비주기적 태스크 서버들을 지원하기 위한 확장된 실시간 스케줄러 모델

심 재 홍[†]·김 영 일[††]·최 경 희[†††]·정 기 현[††††]·유 해 영[†††††]

요 약

본 논문은 상위 단계의 태스크 스케줄러와 하위 단계의 스케줄링 Framework으로 구성된 기존의 스케줄러 모델[4,5]을 수정하여, 다양한 비주기적 태스크 서버들을 지원할 수 있는 확장된 스케줄러 모델을 제안한다. 제안 모델은 기존 스케줄링 Framework과 태스크 스케줄러를 기반으로 한다. 그러나 비주기적 태스크 스케줄링을 위해 태스크 스케줄러를 다시 주기적 태스크 제어부와 비주기적 태스크 제어부로 분리하였다. 제안 모델은 대부분의 실시간 커널에서 복잡하게 결합되어 하나의 커널 스케줄러를 구성하던 구성 요소들을 기능별로 재구성이 가능하도록 명확하게 구분함으로써, 커널 하부 메커니즘과는 독립적으로 새로운 스케줄링 알고리즘과 비주기적 태스크 서버들을 구현할 수 있게 했다. Real-Time Linux[6]에 제안된 스케줄러 모델을 구현한 후, 이를 기반으로 다양한 스케줄러와 서버들을 시험적으로 구현하여 보았다. 이를 통해 향후 새로운 알고리즘과 서버를 하부의 복잡한 커널 메커니즘 수정 없이 독립적으로 개발할 수 있음을 확인하였다. 또한 여러 성능 실험을 통해 제안 모델을 기반으로 다양한 스케줄러와 서버를 구현한다 해도 실행시의 부하는 크지 않은 반면, 시스템 재구성과 새로운 스케줄러 개발을 효과적으로 지원할 수 있다는 것을 확인할 수 있었다.

An Expanded Real-Time Scheduler Model for Supporting Aperiodic Task Servers

Jae-Hong Shim[†] · Yeong-Ill Kim[††] · Kyung-Hee Choi[†††] · Gi-Hyun Jung[††††] · Hae-Young Yoo[†††††]

ABSTRACT

This paper proposes an *extended scheduler model* that is an extension of the existing model proposed already in [4,5], which consists of upper layer *task scheduler* and lower layer *scheduling framework*. However, in order to support aperiodic task scheduling, the task scheduler has been divided into two parts, such as *periodic task control component* and *aperiodic task control component*. Thus, the proposed model can support various bandwidth-preserving servers that can service aperiodic tasks. The model distinctly separates a classic monolithic kernel scheduler into several kernel components according to their functionality. This enables system developers to implement a new scheduling algorithm or aperiodic task server independent of complex low kernel mechanism, and reconfigure the system at need. In Real-Time Linux [6], we implemented the proposed scheduling framework, representative scheduling algorithms, and several bandwidth-preserving servers on purpose to test. Throughout these implementations, we confirmed that a new algorithm or server could be developed independently without updates of complex low kernel modules. In order to verify efficiency of the proposed model, we measured the performance of several aperiodic task servers. The results showed that the performance of model, which even consisted of two hierarchical components and several server modules, didnt have such high run-time overhead, and could efficiently support system reconfiguration and scheduler development.

Key word : 실시간 시스템(real-time system), 스케줄링 알고리즘(scheduling algorithm), 커널 재구성(kernel reconfiguration), 비주기적 태스크 스케줄링(aperiodic task scheduling)

1. 서 론

실시간 시스템의 적용 대상이 다양해짐에 따라 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].

† 정 회 원 : 아주대학교 정보통신전문대학원, Post Doc
†† 준 회 원 : 아주대학교 대학원 정보컴퓨터공학부
††† 정 회 원 : 아주대학교 정보컴퓨터공학부 교수
†††† 정 회 원 : 아주대학교 전기전자공학부 교수
††††† 정 회 원 : 단국대학교 전산통계학과 교수

