, Addision-Wesley Publishing co., 1988.

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

[24] 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.

[25] IEEE Portable Applications Standards Committee, P1003.4: Realtime Extension for Portable Operating Systems(Draft 9), IEEE, 1989.

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

[27] 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.

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

[29] 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.

[30] D.D. Redell et. al., "Pilot: An operating system for a person computer", Communications of the ACM, Vol.23, No.2, Feb. 1980.

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

[32] A. Tevanian, R.F. Rashid, D.B. Golub, D.L. Black, E.Cooper, and M.W.Young, MACH threads and the UNIX kernel: The battle for con-

trol, Proceedings of the USENIX Conference, Summer 1987, 185-1997.

[33] Hideyuki Tokuda, Tatsuo Nakajuma, and Prithvi Rao, Real-Time MACH: towards a predictable real-tmie system, USENIX MACH Workshop, Oct. 1990.

[34] Andrews tanenbaum, Distributed Operating Systems, Prentice Hall, 1995.

김 영 곤

1994년 2월 울산대학교 전자계산학과 졸업(학사)
1996년 2월 울산대학교 전자계산학과 졸업(석사)
1996년 3월~현재 울산대학교 전자계산학과 박사과정
관심분야 : 프로그래밍 언어시스템, 병행 프로그래밍, 분산객체 프로그래밍등

정 영 필

1987년 2월 울산대학교 전자계산학과 졸업(학사)
1989년 2월 울산대학교 컴퓨터공학과 졸업(석사)
1996년 2월 울산대학교 컴퓨터공학과 졸업(박사)
관심분야 : 인터넷응용, 병행 및 실시간처리, 데이타베이스응용등

박 양 수

1978년 2월 울산대학교 전자계산학과 졸업(학사)
1981년 2월 서울대학교 계산통계학과 졸업(석사)
1986년 3월~현재 서울대학교 계산통계학과 박사과정
1980년 3월 울산대학교 전자계산학과 근무(현재 부교수)
관심분야 : 분산처리, 컴퓨터알고리즘등

이 명 준

1980년 2월 서울대학교 수학과 졸업(학사)
1982년 2월 한국과학기술원 전산학과 졸업(석사)
1991년 8월 한국과학기술원 전산학과 졸업(박사)
1982년 3월 울산대학교 전자계산학과 근무(현재 교수)
1993년 8월~1994년 7월 미국 버지니아대학 교환교수
관심분야 : 프로그래밍언어, 분산객체 프로그래밍시스템, 병행실시간 컴퓨팅, 인터넷 프로그래밍시스템등