우선순위 역전 문제를 해결하기 위한 통합 실시간 스케줄링 모델

정회원 송 재 신*, 심 재 홍** 최 경 희**, 정 기 현***, 김 흥 남****

An Integrated Real-Time Scheduling Model for Solving Priority Inversion Problem

Jae-Shin Song*, Jae-Hong Shim**, Kyung-Hee Choi**, Gi-Hyun Jung***, Heung-Nam Kim****

Regular Members

요 약

본 논문은 다양한 실시간 스케줄링 알고리즘과 자원 접근 제어 정책을 통합적으로 설계/구현하되 필요에 따라 시스템을 선택적으로 재구성할 수 있게 하는 통합 실시간 스케줄링 모델을 제안한다. 제안 모델은 [3, 4]에서 제안된 기존 모델에 자원 관리자 및 대기 큐 관리자가 추가되었다. 사용자는 태스크 및 자원 속성을 기반 스케줄링 알고리즘에 상관 없이 동일하게 지정할 수 있다. 반면 시스템 설계자는 우선순위 역전 문제를 해결하고 공유 자원에 대한 한정된 블록킹 시간을 보장하기 위한 다양한 자원 접근 제어 정책들을 하위 단계의 복잡한 커널 모듈을 수정하지 않고도 효율적으로 개발할 수 있다. Real-Time Linux [6]에 제안된 스케줄러 모델을 구현한 후, 이를 기반으로 다양한 스케줄링 알고리즘과 자원 접근 제어 정책들을 시험적으로 구현하여 보았다. 여러 성능 실험을 통해 제안 모델을 기반으로 다양한 알고리즘과 정책을 구현한다 해도 실행시의 오버헤드는 크지 않은 반면, 시스템 재구성과 자원 접근 제어 정책을 효과적으로 지원할 수 있다는 것을 확인할 수 있었다.

ABSTRACT

This paper proposes an *integrated real-time scheduling model* that makes system developers design and implement various real-time scheduling algorithms and resource access control protocols integrally, but enables them to reconfigure the system at need. We added a *resource manager* and a *waiting queue manager* to the model already proposed in [3, 4], and defined new *application programming interfaces* and *internal standard interfaces* for the added components. The proposed model enables users to specify direct timing and resource constraints independent of used scheduling algorithm. Also, in order to resolve priority inversion problem and guarantee bounded blocking time on shared resources, system developers can implement various resource access control protocols efficiently without complex low kernel modifications. In Real-Time Linux [6], we implemented the proposed model, representative scheduling algorithms, and access control protocols on purpose to test. In order to confirm efficiency of the model, we measured the performance of several resource access control protocols. The results showed that the model, which even consisted of two hierarchical components and several resource access control modules, didnt have such high run-time overhead, and could efficiently support system reconfiguration and resource access control.

* 아주대학교 정보및컴퓨터공학부,
** 아주대학교 정보통신전문대학원,
*** 아주대학교 전기전자공학부,
**** 한국전자통신연구원

논문번호 : 010013-0220, 접수일자 : 2001년 2월 20일

Ⅰ. 서 론

실시간 시스템에서 가장 중요한 스케줄러는 광범위한 분야의 다양한 응용을 지원하기 위해 반드시 고려되어야 할 커널 구성 요소이다. 실시간 시스템의 적용 대상이 다양해짐에 따라 각 응용 프로그램의 특성에 적합한 다양한 스케줄링 알고리즘들의 필요성도 함께 증가하고 있다[1]. 이는 곧 하나의 알고리즘으로 다양한 실시간 응용들의 요구 조건을 모두 만족시킬 수 없다는 것을 의미하며, 서로 다른 응용은 서로 다른 알고리즘을 필요로 한다. 따라서 실시간 운영체제는 응용 분야별로 널리 알려진 대표적인 알고리즘들을 거의 대부분 지원해야 할 필요가 있다[2]. 이를 바탕으로 시스템 개발자들은 응용 분야별로 적합한 알고리즘을 선택하여 사용할 수 있고, 실시간 시스템 연구자들은 기존 알고리즘을 바탕으로 새로운 알고리즘을 개발할 수가 있기 때문이다. 이를 위해 본 연구팀은 재구성 가능한 스케줄러 모델을 제안한 바 있다[3,4].

실시간 시스템 설계시 고려해야 할 또 다른 중요한 요소는 공유 자원의 효율적인 사용이다. 일반적으로 실시간 시스템은 여러 태스크가 동시에 접근하여 사용할 수 없는 공유 자원을 보호하기 위해 semaphore나 mutex와 같은 태스크간 동기화 메커니즘(synchronization mechanism)을 제공한다. 그러나 이러한 동기화 메커니즘을 사용할 경우, 각 태스크의 응답 시간을 예측할 수 없게 하는 우선순위 역전(priority inversion) 현상이 발생할 수 있다[7]. 우선순위 역전이란 높은 우선순위 태스크가 보다 낮은 우선순위 태스크에 의해 예측할 수 없는 기간 동안 블록킹(blocking) 당하는 것을 의미한다. 이러한 우선순위 역전은 경성(hard) 실시간 태스크들의 시간 제약(time constraints: deadline)을 만족시키지 못하게 할 뿐 아니라, 시스템 동작(behavior)을 예측할 수 없게 하는 결정적인 요인이 된다[8,10]. 따라서 높은 스케줄 가능성(schedulability)을 제공하기 위해서는 우선순위 역전 기간을 최소화하는 것이 중요하며, 경우에 따라서는 이 기간을 한정(bounding)시킬 필요가 있다.

따라서 본 연구에서는 다양한 실시간 스케줄링 알고리즘과 자원 접근 제어 정책을 통합적으로 구현하되 필요에 따라 알고리즘과 정책을 선택적으로 재구성할 수 있는 통합 실시간 스케줄링 모델을 제안하고자 한다. 이는 [3, 4]에서 제안된 스케줄러 모델의 확장된 모델이며, 태스크 스케줄링과 자원 접근 제어를 통합적으로 지원할 수 있는 방안을 제시한다. 이를 바탕으로 스케줄링 알고리즘과 자원 접근 제어 정책을 커널과 복잡하게 연결된 형태로 제공하는 것이 아니라, 하위 단계의 복잡한 커널 모듈을 수정하지 않고도 해당 알고리즘과 정책을 효율적으로 개발할 수 있도록 하고자 한다.

본 논문의 구성은 다음과 같다. 2절에서 자원 접근 제어 정책에 대한 기존 연구를 분석하고, 3절에서는 이들을 지원하는 확장된 통합 스케줄링 모델을 제안한다. 4절에서는 시험적으로 구현된 다양한 자원 접근 제어 정책들의 성능을 측정한 후, 이를 바탕으로 제안 모델의 유용성과 타당성을 검토한다. 마지막 5절에서 향후 연구 계획에 대해 논의한다.

Ⅱ. 자원 접근 제어 정책

자원 접근 제어 정책(*resource access control protocol*)이란 언제 어떤 조건에서 자원 접근을 허용하고, 자원을 요구한 태스크들을 어떻게 스케줄링 할 것인가를 정의하는 규칙들의 집합이다. 자원 접근 제어 정책의 궁극적인 설계 목적은 우선 순위 역전 현상을 제어하는데 있다. 즉, 낮은 우선순위 태스크의 자원 소유로 인해 높은 우선순위 태스크가 대기해야 하는 시간(blocking time)을 한정(bounding)시키는 것이다.