논문접수 : 2001년 2월 21일, 심사완료 : 2001년 3월 28일

기존 연구들[4,5]에서 폭 넓은 실시간 응용의 다양한 요구에 적합한 여러 실시간 스케줄링 알고리즘을 구현할 수 있는 재구성이 가능한 스케줄러 모델이 제시되었다. 이를 바탕으로 시스템 설계자들은 해당 응용에 맞는 최적의 알고리즘을 선택하거나 개발하여, 응용의 요구 조건을 최적으로 지원하는 시스템을 구성할 수 있게 되었다. 또한 스케줄링 알고리즘을 커널과 복잡하게 연결된 형태로 구현하는 것이 아니라, 하위 단계의 복잡한 커널 모듈을 수정하지 않고도 해당 알고리즘을 효율적으로 구현할 수 있는 실시간 스케줄링 Framework도 함께 제안되었다. 이를 토대로 실시간 시스템 연구자들은 해당 응용을 효과적으로 지원하는 새로운 알고리즘을 개발하고 테스트할 수 있는 기반 환경을 확보할 수 있게 되었다.

그러나 전통적으로 내장형 제어 시스템과 같은 경성(hard) 실시간 시스템의 경우 주기적 경성 태스크(periodic hard task)들로만 구성되지만, 오늘날 실시간 시스템의 활용 범위가 확대되면서 주기적 경성/연성(soft) 태스크와 비주기적(aperiodic) 경성/연성 태스크 등이 복합적으로 구성된 시스템에 대한 요구가 늘어 나고 있다[2, 22]. 따라서 새로운 시스템을 구축해야 할 경우, 시스템 개발자가 매번 복합 태스크 스케줄링 알고리즘을 직접 구현해야 하는 어려움이 있다. 경우에 따라선 서로 다른 기반 스케줄러로 인해 특정 알고리즘을 구현하기 어렵거나 기존 알고리즘 보다 성능이 더 떨어지는 문제가 발생한다. 예를 들어, Total Bandwidth Server(TBS)[7]의 경우 Earliest Deadline First (EDF)[8]를 기반으로 하기 때문에 Rate Monotonic(RM)[8] 하에서는 구현될 수 없다. 반면 Sporadic Server(SS)[9]는 RM과 EDF 모두에 구현될 수 있지만, EDF상의 SS는 TBS 보다 성능이 더 떨어진다[10]. 따라서 시스템 설계자는 해당 응용에 가장 적합한 기본 스케줄링 알고리즘 외에도 복합 태스크 집합을 지원할 수 있는 알고리즘도 함께 지원해야 한다.

따라서 본 연구에서는 비주기적 태스크들을 서비스하는 다양한 bandwidth preserving servers을 지원할 수 있는 실시간 스케줄러 모델을 제안하고자 한다. 이 모델은 [4,5]에서 제안된 재구성 가능한 스케줄러 모델의 확장된 개념이며, 주기적 태스크 스케줄러에서 다양한 비주기적 태스크 서버들을 지원할 수 있는 방안을 제시한다. 이를 통해 시스템 구축자는 복합 태스크 집합으로 구성된 응용에 가장 적합한 주기적 태스크 스케줄러와 비주기적 태스크 서버를 통합적으로 고려할 수 있고, 필요에 따라 선택적으로 골라 사용(개발)할 수 있는 유연성을 가질 수 있다.

본 논문의 구성은 다음과 같다. 2절에서 기존 제안 스케줄러 모델을 간단히 살펴보고, 비주기적 태스크 스케줄링 알고리즘에 대해 분석한다. 3절에서는 이들을 지원하는 확장된 스케줄러 모델을 제안하고, 구성 요소들이 사용하는 알고리즘에 대해 논의한다. 4절에서는 시험적으로 구현된 비주기적 태스크 서버들의 성능을 측정한 후, 이를 바탕으로 제안 모델의 유용성과 타당성을 검토한다. 마지막 5절에서는 향후 연구 계획에 대해 논의한다.