Mok [11]에 의해 제안된 Non-preemptive Critical Section Protocol(NCSP)은 하나의 태스크가 임계 영역에 진입하면 그 태스크를 가장 높은 우선순위에서 실행하게 하는 방법이다. 일단 태스크가 자원을 획득하면, 이를 사용하는 동안에는 선점으로 인한 다른 태스크로의 문맥 교환이 일어나지 않기 때문에 교착 상태를 방지할 수 있다. NCSP는 구현하기가 간단하고 블록킹 시간을 기껏해야 하나의 임계 영역으로 제한할 수 있는 장점이 있지만, 낮은 우선순위 태스크에 의해 이와 상관이 없는 모든 높은 우선순위 태스크들이 블록킹될 수 있다.

Priority Inheritance Protocol(PIP) [7]은 요청한 자원이 이미 사용 중인 경우 그 자원을 요청한 태스크를 블록킹(direct blocking) 시킨다. 이 경우 그 자원을 소유하고 있는 태스크는 블록킹 당한 태스크의 우선순위를 승계(inherit)한다. PIP는 태스크가 블록킹될 때마다 그 태스크의 블록킹 시간은 보다 낮은 우선순위 태스크들의 임계 영역 중 가장 긴 실행 시간으로 제한된다. 그러나 한 태스크가 여러

개의 자원을 여러 개의 태스크들과 서로 경쟁할 경우에는 여러 번 블록킹(transitive blocking)될 수 있다. PIP는 구현 자체는 간단하지만, 교착 상태(dead lock)를 방지할 수 없다.

반면, Basic Priority Ceiling Protocol(BPCP) [7]은 태스크들간의 연계 블록킹(transitive blocking)과 교착 상태를 방지할 수 있다. 또한 BPCP는 태스크를 기껏해야 한번 블록킹 시키고, 블록킹 시간을 낮은 우선순위 태스크들의 임계 영역들 중 가장 긴 실행 시간으로 제한 시킨다. 스케줄러는 사전에 각 자원의 우선순위 ceiling(그 자원을 사용하려는 태스크들의 가장 높은 우선순위)을 알아야 한다. 자원을 요청한 태스크의 우선순위가 시스템 우선순위ceiling(현재 사용 중인 모든 자원들의 가장 높은 우선순위 ceiling)보다 높거나, 또는 요청한 태스크가 시스템 우선순위ceiling과 동일한 우선순위 ceiling을 가지는 또 다른 자원을 이미 사용 중일 때에만, 요청한 자원을 허용한다. 그렇지 않으면 요청 태스크는 블록킹 되고, 그 태스크의 우선순위는 해당 태스크를 블록킹 시킨 태스크에게 승계된다. BPCP는 고정 우선순위 스케줄링 시스템에 가장 적합하다. 동적 우선순위 시스템을 위한 수정된 BPCP 정책이 Chen과 Lin [12]에 의해 제안되었으나, 새로운 작업의 도착 때마다 시스템 내의 모든 자원의 우선순위 ceiling을 수정해야 하는 실행시의 오버헤드가 문제로 지적되었다.

동적 우선순위 스케줄링 시스템을 위한 보다 효율적인 Stack Resource Policy(SRP)라는 정책이 Baker [9]에 의해 제안되었다. SRP는 태스크들에게 그들의 상대 만기를 기준으로 하여 짧은 상대 만기일수록 높은 선점 단계(preemption level)를 할당한다. 따라서 주기적 태스크들은 상대 만기를 기준으로 선점 단계를 할당 받으므로 고정 선점 단계를 가진다. SRP에서 각 자원의 선점 ceiling은 그 자원을 사용하려는 태스크들의 가장 높은 선점 단계이고, 임의의 시점에서 시스템 선점 ceiling은 그 시점에서 사용 중인 모든 자원들의 가장 높은 선점 ceiling이다. SRP는 각 태스크의 새로운 작업이 도착할 때, 작업의 선점 단계가 시스템 선점 ceiling보다 높아질 때까지 블록킹 시킨다. 이 경우 블록킹 당한 태스크의 우선순위는 블록킹 시킨 태스크에게 승계된다. 이후 블록킹에서 해제된 작업이 실행될 때는 자신의 원래 우선순위에서 실행된다. SRP는 교착 상태 방지, 블록킹 시간 한정, 사전 자원 요구 사항 등에서 BPCP와 동일한 효과를 가지만, BPCP보다 구현하기 간단하고 낮은 실행 오버헤드를 가진다. 동적 우선순위 시스템에서의 SRP를 Stack Based *Preemption* Ceiling Protocol이라고도 한다 [10].

SRP는 또한 고정 우선순위 시스템에서도 사용 가능하다. 고정 우선순위 시스템에서의 SRP는 태스크의 상대 만기가 아닌 주기를 기준으로 각 태스크에게 선점 단계(고정 우선순위와 동일)를 부여하는 것만 다를 뿐, 자원 할당 전략은 동적 우선순위 시스템에서와 동일하다 (그러나 우선순위는 승계하지 않는다). 이를 Stack Based *Priority* Ceiling Protocol이라 한다. 이상의 장점으로 인해 태스크가 스스로를 블록킹 시키지 않을 경우, BPCP보다 SRP가 더 많이 사용된다.

그러나 동적 우선순위 시스템의 SRP와 같은 원리를 적용하면 BPCP도 동적 우선순위 시스템에 적용할 수 있다. 즉, BPCP에서 사용하는 각 태스크의 우선순위와 자원의 우선순위 ceiling 대신 SRP에서와 같은 각 태스크의 선점 단계와 자원의 선점 ceiling을 사용한다. 이를 Basic *Preemption* Ceiling Protocol이라 한다. 본 연구에서는 Basic *Priority/Preemption* Ceiling Protocol을 모두 *PCP*라 지칭하고, Stack Based *Priority/Preemption* Ceiling Protocol을 간단히 *SRP*라 통칭한다.

이상의 PCP, SRP와 같은 자원 접근 제어 정책에서 알 수 있듯이 동일한 정책일지라도 기반 스케줄링 알고리즘에 따라 구체적인 자원 할당 알고리즘이 달라진다. 즉, 고정 우선순위 스케줄링 알고리즘에서는 태스크의 우선 순위와 자원의 우선순위 ceiling을 기반으로 자원 접근을 제어하지만, 동적 우선순위 알고리즘에서는 태스크의 선점 단계(preemption level)와 자원의 선점 ceiling을 기반으로 제어한다. 이런 이유로 인해 본 연구에서는 스케줄링 알고리즘과 자원 접근 제어 정책을 통합적으로 고려하고자 한다.

Ⅲ. 통합 실시간 스케줄링 모델 (Integrated Real-Time Scheduling Model)

본 절에서는 본 연구팀에 의해 제안되었던 기존 재구성 가능한 스케줄러 모델을 간단히 살펴보고, 이 모델이 가지는 한계점을 극복하기 위한 새로운 모델의 필요성에 대해 기술한다. 그리고 스케줄링 알고리즘과 자원 접근 제어 정책을 통합적으로 지원할 수 있는 확장된 스케줄링 모델을 제안한다.

본 연구에서는 실시간 시스템에 의해 스케줄링되고 실행되는 단위 일(work)을 하나의 작업(*job*)이라 하고, 하나의 시스템 기능을 제공하기 위해 서로 밀접한 연관을 가진 여러 작업들의 집합을 하나의 태스크(*task*)라 한다. 즉, 스케줄링 이벤트(event)가 발생했을 때 실행되는 태스크의 실체(instance)를 해당 태스크의 작업이라 한다.

3.1 기존 제안 모델의 개요 및 문제점

실시간 시스템에서 모든 실시간 태스크들은 주기, 최악의 경우 실행 시간, 상대 만기, 절대 만기, 도착 시간 등과 같은 태스크 속성(task attributes)에 의해 정의된다. 그러나 응용에 의해 정의된 높은 단계의 태스크 속성들은 스케줄링 메커니즘(낮은 단계의 스케줄러)에 의해 처리될 수 있도록 낮은 단계의 속성들로 변환되어야 한다. 예를 들어, Rate Monotonic 스케줄러의 경우 응용은 단지 높은 단계의 태스크 속성들만 지정한다. 따라서 낮은 단계의 스케줄러와 사용자 응용 사이에 가상적인 중간 단계 스케줄러(스케줄링 알고리즘을 구현한 모듈)가 존재하여, 높은 단계의 태스크 속성들을 낮은 단계의 작업 속성들(작업의 시작 시간, 절대 만기, 우선순위 등)로 변환하여야 한다. 대부분의 실시간 커널에서는 이러한 가상적인 중간 단계 스케줄러와 낮은 단계 스케줄러가 서로 밀접하게 결합되어 하나의 커널 스케줄러를 구성한다. 이들은 스케줄러 데이터 구조와 함수들을 공유하면서 복잡하게 얽히고 섞여 있다. 만약 스케줄러 내부에 잠재해 있는 이들 두개의 추상적 모듈들을 분명하게 분리할 수 있다면, 많은 이점을 얻을 수 있다.

본 연구팀은 [3, 4]에서 두개의 구성 요소로 이루어진 재구성이 가능한 스케줄러 모델을 제안했다. 이 스케줄러 모델은 스케줄링 알고리즘을 구현한 상위 계층의 태스크 스케줄러와 이것에 의해 생성된 작업 속성들을 이용해 작업들을 구체적으로 dispatching하는 하위 계층의 스케줄링 *Framework*으로 구성된다. 스케줄링 Framework은 태스크 스케줄러로부터 작업 속성, 타이머 속성 등을 넘겨 받아 실행 가능한 작업(job)들을 준비 큐에 관리하면서, 이들 중 최우선순위를 가진 작업을 선택하여 CPU를 할당한다. 또한 스케줄러에 의해 예약된 시간들을 관리하면서, 예약 시간이 되면 사전 등록된 함수를 실행시켜 준다. 태스크 스케줄러는 스케줄링 Framework를 기반으로 하나의 특정 스케줄링 알고리즘을 구현하며, 태스크 속성을 작업 속성으로 변환하는 기능을 담당한다. 스케줄링 알고리즘은 작업 속성, 즉 언제, 어떤 작업에게, 얼마 만큼의 프로세서 시간을, 그리고 어떤 우선권을 부여할 것인지를 결정한다. 이 스케줄러 모델은 하위 계층의 스케줄링 Framework을 수정하지 않고도 상위 계층의 새로운 태스크 스케줄러를 설계하고 구현할 수 있다는 이점과, 이미 다양한 태스크 스케줄러들을 확보하고 있을 경우 구축하고자 하는 응용에 가장 적합한 것을 선택함으로써 손쉽게 새로운 시스템을 구축할 수 있다는 이점을 제공한다. 이는 동일한 커널로 다양한 실시간 응용을 지원하기 위해서는 반드시 필요한 스케줄러의 재구성 기능이다.

그러나 이 모델은 기반 스케줄링 Framework과 다양한 태스크 스케줄러들만 제공할 뿐, 기반 태스크 스케줄러와 밀접한 연관이 있는 자원 접근 제어 정책에 대한 고려가 빠져 있다. 이는 곧 시스템 설계자가 해당 응용에 적합한 기존 태스크 스케줄러를 활용하여 시스템을 재구성할지라도, 이 스케줄러와 관련한 자원 접근 제어 정책은 별도로 개발해야 한다는 것을 의미한다.

3.2 태스크와 자원의 통합 스케줄링 모델

본 연구에서는 그림 1과 같은 우선순위 역전 문제를 해결하기 위한 통합 실시간 스케줄링 모델을 제안한다. 제안 스케줄러 모델은 스케줄링 알고리즘과 자원 접근 제어 정책을 구현한 상위 계층의 통합 스케줄러와 이것에 의해 생성된 작업 속성들을 이용해 구체적으로 작업들을 dispatching하는 하위 계층의 스케줄링 *Framework*으로 구성되며, 이들 두 구성 요소들은 계층적 상하 관계를 가진다. 응용은

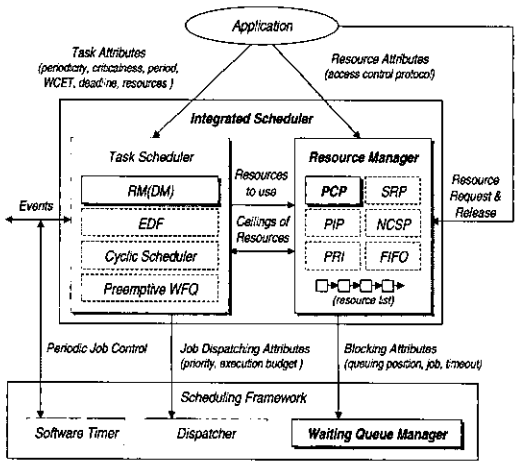


그림 1. 통합 실시간 스케줄링 모델

application programming interface(API)를 통해 태스크 속성과 자원 속성들을 통합 스케줄러에게 전달하며, 통합 스케줄러는 다시 스케줄러 내부 인터페이스를 통해 작업 속성, 타이머 속성, 대기 속성을 스케줄링 Framework에게 전달한다. 태스크 스케줄러와 스케줄링 Framework의 Dispatcher, 소프트웨어 타이머는 [3, 4]에서 이미 제안한 바 있다.

3.2.1 스케줄링 Framework

스케줄링 Framework은 *Dispatcher*, 소프트웨어 타이머, 대기 큐 관리자로 구성된다. Dispatcher는 실행 가능한 작업들을 준비 큐에 관리하면서, 이들 중 최우선순위를 가진 작업을 선택하여 CPU를 할당한다. Dispatcher가 필요로 하는 작업 속성들은 유효 우선순위(*effective priority*), 절대 만기(*absolute deadline*), 실행 예산(*execution budget*) 등이다. 유효 우선순위는 시스템 내의 다른 작업들에 대한 상대적인 실행 우선권을 나타낸다. 실행 예산은 해당 작업에게 부여된 프로세서 할당 시간이다. 실행 예산을 모두 소진했을 때 해당 작업은 강제로 종료된다. 비록 작업이 끝나지 않았고 아직 실행 예산이 남아 있다 할지라도 만기 시간이 되면 이 역시 강제로 종료된다.

소프트웨어 타이머는 스케줄러와 스케줄링 Framework에 의해 예약된 시간들을 관리하면서, 예약 시간이 되면 사전 등록된 함수를 실행시켜 준다. 태스크 스케줄러는 타이머 예약시 예약 시간, *Callback* 함수, 함수 매개 변수 등으로 구성된 타이머 속성을 함께 제출해야 한다. 가장 빠른 예약 시간이 되면 소프트웨어 타이머는 예약 시간들 중 현재 시간보다 같거나 이전인 모든 시간에 대해 사전에 등록된 해당 callback 함수를 매개 변수와 함께 호출한다.

대기 큐 관리자는 작업들을 대기 큐에 삽입/삭제하는 간단한 큐 관리자이다. 요청한 공유 자원을 획득하지 못한 작업들은 그 자원을 사용할 수 있을 때까지 해당 자원의 대기 큐에서 대기해야 한다. 자원 관리자는 자원 접근 제어 정책별로 적절한 대기 정책을 선택해야 한다. 상위 단계의 자원 관리자로부터 넘겨 받은 대기 큐와 대기 속성(정책)을 이용하여 블록킹된 작업을 대기 큐 내의 적절한 위치에 삽입한다. *FIFO* 정책(큐에 들어온 순서대로 관리), 우선순위 정책(작업의 우선순위 순서로), 선점 단계 정책(작업의 선점 단계 순으로) 등을 대기 정책으로는 사용할 수 있다.