2. 비주기적 태스크 스케줄링 알고리즘

주기적 경성/연성 태스크와 비주기적 경성/연성 태스크 등이 복합적으로 구성된 태스크 집합에 대한 스케줄링 문제는 이미 오래 전부터 많은 연구가 이루어졌다. RM 알고리즘을 기반으로 하는 비주기적 태스크들의 스케줄링 알고리즘이 Lehoczky에 의해 처음으로 제안되었다[11]. 그는 Deferrable Server(DS)와 Priority Exchange 같은 서버 메커니즘을 사용했으며, 기본 아이디어는 비주기적 태스크의 실행을 가능한 빠르고 효율적으로 서비스하기 위해 특별한 주기적 태스크를 사용하는 것이다. 그러나 이 방법은 기존의 주기적 태스크에 대한 새로운 스케줄 가능성 분석(schedulability analysis)을 필요로 하고, DS와 동일한 주기와 실행 시간을 가진 일반적인 주기적 태스크보다 스케줄 가능 이용률(schedulable utilization)이 더 낮아진다. 또한 DS가 가장 높은 우선순위를 가질 경우에만 스케줄 가능성 분석을 할 수 있다. 이를 개선한 것이 Sprunt에 의해 제안된 Sporadic Server(SS)이다[9]. 이는 SS에 동일한 조건의 주기적 태스크보다 더 많은 실행 시간을 할애하지 않으므로, SS를 일반적인 주기적 태스크와 동일하게 스케줄 가능성 분석을 할 수 있게 했다. 따라서 스케줄 가능 이용률의 손실이 없으며, 어떠한 우선순위에서도 SS를 실행할 수 있고 서로 다른 우선순위를 가진 여러 개의 SS를 생성할 수 있게 했다. 이후 Lehoczky와 Ramos-Thuel[12]은 주기적 태스크들의 만기를 준수하는 범위 내에서 가능한 모든 여유 시간(slack or laxity : off-line으로 계산)을 활용(stealing)하는 기본 아이디어를 이용해 Slack Stealer라는 최적 서비스 방법을 고안했다. 실행 부하가 커서 현실적으로 사용하기에는 문제가 있지만, 비주기적 응답 시간에 대한 최적의 하한치(low bound)를 제시한다는데 의의가 있다. 이들은 또한 [13]에서 비주기적 경성 태스크들을 스케줄링하기 위해 알고리즘을 확장하였다. 이러한 전략은 동적으로 여유 시간을 계산하는 알고리즘이 Davis[14]에 의해 제안되어, 보다 범용적으로 적용 가능하게 되었다.

EDF 알고리즘을 기반으로 하는 비주기적 태스크들의 스케줄링은 Chetto[15, 16]에 의해 처음으로 제안되었다. 이들은 sporadic 태스크(비주기적 경성 태스크)들 또는 선행 관계가 있는 비주기적 연성 태스크들의 그룹에 대한 수용 테스트 알고리즘을 제안하였다. 이는 프로세서 이용률에서는 최적이지만 실제 응용에 적용하기에는 실행 부하가 너무 크다. 이후 앞서 언급한 RM 기반의 DS, SS를 EDF 기반으로 변화한 Deadline Deferrable Server, Deadline Sporadic Server(DSS)

등이 Ghazalie와 Baker에 의해 제안되었으며, DSS를 응용한 보다 단순한 Deadline Exchange Server를 고안하였다[10]. 그러나 보다 단순하면서도 주기적 태스크에 의해 사용되지 않는 여분의 시간을 활용할 수 있는 Total Bandwidth Server(TBS) 알고리즘이 Spuri와 Buttazzo에 의해 제안되었다[7]. 이 서버는 주어진 서버 크기와 동일한 크기를 가진 sporadic 태스크처럼 스케줄링 된다. 비슷한 전략을 사용하지만 여분의 시간을 활용할 수 없는 Constant Utilization Server(CUS)가 Deng에 의해 제안되었다[17]. 이 서버는 비선점형(non-preemptive) 태스크들을 스케줄링하기 위해 고안되었다. 이들 두 서버는 스케줄링을 위해 비주기적으로 도착하는 작업들의 실행 시간을 알아야 하는 제약이 있다. 이러한 문제점을 보완하기 위해 Abeni [18]는 도착 작업들의 실행 시간을 예측하기 힘든 멀티미디어 데이터를 처리하는 비주기적 태스크들을 스케줄링하기 위한 Constant Bandwidth Server(CBS)를 제안했다. 이는 사전에 정의된 서버의 크기만을 이용해 작업들을 스케줄링하므로 도착 작업들의 실행 시간을 필요로 하지 않는다.

3. 실시간 스케줄러 모델

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 제안되었던 기존 모델을 기반으로 비주기적 태스크를 지원할 수 있는 확장된 스케줄러 모델을 새로이 제안하고, 구성 요소들이 사용하는 알고리즘에 대해 논의한다.

3.1 기존 제안 스케줄러 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.