3.2.2 통합 스케줄러

통합 스케줄러는 그림 1에 보인 바와 같인 다시 태스크 스케줄러와 자원 관리자로 세분화 된다. 태스크 스케줄러는 태스크 속성을 작업 속성으로 변환하는 기능을 담당하고, 자원 관리자는 공유 자원에 대한 접근을 제어한다.

태스크 스케줄러는 새로운 경성 태스크에 대한 수용 여부를 결정하고, 각 태스크의 새로운 작업 도착을 감시하면서 새로운 작업에 대해 작업 속성을 생성하는 역할을 담당한다. 반면, 자원 관리자는 새로운 작업이 아니라, 현재 수행중인 작업에 대한 자원 요청을 제어하면서, 현 작업을 블록킹 시키거나(대기하게 함) 또는 기존 블록킹된 작업을 깨워주는 역할을 담당한다. 자원 관리자와 태스크 스케줄러의 이러한 행위는 각각 자원 접근 제어 정책과 스케줄링 알고리즘에 의해 결정된다. 스케줄링 알고리즘은 태스크 스케줄러로 하여금 언제, 어떤 작업에게, 얼마 만큼의 프로세서 시간을, 그리고 어떤 우선권을 부여할 것인지를 결정(즉, 작업 속성 결정)하게 한다. 자원 접근 제어 정책은 자원의 할당 여부 및 이로 인해 어떤 작업이 영향을 받는지를 판단하고(작업의 대기, 깨움), 필요한 경우 그런 작업의 속성을 어떻게 변경(즉, 우선 순위 승계)할 것인지를 결정한다.

시스템 설계자는 시스템 구성시 여러 스케줄링 알고리즘과 자원 접근 제어 정책들을 선택적으로 하나 또는 그 이상을 골라 사용할 수 있다. 그러나 스케줄링 알고리즘의 경우 구현된 여러 알고리즘 모듈들을 하나의 시스템에 포함시킬 수는 있지만, 동시에 사용할 수는 없고 적절한 모드 변경을 통해 하나의 스케줄러만을 선택적으로 골라 사용해야 한다.

3.2.3 자원 관리자

응용으로부터 자원 접근 제어 정책과 함께 특정 자원에 대한 요청이 들어오면, 자원 관리자는 주어진 접근 제어 정책에 따라 해당 자원을 할당할 것인지 아닌지를 결정한다. 만약 자원 사용을 허용할 수 없는 경우에는 해당 작업이 대기해야 할 대기 큐와 함께 대기 속성을 대기 큐 관리자에게 넘겨준다. 대기 큐 관리자는 해당 작업을 대기 속성에 따라 큐 내의 적절한 위치에 삽입시킨다. 뿐만 아니라, Dispatcher에 통보하여 해당 작업을 준비 큐에서 삭제하도록 요청한다.

응용은 사용이 끝난 자원에 대해서는 이를 반환

해야 한다. 자원 관리자는 이로 인해 대기하고 있는 작업이 존재하는지를 확인하고, 깨어날 수 있는 모든 작업들을 깨워준다. 이를 위해 먼저 스케줄링 Framework의 대기 큐 관리자에 요청하여 해당 작업들을 대기 큐에서 제거하고, 이를 작업 속성과 함께 Dispatcher에게 넘겨준다.

자원 관리자는 동일 시스템 내에서 다양한 자원 접근 제어 정책들을 동시에 지원할 수 있다. PCP와 SRP의 경우는 기반 스케줄링 알고리즘이 각각 RM과 EDF인 경우에만 사용할 수 있지만, 그 외는 스케줄링 알고리즘과 상관 없이 지원될 수 있다. PCP나 SRP 정책을 사용할 경우, 사용자는 각 태스크가 사용하려는 모든 자원에 대한 사전 정보를 자원 관리자에게 통보해 주어야 한다.

그림 1에서 FIFO, PRI(priority)는 고전적인 자원 접근 제어 정책들이다. FIFO는 요청한 순서대로 해당 자원을 할당하는 정책이고, PRI는 요청 태스크들 중 우선순위가 가장 높은 태스크에게 해당 자원을 할당하는 정책이다. 이들은 비실시간 태스크들을 위해서 주로 사용된다. 그러나 이들 정책들은 시스템 구성시 커널 크기를 줄이기 위해 응용에 의해 사용되는 것들만 추가되고, 나머지 필요 없는 정책들은 제거시킬 수 있다.

3.3 태스크 스케줄러와 자원 관리자의 상관 관계

태스크 스케줄러와 자원 관리자는 자신들의 독립된 고유 업무를 수행하면서도 상호간 밀접한 연관 관계를 가진다. 이유는 자원 접근 제어 정책의 구체적인 자원 할당 알고리즘이 기반 스케줄링 알고리즘에 따라 서로 다른 경우가 있고, 또한 태스크의 블록킹 또는 수행 여부, 자원의 할당 여부 등을 결정하는 것이 어느 한쪽에서의 고유 권한이 아니기 때문이다.

만약 RM을 기반으로 하는 PCP 자원 접근 제어 정책을 사용할 경우, 이 작업이 향후 요청할 공유 자원을 모두 획득할 수 없다면, 자원 관리자는 해당 작업을 자원 획득이 가능할 때까지 대기 큐에서 대기하도록 한다. 이를 위해 자원 관리자는 작업을 준비 큐에서 제거하고 이를 대기 큐에 삽입하며, 또한 블록킹 당한 작업의 우선순위를 블록킹시킨 작업에게 승계 시킨다. 이 과정에서 자원 관리자가 모든 결정 권한을 가지고 있다.

그러나 EDF를 기반으로 하는 SRP인 경우는 그 반대이다. 즉, 새로운 작업 도착시 현 시스템 선점 ceiling 값을 참조하여 해당 작업을 실행 시킬지 또는 블록킹 시킬지를 태스크 스케줄러가 결정하고, 이에 따라 작업의 블록킹, 우선순위 승계 등을 수행한다. 이 경우 자원 관리자는 자원 요청시 무조건 이를 허용하면 된다. 그러나 자원이 반납될 때는 다음으로 높은 자원의 ceiling 값을 태스크 스케줄러로 통보해 주어야 한다. 이 과정에서 태스크 스케줄러가 자원 할당 및 태스크의 실행 여부를 모두 결정한다.

뿐만 아니라, PCP와 SRP는 기반 스케줄링 알고리즘에 따라 적용되는 세부 규칙은 조금씩 달라진다 (표 1 참고). RM 기반 스케줄러상에서는 해당 자원을 사용하려는 태스크들 중 최우선순위 태스크의 우선순위를 그 자원의 우선순위 ceiling으로 채택한다. 이 경우 PCP는 Basic Priority Ceiling Protocol이고, SRP는 Stack Based Priority Ceiling Protocol이다. EDF 기반 스케줄러상에서는 태스크의 상대 만기를 기반으로 각 태스크에게 고정된 선점 단계(preemption level)를 부여하고, 해당 자원을 사용하려는 태스크들 중 선점 단계가 가장 높은 태스크의 선점 단계를 그 자원의 선점 ceiling으로 채택한다. 이 경우 PCP는 Basic Preemption Ceiling Protocol이고, SRP는 Stack Based Preemption Ceiling Protocol이다. 따라서 RM 기반에서는 태스크의 고정 우선순위를 태스크 스케줄링 및 자원 접근 제어용으로 겸용하지만, EDF 기반에서는 태스크 스케줄링용으로 동적 우선순위를 사용하고 자원 접근 제어용으로 태스크의 선점 단계와 자원의 선점 ceiling을 이용한다[10].