이러한 아이디어를 기반으로 두개의 구성 요소로 이루어진 재구성이 가능한 스케줄러 모델이 참고문헌 [4, 5]에서 제안되었다. 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 하위 단계의 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.

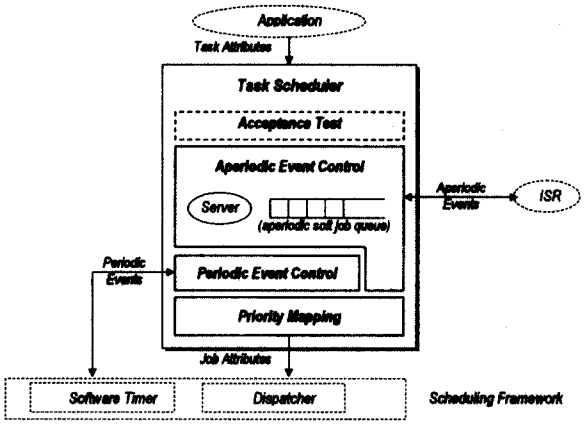
그러나 이 모델은 주기적 태스크를 스케줄링 할 수 있는 기본 틀만 제공할 뿐, 주기적 경성/연성 태스크와 비주기적 경성/연성 태스크 등이 복합적으로 구성된 시스템을 지원하기에는 한계가 있다. 즉, 주어진 기본 틀을 바탕으로 복합 태스크 스케줄링 알고리즘을 시스템 개발자가 직접 구현해야 하는 어려움이 있다. 따라서 본 연구에서는 이러한 복합 태스크 집합을 스케줄링 할 수 있는 확장된 스케줄러 모델을 제안하고자 한다.

3.2 비주기적 태스크 서버를 위한 확장된 태스크 스케줄러 모델

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. 태스크 스케줄러는 태스크 속성을 작업 속성으로 변환하는 기능을 담당한다. 태스크 스케줄러는 새로운 경성 태스크에 대한 수용 여부를 결정하고, 각 태스크의 새로운 작업 도착을 감시하면서 새로운 작업에 대해 작업 속성을 생성한다. 태스크 스

케줄러의 이러한 행위는 스케줄링 알고리즘에 의해 결정된다. 스케줄링 알고리즘은 태스크 스케줄러로 하여금 언제, 어떤 작업에게, 얼마 만큼의 프로세서 시간을, 그리고 어떤 우선권을 부여할 것인지를 결정하게 한다. 태스크 스케줄러는 하위 단계의 스케줄링 Framework를 기반으로 하나의 특정 스케줄링 알고리즘을 구현한다. 다른 요구 조건을 가진 다양한 실시간 응용을 만족시키기 위해서는 응용별 적합한 여러 종류의 알고리즘들이 필요하며, 이들은 동일한 스케줄링 Framework를 기반으로 하나의 독립된 모듈로 존재할 수 있다.

태스크 스케줄러가 RM 알고리즘을 지원한다면, 이는 주기적으로 수행해야 하는 태스크에 대해 매 주기의 시작을 타이머에 설정해야 한다. 설정한 타이머가 작동하거나 또는 외부로부터 비주기적인 이벤트가 발생할 경우, 해당 태스크의 작업을 수행하기 위해 필요로 하는 프로세서 시간, 작업의 우선순위, 그리고 작업의 절대 만기 등을 결정한다. 그런 후, 해당 작업과 함께 앞서 결정된 작업 속성들을 정해진 스케줄러 내부 인터페이스를 통해 스케줄링 Framework에게 넘겨 준다. 이러한 태스크 스케줄러의 행위는 분리 가능한 네 가지 독립적인 기능을 가진다. (그림 1)은 태스크 스케줄러의 기능을 계층적으로 분리하여 도시화한 것이다.



(그림 1) 태스크 스케줄러의 기능적 모델

새로 생성되는 모든 경성 태스크들은 이들의 만기가 준수될 수 있는지를 검증 받기 위해 수용 테스트를 거쳐야 한다. 이 테스트는 기존 주기적 태스크들과 사전에 승인된 다른 비주기적 경성 태스크들의 만기를 준수하면서 새로운 태스크를 만기 내에 실행할 수 있는지를 체크 한다. 수용 테스트를 통과하지 못한 태스크들은 실행이 거절되고, 이러한 사실은 사전에 정의된 절차에 의해 사용자에게 통보된다. 이미 사전에 off-line으로 스케줄 가능성이 검증되고, 동적으로 태스크들을 생성/삭제할 필요가 없는 정적 시스템에서는 굳이 시스템 실행 중에 이러한 테스트를 수행할 필요는 없다. 따라서 태스크 스케줄러 내의 이 기능은 시스템의 적용 환경과 기반 스케줄링 알고리즘에 따라 시스템 구성시 선택적으로 추가/삭제가 가능하게 해야 한다.

주기적 태스크에 대해서는 주기적으로 각 태스크를 실행시켜야 한다 ((그림 1)의 주기적 이벤트 제어 부분). 이를 위해 태스크 스케줄러는 사전에 각 태스크의 주기 시작 시간을 타이머에 설정한다. 설정된 시간 순서에 따라 타이머가 작동하면 해당 태스크의 새로운 작업을 생성한다. 태스크의 최악의 경우 실행 시간을 작업의 실행 예산 속성으로 설정한다. 또한 그 태스크의 다음 작업을 위해 다음 주기의 시작 시간을 소프트웨어 타이머에 미리 설정한다.

비주기적 태스크에 대해서는 해당 태스크로 이벤트가 전달될 때 ((그림 1)의 비주기적 이벤트 제어 부분), 그 태스크가 경성 또는 연성 태스크인가에 따라 처리가 달라진다. 경성 태스크인 경우, 주기적 태스크와 같이 해당 태스크의 새로운 작업을 생성하고, 그 작업의 실행 예산 속성으로 태스크의 실행 시간 속성을 설정한다. 그러나 연성 태스크인 경우, 그 태스크의 작업을 직접 스케줄링하여 실행하는 것이 아니라, 이를 대신 수행해 주는 전용 서버에게 의뢰해야 한다. 서버는 이러한 목적으로 도착한 비주기적 연성 작업을 보관하고 관리하는 하나의 작업 큐를 가진다. 의뢰된 연성 작업들은 이 큐에 삽입되며, 이들은 FIFO 방식으로 서버에 의해 서비스된다. 서버의 작업 큐에서 관리되는 비주기 작업은 해당 태스크가 수행하는 일(함수 또는 코드)을 추상적으로 표현한 일 단위이며, 하부 단계의 스케줄링 Framework에 의해 실제적으로 스케줄링 되는 단위이기도 하다. 즉, 비주기적 작업을 서비스하는 서버가 실질적으로 스케줄링 되는 것이 아니라, 비주기적 작업 자체가 스케줄링 된다. 서버는 단지 매번 실행될 비주기적 작업의 실행 예산과 만기 등을 할당하고, 실행 예산이 소진되었을 경우 이의 보충 시간을 적절히 제어하는 역할을 담당한다.

실행 예산이 설정된 작업들은 마지막 단계로 스케줄링 Framework의 Dispatcher에 의해 사용될 우선순위를 할당 받아야 한다. 이 우선순위는 기반 스케줄링 알고리즘별 할당 방법이 모두 다르다. 고정 우선순위 스케줄링 알고리즘(e.g., RM)인 경우, 해당 태스크 생성시 부여된 우선순위를 그대로 활용하면 된다. 그러나 동적 우선순위 알고리즘(e.g., EDF)은 작업이 생성될 때마다 그 태스크의 앞 전 작업과는 다른 우선순위(절대 만기)가 부여된다. 따라서 이 경우에는 작업의 절대 만기를 우선순위 값으로 설정한다. 시간 유도형(i.e., Cyclic Scheduler)은 시스템 내에 실행 가능한 작업은 항상 하나 뿐이므로, 우선순위의 의미는 없다[20].

3.3 태스크 스케줄러 제어 알고리즘

우선순위 기반형 태스크 스케줄러는 기본적으로 (그림 2)의 알고리즘에 따라 실행한다. 전체적으로 우선순위 기반형 스케줄링 알고리즘들은 모두가 이와 같은 방식으로 작동되

며, 우선순위 결정과 비주기적 연성 작업을 위한 서버 알고리즘에서 구체적으로 차이가 발생한다. 알고리즘에서 **요청**은 상위 단계의 응용으로부터 도착하는 것이고, **이벤트**는 이벤트 관리자 또는 하위 단계의 스케줄링 Framework으로부터 들어오는 것이다.