표 1. 기반 스케줄러별 PCP, SRP의 특성

	Rate Monotonic	Earliest Deadline First
PCP	Basic Priority Ceiling Protocol	Basic Preemption Ceiling Protocol
SRP	Stack Based Priority Ceiling Protocol	Stack Based Preemption Ceiling Protocol
자원 할당 판단 요소	Effective priority of task, highest priority ceiling of resources	Task preemption level, highest preemption ceiling of resources
Effective priority (preemption level)	Current (inherited or initial) priority of task	Relative deadline of task

자원 관리자와 태스크 스케줄러간의 밀접한 연관 관계로 인해 스케줄링 알고리즘들은 자원 접근 제

어 정책과 별도로 관리될 수 없다. 이를 위해 본 연구에서는 재구성도 가능해야 하지만 자원 접근 제어 정책과 스케줄링 알고리즘들을 서로 묶은 통합 스케줄링 모델을 제안 했다. 따라서 우선순위 역전 문제를 해결하고, 보다 효율적으로 경성 태스크들의 시간 제약을 만족시킬 수 있다.

3.4 통합 스케줄러 알고리즘

우선순위 기반형 태스크 스케줄러는 기본적으로 그림 2의 알고리즘에 따라 실행한다. 전체적으로 우선순위 기반형 스케줄링 알고리즘들은 모두가 이와 같은 방식으로 작동되며, 구체적으로 우선순위 결정 방법에서 차이가 발생한다. 알고리즘에서 **요청**은 상위 단계의 응용으로부터 도착하는 것이고, **이벤트**는 하위 단계의 스케줄링 Framework으로부터 들어오는 것이다. 알고리즘에서 비주기적 태스크 스케줄링은 고려하지 않았다. 자원 접근 제어와 관련한 태스크 우선순위(또는 선점 단계)는 각각 고정 우선순위(또는 동적 우선순위) 스케줄러일 경우 적용된다. 즉, 고정 우선순위 알고리즘(e.g., RM)인 경우 태스크의 주기에 따라 결정된 고정 우선순위가 사용되고, 동적 우선순위 알고리즘(e.g., EDF)인 경우 태스크의 상대 만기에 따라 결정되는 태스크 선점 단계가 사용된다. 이에 따라 자원(또는 시스템)의 우선순위(선점) ceiling이 결정된다.

다음의 요청들이 도착하는 즉시 각 요청에 대해 적절히 대응한다.

새로운 **태스크 생성** 요청:
- 태스크의 수용 테스트를 실시하고, 이를 통과한 태스크에 한해서 해당 태스크를 위한 작업을 생성한다.
- 태스크의 우선순위(선점 단계)를 결정하고, 그 태스크가 사용하려는 각 자원에 대해 자원 관리자에게 자원 예약 요청을 보낸다. (그림 3 참고)
- 스케줄링 Framework에게 첫 작업의 시작 시간으로 활성화 대기 요청을 보낸다.

작업 활성화 이벤트:
- 작업의 실행 예산을 설정하고, 우선순위를 결정 결정한 후, 스케줄링 Framework에게 실행 준비 요청을 보낸다.
- 자원 관리자에게 작업 수행 여부 요청을 보내고, 수행 가능한 작업에 한해서 스케줄링 Framework에게 실행 준비 요청을 보낸다.

실행 완료 요청:
- 태스크의 다음 주기 시작 시간을 작업의 다음 시작 시간으로 설정하여, 스케줄링 Framework에게 활성화 대기 요청을 보낸다.

그림 2. 고정(동적) 우선순위 기반형 태스크 스케줄러 알고리즘

자원 관리자 내의 자원 접근 제어 정책은 기반 스케줄러에 따라 구체적인 자원 할당 방법이 달라진다. 그림 3은 우선순위 기반형 태스크 스케줄러 상의 SRP 알고리즘을 예제로 제시하였다. 여기서 자원이란 논리적 자원을 의미하며, 구체적으로 mutex이다. 알고리즘에서 **요청**은 응용 또는 태스크 스케줄러로부터 도착한다. SRP 정책의 특성상 사용중인 자원들은 하나의 자원 스택에서 관리되며, LIFO 형식으로 push(할당시)되고 pop(반환시)된다. 블록킹 당한 작업들이 대기하는 대기 큐 역시 하나만 사용된다.

자원 생성 요청: 자원을 초기화한다.

자원 예약 요청:
- 자원의 현재 우선순위(선점) ceiling 값이 그 자원을 예약(향후 사용하려는)하려는 태스크의 우선순위(선점 단계)와 비교하여 이보다 작을 경우, 태스크의 우선순위(선점 단계)가 자원의 새로운 우선순위(선점) ceiling 값이 된다.

자원 획득 요청:
- 요청된 자원 사용을 즉시 허용하고, 현재의 시스템 우선순위(선점) ceiling 값을 해당 자원의 ceiling 값으로 수정한다. 할당된 자원을 자원 스택에 push한다.

자원 반환 요청:
- 반환된 자원을 자원 스택으로부터 pop하고, 시스템 우선순위(선점) ceiling 값을 자원 큐의 맨 앞에 있는 자원의 선점 ceiling 값으로 수정한다.
- 자원을 반환한 작업의 우선순위를 자원 획득하기 전의 우선순위로 변경한다. 이를 위해 스케줄링 Framework에게 우선순위 변경 요청을 보낸다.
- 대기 큐에 대기 중인 작업들 중 작업의 우선순위(선점 단계)가 시스템 우선순위(선점) ceiling보다 더 큰 모든 작업들을 대기 큐에서 삭제하고, 스케줄링 Framework에게 실행 준비 요청

을 보낸다.

작업 수행 여부 요청:

- 작업의 우선순위(선점 단계)가 현재의 시스템 우선순위(선점) ceiling 보다 크고 현재 수행 중인 작업의 우선순위(선점 단계) 보다 큰 경우, 작업 수행 가능 으로 통보한다.
- 그렇지 않은 경우, 현재 수행 중인 작업의 우선순위를 블록킹 당하는 작업의 우선순위로 변경한다. (동적 우선순위 스케줄러일 경우에만 우선순위 승계) 블록킹 당한 작업은 대기 큐에 작업의 우선순위(선점 단계) 순으로 보관한다. 태스크 스케줄러에게 작업 수행 불가능 으로 통보한다.

그림 3. 고정(동적) 우선순위 기반형 태스크 스케줄러 상의 SRP 알고리즘

3.5 제안 모델의 이점

본 연구에서 제안하는 통합 실시간 스케줄링 모델은 서로 다른 시간 속성을 가진 태스크들을 지원하기 위한 다양한 스케줄링 알고리즘과 자원 접근 제어 정책을 지원하면서, 새로운 알고리즘과 정책을 개발할 수 있는 스케줄링 Framework을 제공한다. 따라서 제안 모델은 다음과 같은 이점을 가진다.

- **예측성(predictability)**: 다양성과 독립성을 제공하되 시스템 행위에 대한 예측이 가능하도록 한다. 이를 위해 분석이 가능하고 널리 사용되는 대표적인 스케줄링 알고리즘들과 공유 자원에 대한 한정된(bounded) 블록킹 시간을 보장하는 자원 접근 제어 정책들을 스케줄링 Framework을 기반으로 시험적으로 구현하여 제공한다.
- **유연성(flexibility)**: 대부분의 내장형 실시간 응용들은 예측 가능하면서도 유연성을 요구하기 때문에, 실시간 시스템 개발자들은 향후에도 자신들이 직접 해당 응용에 적합한 스케줄러를 개발할 것이다. 따라서 제안 모델의 스케줄링 Framework 인터페이스만 준수하면, 하위 단계의 복잡한 커널 메커니즘을 수정하지 않고도 새로운 알고리즘을 손쉽게 개발할 수 있다. 개발자들은 제안 스케줄러 모델의 기본 틀인 스케줄링 Framework를 활용하여 오로지 상위 단계의 알고리즘 구현에만 전념할 수 있다. 또한 스케줄링 알고리즘별 별도의 자원 접근 제어 정책을 구현할 수 있다.
- **이식성(portability)**: 쉽게 타 시스템에이식할 수 있는 잘 정립된 커널 내부 인터페이스 및 독립된 데이터 구조를 가지게 한다. 이것만 이식될 경우, 이를 기반으로 제공된 다양한 스케줄링 알고리즘과 자원 접근 제어 정책들은 수정 없이 이식될 수 있다.
- **조립성(modularity)**: 구성 요소를 세분화 시키고 표준화된 내부 인터페이스 사용을 준수하도록 한다. 알고리즘 및 정책을 구현하는 구성 요소만 교체함으로써, 손쉽게 시스템 재구성을 이룰 수 있다.
- **재구성(reconfigurability)**: 여러 분야의 다양한 실시간 응용의 요구 조건을 가능한 모두 수용할 수 있도록 특성별 다양한 알고리즘과 정책을 구비하도록 하여, 시스템 개발자로 하여금 응용에 가장 적합한 것을 선택하여 시스템을 재구성할 수 있게 한다. 필요한 경우 개발자가 직접 설계 및 테스트할 수 있는 환경도 함께 제공할 수 있다.

Ⅳ. 성능 측정

본 절에서의 성능 측정 목적은 다양한 커널 함수들의 최악의 경우 실행 시간을 측정하는 것이 아니라, 우선순위 역전을 방지하기 위해 제안된 통합 스케줄링 모델의 유용성과 타당성을 검증하는데 있다. 즉, 새로운 자원 접근 제어자를 태스크 스케줄러나 복잡한 커널 메커니즘 수정 없이 독립적으로 개발될 수 있는가를 확인하고, 또한 자원과 태스크를 통합적으로 스케줄링하는 오버헤드가 얼마 정도인가를 확인하는데 있다.

본 연구에서는 제안된 통합 실시간 스케줄링 모델을 Real-Time Linux(RT-Linux)상에 시험적으로 구현하였다. 기반 RT-Linux는 Linux 2.2.14에 구현된 RT-Linux 2.2를 수정하여 사용했다. RT-Linux는 rtl_time.o, rtl_sched.o, rtl_fifo.o, rtl_posix.o 등의 동적으로 시스템에 삽입 가능한 네 개의 모듈로 구성되어 있으며, rtl_sched.o 모듈은 태스크 관리자, RM 스케줄러, mutex 등을 포함한다. 본 연구팀은 [3, 4]에서 RT-Linux의 기존 rtl_sched.o를 스케줄링 Framework, 태스크 관리자 등을 구현한 rtl_thread.o로 대체하였고, 이를 기반으로 동적으로 삽입/삭제 가능한 RM, EDF 태스크 스케줄러 모듈들을 시험적으로 구현했었다. 본 연구에서는 기존에 구현된 이러한 기반 스케줄러를 본 연구에서 제안된 통합 스케줄링 모델을 수용하도록 수정/확장하였다. 이를 기반으로 다양한 자원 접근 제어 정책들을 시험적으로 구현하였다.

4.1 시스템 사양 및 성능 측정 방법

성능 측정에 사용된 시스템은 Pentium II MMX 233MHz, 32KB L1 Cache(Enabled), 512KB L2 Cache(Enabled), 128MB RAM, 33MHz PCI bus, GASAN TVGA, 3Com 3C905, Seagate st34572W 4GB HDD 등의 하드웨어사양을 가진다. 일반적으로 사용하는 성능 측정 방법은 시스템 타이머를 사용하는 방법과 외부 측정 장치를 이용하는 방법이 있다[5]. 그러나 본 연구에서는 펜티엄 프로세서에서 제공하는 TSC(time stamp counter)를 이용하였다. TSC는 펜티엄 프로세서 계열의 chip 내부에 내장된 전용 64-bit time stamp 계측기이다. 이 계측기의 64-bit 값은 매 프로세서 사이클 당 1씩 증가하며, 사용자는 RDTSC(read time stamp counter) 명령어를 통해 이 값을 읽어 낼 수 있다. 실험에 사용된 프로세서는 233MHz이므로 하나의 CPU 사이클 당 약 4~5 nanosecond 정도의 해상도를 가진다.

본 절에서의 성능 측정 결과는 동일한 실험 과정을 10000번 실행하고, 매번 실행시 소요된 시간을 모두 더해서 얻어진 평균값이다. 실험 중간 중간에 다양한 하드웨어 인터럽트가 발생할 수 있으며 평균값에는 이러한 시간도 모두 포함되었다. 따라서 성능 측정 결과는 각 항목에 대한 최악의 경우 실행 시간은 아니다.

4.2 Mutex 잠금(lock)/해제(unlock) 지연

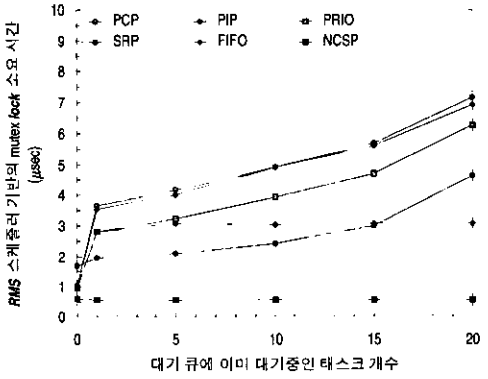
본 연구팀은 [3, 4]에서 제안한 스케줄링 Framework상에 구현된 Cyclic Executive [13], Rate Monotonic [14], Earliest Deadline First [14], Preemptive Weighted Fair-Queuing [15] 등과 같은 대표적인 스케줄러들의 주기적 태스크 활성화/비활성화 지연 실험을 실시했었다. 실험 결과 재구성을 위해 두 단계 계층적 구조를 가진 스케줄러 모델(스케줄링 Framework상에서 구현된 태스크 스케줄러) 일지라도 하부 커널과 복잡하게 연결된 기존 스케줄러와 성능상의 차이가 거의 없다는 것을 확인하였다.

본 절에서는 다양한 자원 접근 제어 정책별 mutex 잠금/해제 지연 시간을 측정하고, 이를 검토해 본다. 실험에서는 FIFO, PRIority, PIP(Priority Inheritance Protocol), NCSP(Non-preemptive Critical Section Protocol), PCP(Priority Ceiling Protocol), SRP(Stack Resource Policy) 등의 성능을 측정하였다. 이중 FIFO, PRI, PIP 등은 기반 스케줄러의 영향을 받지 않는 대신, PCP와 SRP 등은 기반 스케줄러가 RM인지 아니면 EDF인지에 따라 다르게 구현되어야 한다. 일반적으로 RM를 기반으로 하여 구현된 PCP는 Basic Priority Ceiling Protocol을 적용하는 반면, EDF에서는 Basic Preemption Ceiling Protocol을 적용한다. 또한 RM를 기반으로 구현된 SRP는 Stack Based Priority Ceiling Protocol을 적용하며, EDF상에서는 Stack Based Preemption Ceiling Protocol을 적용한다[10]. 따라서 SRP와 PCP를 구현하는데 있어 기반 태스크 스케줄러와 자원 접근 제어 정책은 밀접한 연관 관계를 가지며 상호 보완적이어야 한다. 또한 NCSP의 경우 사실상 기반 스케줄러에서 제어해야 한다. 이러한 측면에서 태스크 스케줄링과 자원 접근 제어는 상호 독립성을 유지하면서도 시스템 전체적으로는 통합적으로 관리되어야 한다. 본 절에서는 서로 다른 기반 스케줄러상에서도 이러한 자원 접근 제어 정책들이 원활하게 작동하고 상대적인 실행 오버헤드가 얼마나 되는가를 확인하고자 한다.