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.
  - 스케줄링 Framework에게 첫 작업의 시작 시간으로 활성화 대기 요청을 보낸다.
- **작업 활성화** 이벤트 :
  - 주기적 작업인 경우, 작업의 실행 예산을 설정하고, 우선순위를 결정한 후, 스케줄링 Framework에게 실행 준비 요청을 보낸다.
  - 서버 태스크인 경우, 서버에게 예산 보충 통보를 보낸다.
- **예산 소진** 이벤트 : 해당 서버에게 예산 소진 통보를 보낸다.
- **비주기적 작업** 도착 이벤트 :
  - 비주기적 경성 작업인 경우, 작업의 실행 예산과 절대 만기를 설정하고 우선순위를 결정한 후, 스케줄링 Framework에게 실행 준비 요청을 보낸다.
  - 비주기적 연성 작업인 경우, 해당 서버에게 작업 도착 통보를 보낸다.
- **실행 완료** 요청 :
  - 주기적 작업인 경우, 태스크의 다음 주기 시작 시간을 작업의 다음 시작 시간으로 설정하여, 스케줄링 Framework에게 활성화 대 기 요청을 보낸다.
  - 비주기적 연성 작업인 경우, 해당 서버에게 실행 완료 요청을 보낸다.
  - 비주기적 경성 작업인 경우, 해당 작업이 다음 동기화 이벤트를 기다리도록 이벤트 관리자에게 동기화 이벤트 대기 요청을 한다.

(그림 2) 우선순위 기반형 태스크 스케줄러 제어 알고리즘

이벤트 관리자는 동기화 이벤트 대기 요청이 들어 올 경우, 다른 태스크나 ISR(interrupt service routine)로부터 해당 태스크로 소프트웨어 이벤트가 도착할 때까지 그 태스크를 블록킹 시킨다.

- **서버 초기화** : - 실행 예산 = 0; 절대 만기 = 0.
- **서버 스케줄링** :
  - 실행 예산이 존재하고 비주기적 작업 큐에 준비중인 작업이 존재할 경우, 서버 알고리즘에 따라 비주기적 작업의 실행 예산, 절대 만기, 우선순위를 결정하고, 스케줄링 Framework에게 해당 작업의 실행 준비 요청을 보낸다.
- **예산 보충 통보** :
  - 현 서버의 실행 예산을 보충하고, 이미 서비스 중인 작업이 존재하지 않을 경우 서버 스케줄링을 호출한다. 이 통보는 서버 알고리즘에 따라 무시될 수도 있다.
- **예산 소진 통보** :
  - 현재 서비스 중인 작업의 실행을 잠시 중단한다. 중단된 작업은 이후 예산 보충 통보가 도착되면 재실행될 수 있다. 이 통보는 근본적으로 스케줄링 Framework으로부터 발생되며, 서버에 따라 무시될 수도 있다.
- **작업 도착 통보** :
  - 해당 작업을 비주기적 작업 큐의 맨 뒤에 삽입한다. (FIFO 서비스)
  - 비주기적 작업 큐에 새로 도착한 작업 뿐일 경우 (즉, 도착 이전에 큐가 비어 있을 경우에), 서버 스케줄링을 호출한다.
- **실행 완료 요청** :
  - 완료된 작업을 비주기적 작업 큐로부터 제거하고, 다음 동기화 이벤트를 기다리도록 이벤트 관리자에게 동기화 이벤트 대기 요청을 한다.
  - 수행할 작업이 더 이상 없을 경우, 아무것도 수행하지 않는다. (휴면 상태)
  - 그렇지 않은 경우, 작업 큐의 맨 처음 작업에 대해, 서버 스케줄링을 호출한다.

(그림 3) 비주기적 태스크 서버의 제어 알고리즘

비주기적 태스크 서버들은 기반 태스크 스케줄러나 이벤트 관리자로부터 4가지 종류의 서로 다른 요청/통보를 받는다. 해당 서버의 알고리즘에 따라 각 요청/통보는 서로 다르게 처리 된다. (그림 3)은 비주기적 태스크 서버의 일반적인 제어 알고리즘이다.