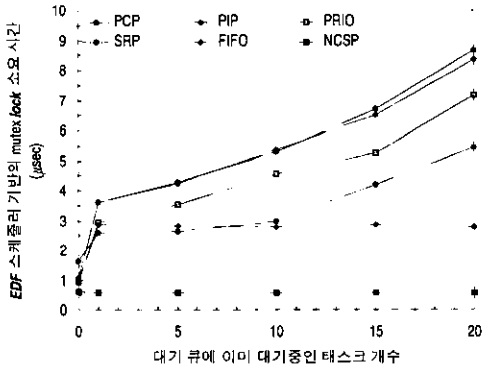
그림 4는 RM 및 EDF 기반 스케줄러상에서의 자원 접근 제어 정책별 mutex lock 지연을 보여 준다. 여기서 mutex lock 지연이란 태스크가 mutex lock을 요청한 후 (pthread_mutex_lock() 호출), 그 즉시 성공적으로 locking했거나(mutex가 unlock 상태일 경우) 또는 자신을 블록킹 시키고 다른 태스크에게 프로세서를 넘기기까지 소요된 총 시간을 의미한다. 이 시간에는 자신을 대기 큐에 삽입하고 다음 실행될 태스크를 스케줄링 하는데 소요된 시간을 포함한다. 그림에서 x축은 mutex lock 요청에 실패하여 대기 큐에 이미 대기 중인 태스크의 개수이며, 이 개수는 현재 lock을 요청한 태스크를 포함한 개수이다. 예외적으로 대기 태스크 개수가 0인 경우는 mutex가 unlock 상태임을 의미한다. y축은 대기 태스크 개수별 mutex lock 지연 시간이다. 실험에서 준비 큐에는 항상 현재 mutex를 소유한 태스크만이 선점된 상태에서 혼자 대기하고 있다.

앞서 밝힌 바와 같이 FIFO, PRI, PIP, NCSP는 기반 태스크 스케줄러의 종류와는 무관하게 독립적으로 구현될 수 있다. 따라서 이들 자원 접근 제어 정책은 RM(그림 4(a))과 EDF(그림 4(b)) 스케줄러상에서 동일한 지연 시간을 보여 주고 있다. PCP 역시 두 기반 스케줄러에서 동일한 지연 시간을 가진다. 이는 동일한 자원 할당 전략을 사용하고 단지 비교 대상이 태스크의 우선순위가 아니라 선점 단계라는 점에서 다른 뿐이기 때문이다. 그러나 SRP의 경우 EDF에서 0.5 ~ 1 msec 정도 더 지연된다.

이는 RM상의 SRP는 태스크 블록킹시 우선순위를 승계하지 않는 반면, EDF상의 SRP는 우선순위를 승계하기 때문인 것으로 분석된다. 그림 4(a)와 그림 4(b) 모두에서 NCSP와 FIFO를 제외한 나머지 정책의 경우 대기 큐의 태스크 개수가 증가할수록 지연 시간이 증가하는 것은 대기 태스크들을 대기 큐에서 우선순위(또는 선점 단계) 순으로 관리하기 때문이다. NCSP는 기반 스케줄러 내의 특정 변수의 값만 변경시키는 것이므로 가장 적은 지연 시간을 보인다.



(a) RM 스케줄러

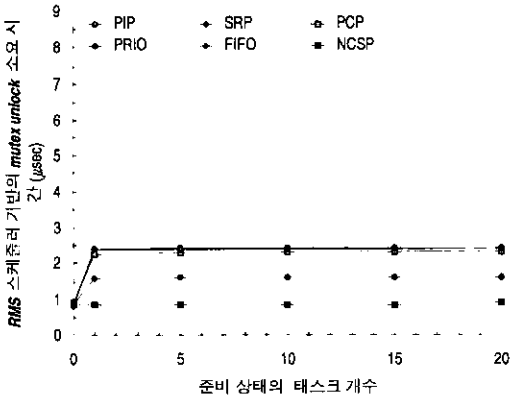


(b) EDF 스케줄러

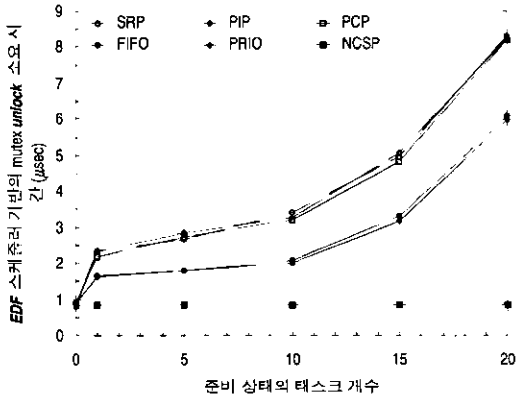
그림 4. Mutex *lock* 지연

그림 5는 RM 및 EDF 기반 스케줄러상에서의 자원 접근 제어 정책별 mutex unlock 지연을 보여준다. 여기서 mutex unlock 지연이란 태스크가 mutex unlock을 요청한 후 (pthread_mutex_unlock() 호출), 그 즉시 복귀하거나(대기 태스크가 없을 경우) 또는 대기 태스크들 중 적절한 태스크를 깨우는 데까지 소요된 총 시간을 의미한다. 이 시간에는 깨어난 태스크를 준비 큐에 삽입하고 다음 실행할 태스크를 스케줄링 하는데 소요된 시간을 포함한다. 그림에서 x축은 준비 큐(대기 큐가 아님)에 대기 중인 태스크의 개수이다.

RM 스케줄러상에서는 전체적으로 준비 큐의 태스크 개수와는 상관없이 각 정책별로 동일한 시간을 보이는 반면, EDF의 경우 점점 증가하는 양상을 볼 수 있다. 이는 다음 수행할 태스크를 선택하기 위해 스케줄링하는 과정에서 소비된 시간 때문이며, EDF의 특성으로 인한 것이다[3,4]. 그러나 NCSP는 기반 스케줄러 내의 특정 변수의 값만 변경시키는 것이므로, 두 스케줄러상에서 모두 가장 적은 지연 시간을 보이며 태스크 개수와는 무관하다는 것을 알 수 있다.



(a) RM 스케줄러



(b) EDF 스케줄러

그림 5. Mutex *unlock* 지연

그림 5(a)와 그림 5(b) 모두에서 PIP, SRP, PCP 등이 가장 높은 비슷한 지연 시간을 보이고, FIFO와 PRIO가 중간 정도의 지연 시간을 보인다. 이는 PIP, SRP, PCP의 경우 unlock을 호출한 태스크의 승계된 우선순위를 lock하기 전의 우선순위로 복구하는데 소비된 시간인 것으로 판단된다.

이상의 mutex 실험을 통해 다양한 자원 접근 정책별로 별도의 메커니즘(구현 모듈)을 제공하고 이를 태스크 스케줄링과 통합적으로 관리해도 이에 따른 자원 관리 및 스케줄링 오버헤드가 크지 않는다는 사실을 확인할 수 있다. 따라서 시스템 설계자는 필요한 정책의 메커니즘만을 선택하여 시스템을 재구성할 수 있는 유연성을 가지며, 불필요한 정책을 시스템에서 제거할 수 있는 이점을 가질 수 있다.

V. 결 론

본 연구에서는 상위 단계의 태스크 스케줄러와 하위 단계의 스케줄링 Framework으로 구성된 기존 제안 스케줄러 모델을 수정하여, 태스크와 자원을 통합적으로 스케줄링할 수 있는 확장된 통합 스케줄링 모델을 제안하였다. 제안 모델은 기존 모델에 자원 관리자 및 대기 큐 관리자를 추가하였으며, 이들간 내부 표준 인터페이스와 응용 API를 새로이 정의하였다. 제안 모델은 대부분의 실시간 커널에서 복잡하게 결합되어 하나의 커널 스케줄러를 구성하던 구성 요소들을 기능별로 재구성이 가능하도록 명확하게 구분하였다. 이 모델은 사용자로 하여금 기반 스케줄링 알고리즘에 상관 없이 주기, 상대 만기, 실행 시간, 사용할 자원 리스트, 자원 관리 정책 등의 동일한 태스크 및 자원 속성을 지정할 수 있게 하고, 공유 자원에 대한 한정된 블록킹 시간을 보장한다. Real-Time Linux상에 제안된 스케줄러 모델을 구현하고 성능 측정을 해 본 결과, 제안 모델을 기반으로 다양한 스케줄링 알고리즘과 자원 접근 제어 정책을 구현한다 해도 실행시의 오버헤드는 크지 않은 반면, 시스템 재구성과 새로운 알고리즘 및 정책 개발을 효과적으로 지원할 수 있다는 것을 확인했다.

본 연구에서 제안한 통합 스케줄링 모델은 단일 프로세서상에서의 스케줄링을 전제로 하고 있다. 제안된 모델을 조그만 확장한다면 SMP(symmetric multiprocessor) 시스템에서도 사용 가능해진다. SMP 시스템은 제안 모델의 기능을 제대로 활용할 수 있는 좋은 기반 시스템이다. 각 CPU마다 서로 다른 스케줄링 알고리즘과 자원 접근 제어 정책을 사용하면서 동일한 스케줄링 Framework을 사용할 수 있기 때문이다. 따라서 본 연구에서는 현재 SMP 시스템을 위한 제안 모델의 확장 작업을 추진하고 있으며, 확장 모델이 완성되면 동일한 스케줄링 Framework을 사용하면서 CPU별 서로 다른 태스크 집합(정적 프로세서 할당 전략)과 태스크 스케줄러를 활용할 수 있게 된다. 이는 곧 각 CPU에 정적으로 할당된 태스크 집합에 가장 적합한 스케줄링 알고리즘과 자원 관리 정책을 선택하여 사용할 수 있는 장점을 제공한다.

참 고 문 헌

[1] 남민우, 실시간 OS 시장 동향, 한국 정보처리학회 논문지, 제5권, 제4호, Jul. 1998.

[2] 한국전자통신연구원, 조립형 실시간 OS 개발 사업, 기술 보고서, Jan. 1999.

[3] 최경희, 스케줄러의 재구성을 지원하기 위한 실시간 커널의 구조, 한국정보처리학회 정보가전용 실시간 OS 컨퍼런스 자료집, pp 45-56, Nov. 2000.

[4] 심재홍, 다양한 실시간 스케줄러을 지원하기 위한 커널 구조화 및 재구성 방안, 공학 박사학위 논문, 아주대학교 대학원 컴퓨터공학과, Feb. 2001.

[5] 이윤석, RTOS 성능 평가 방법 및 현황, 한국정보처리학회 정보가전용 실시간 OS 컨퍼런스 자료집, pp 73-83, Nov. 2000.

[6] Michael Barabanov, A Linux-based Real-Time Operating System, Master thesis, New Mexico Institute of Mining and Technology, June 1997.

[7] L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization, *IEEE Transactions on Computers*, Vol. 39, No. 9, Sep. 1990.

[8] Mike Jones, What Happened on Mars?, Dec. 1997. http://www.cs.cmu.edu/~rtmach/mars.html.

[9] T.P. Baker, A Stack-Based Resource Allocation Policy for Real-Time Processes, *Proc. of 11th IEEE Real-Time Systems Symposium*, Dec. 1990.

[10] Jane W. S. Liu, *Real-Time Systems*, Prentice-Hall, pp. 190-276, 2000.

[11] A. Mok, Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environments, *Ph.D. Dissertation*, MIT, 1983.
[12] M. I. Chen and Kwei-Jay Lin, Dynamic Priority Ceilings: A Concurrency Control for Real-Time Systems, *Real-Time Systems Journal*, Vol. 2, No. 4, pp. 325-346, Dec. 1990.
[13] T. P. Bake and A. Shaw, The Cyclic Executive Model and Ada, *Proc. IEEE Real-Time Systems Symposium*, pp. 120-129, Dec. 1988.
[14] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Journal of the ACM*, Vol. 20, No. 1, pp. 40-61, 1973.
[15] A. Demers, S. Keshav, and S. Shenker, Analysis and Simulation of a Fair Queuing Algorithm, *Journal of Internetworking Research and Experience*, pp. 3-26, Oct. 1990.

송 재 신(Jae-Shin Song) 정회원

1983년 : 충남대학교 전자공학과 졸업(학사)
1990년 : 원광대학교 전자공학과 졸업(석사)
2000년 : 아주대학교 컴퓨터공학과 박사과정(수료)
1984년~1991년 : ETRI 연구원, 중학교, 고등학교 교사
1991년~1997년 : 한국교육개발원 부연구위원
1997년~현재 : 한국교육학술정보원 연구위원
<주관심 분야> Web-Based Instruction, 시스템 프로그램, 교육정보화 등
e-mail : jssong@keris.or.kr

심 재 홍(Jae-Hong Shim) 정회원

1987년 : 서울대학교 전산과학과 졸업(학사)
1989년 : 아주대학교 컴퓨터공학과 졸업 (석사)
1989년~1994년 : 서울시스템(주) 공학연구소
2001년 : 아주대학교 컴퓨터공학과 졸업 (박사)
2001년~현재 : 아주대학교 정보통신전문대학원 Post Doc.
<주관심 분야> 운영 체제, 분산시스템, 실시간시스템 등
e-mail : jhshim@cesys.ajou.ac.kr

최 경 희(Kyung-Hee Choi) 정회원

1976년 : 서울대학교 수학교육과 졸업(학사)
1979년 : 프랑스 그랑데꼴 Enseeiht대학 졸업 (석사)
1982년 : 프랑스 Paul Sabatier 대학 정보공학부 졸업 (박사)
1982년~현재 : 아주대학교 정보 및 컴퓨터공학부 교수
<주관심 분야> 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템 등
e-mail : khchoi@madang.ajou.ac.kr

정 기 현(Gi-Hyun Jung) 정회원

1984년 : 서강대학교 전자공학과 졸업(학사)
1988년 : 미국 Ilinois주립대 EECS 졸업(석사)
1990년 : 미국 Purdue대학 전기 전자공학부 졸업(박사)
1991년~1992년 : 현대반도체 연구소
1993년~현재 : 아주대학교 전기전자공학부 교수
<주관심 분야> 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등
e-mail : khchung@madang.ajou.ac.kr

김 흥 남(Heung-Nam Kim) 정회원
현재 : 한국전자통신연구원 컴퓨터소프트웨어기술연구소 인터넷정보가전연구부 내장형S/W연구팀장