4. 스케줄러 모델의 구현

기존 제안 모델은 응용과 커널간의 응용 프로그래밍 인터페이스(API), 커널과 스케줄러간의 태스크 스케줄러 인터페이스, 스케줄러와 스케줄링 Framework간의 스케줄링 Framework 인터페이스 등을 제공한다[4, 5]. 본 연구에서는 비주기적 태스크를 위한 사용자 API와 비주기적 태스크 서버를 위한 스케줄러 내부 표준 인터페이스를 새로이 규정하였다. 본 절에서는 이러한 인터페이스에 대해 살펴 보고, 재구성을 위한 이들간의 연관 관계를 고찰해 본다.

4.1 응용 프로그래밍 인터페이스

제안 스케줄러 모델은 주기적/비주기적 태스크들, 경성/연성 태스크들, 그리고 여러 종류의 주기적 태스크 서버들을 지원한다. 또한 비주기적 태스크와 interrupt service routine(ISR)간 동기화 메커니즘인 event도 지원한다. 따라서 제안 모델은 실시간 응용을 위한 태스크 *API*와 *event API*를 제공한다. 태스크 API는 POSIX API [23]를 준수했으며, 기존 POSIX에서 지원되지 않거나 의미가 변경된 API는 끝에 _np(non-portable)를 추가했다((그림 4) 참조).

```
/* Task API */
pthread_create (*thread, *attributes, (*start_routine)(*), *arg)
pthread_create_server_np (*server, server_type, phase, period, wcet)
pthread_delete_np (*thread)
pthread_make_periodic_np (*thread, phase, period, wcet, deadline)
pthread_make_sporadic_np (*thread, *event, utilization, wcet)
pthread_make_aperiodic_np (*thread, *server, *event, wcet)
pthread_wait_np()

/* Event API */
rtl_event_init (*event), rtl_event_destroy (*event)
rtl_event_wait (*event)
rtl_event_postv (*event)
```

(그림 4) 응용 프로그래밍 인터페이스 (API)

태스크 API들은 실시간 응용이 각 태스크에 대해 태스크 속성을 설정하고, 태스크의 생성 및 종료 등을 제어할 때 사용된다. 매개 변수들 중 phase는 태스크의 첫번째 작업의 실행 시작 시간, period는 주기, wcet는 최악의 경우 실행 시간, deadline은 상대 만기, utilization은 프로세서 이용률을 의미한다. 모든 태스크들은 처음에 pthread_create()를 통해 비실시간 태스크로 생성된 후, 나중에 별도의 API를 통해 실시간 태스크로 전환된다. pthread_create_server_np()는 주어진 시작 시간, 서버 크기 등을 가진 비주기적 태스크 서버를 생성한다. 서버의 크기(wcet/period)는 서버가 사용할 수 있는 최대 프로세서 이용률이다. 서버의 종류(server_type)로는 주기적 태스크 스케줄러가 RM인 경우 RTL_SERVER_DS, RTL_SERVER_SS를, EDF인 경우RTL_SERVER_TBS, RTL_SERVER_CUS, RTL_SERVER_CBS, RTL_SERVER_ DDS 등을 지정할 수 있다. period는 서버가 DS, SS, DDS (dynamic deferrable server)인 경우 서버의 주기로 활용되지만, 그 외의 경우는 서버 크기를 결정할 때 사용된다.

pthread_make_periodic_np(), pthread_make_sporadic_np(), pthread_make_aperiodic_np()는 pthread_create()에 의해 생성된 비실시간 태스크를 각각 주기적 경성, 비주기적 경성, 비주기적 연성 실시간 태스크로 전환시키는 함수들이다. 이들 매개 변수들 중 event는 비주기적 태스크가 대기해야 할 이벤트 개체이다. wcet은 서버가 TBS, CUS의 경우에만 지정 가능하다. 또한 비주기적 연성 태스크인 경우 해당 태스크를 서비스할 server도 지정해 주어야 한다. pthread_wait_np()는 해당 태스크의 현 작업의 실행이 끝났음을 통보하는 함수이다. 이는 주기적 태스크인 경우 다음 주기의 시작 때까지 대기하게 하고, 비주기적 태스크인 경우 다음 이벤트의 발생 때까지 대기시킨다.

Event API들은 주로 ISR에서 비주기적 경성/연성 태스크에게 S/W 이벤트를 전달하기 위해 사용된다. rtl_event_wait()는 현 작업을 완료한 비주기적 태스크가 다음 S/W 이벤트를 대기하고자 할 때, rtl_event_post()는 ISR에서 주어진 이벤트가 발생했다는 사실을 통보할 때 사용한다.

4.2 비주기적 태스크 서버의 구현

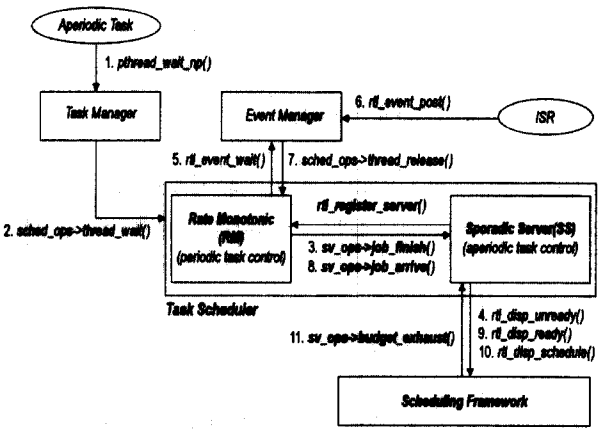
비주기적 태스크 서버 인터페이스는 태스크 스케줄러 내부의 주기적 태스크 제어부와 비주기적 태스크 제어부(서버) 사이에 준수해야 할 인터페이스이며, 특정 비주기적 태스크 서버를 구현하는 서버 모듈이 제공해야 하는 함수들의 집합이다. 시스템 개발자는 사전에 정의된 이 인터페이스를 이용하여 프로그래밍할 수 있으며, 이를 통해 상호 독립적인 다양한 서버들을 개발할 수 있다.

비주기적 태스크 서버는 (그림 5)의 인터페이스 함수들을 정의한 후, 이를 서버 초기화 때 관련 등록 함수(rtl_register_server())를 통해 시스템에 등록하여야 한다. 매개 변수 server_ type은 등록하고자 하는 서버의 종류를 의미한다. 시스템 개발자가 새로운 서버를 개발할 경우 별도의 서버 종류를 정의하여 추가할 수 있다. 서버 등록은 커널에서 제공되는 기능이다. 서버는 시스템 재구성시 정적 또는 동적으로 시스템에 삽입/삭제 가능하다. 하나의 시스템에 여러 개의 서버를 등록할 수 있으며, 현재 구동 중인 스케줄러를 기반으로 하는 서버들은 동시에 운영될 수 있다.

(그림 5)의 서버 인터페이스는 모두 해당 태스크가 비주기적 태스크인 경우에 주기적 태스크 제어부에 의해 호출된다. 서버에 의해 서비스되는 비주기적 태스크가 pthread_wait_np()를 호출한 경우의 태스크 관리자, 태스크 스케줄러, 이벤트 관리자, 스케줄링 Framework간의 인터페이스 호출 예는 (그림 6)과 같다.

```
/* API function set provided by a specific server */
   typedef struct {
        void (* sv_init)(* thread);
        void (* sv_deinit)(* thread);
        void (* job_arrive)(* thread);
        void (* job_finish)(* thread);
        void (* budget_exhaust)(* thread);
   } rtl_server_ops_t;
/* 비주기적 태스크 서버 모듈 초기화시 호출해야 할 등록 함수 */
rtl_register_server(server_type, rtl_server_ops_t *sv_ops);
rtl_unregister_server(server_type);
```

(그림 5) 비주기적 태스크 서버 인터페이스와 관련 등록 함수



(그림 6) 비주기적 태스크 서비스를 위한 인터페이스 호출 예제

태스크 스케줄러 내부의 주기적 태스크 관리부에는 RM 스케줄링 알고리즘이, 비주기적 태스크 관리부에는 SS가 구현되어 있다. 여기서 태스크 관리자는 rtl_register_scheduler()를 통해 등록된 현 스케줄러의 sched_ops -> thread_wait() 멤버 함수를 호출해 주는 단순한 중계자 역할만 담당한다. 태스크 스케줄러는 해당 태스크가 주기적 태스크인지, 비주기적 태스크인지를 확인한다. 비주기적 태스크인 경우, 서버의 sv_ops -> job_finish()를 호출하여 현재 서비스되는 작업의 실행을 중

단하게 하고 다음 이벤트를 기다리게 하기 위해, rtl_event_wait()를 자동으로 호출한다. 임의의 시간이 지난 후, 비주기적 태스크가 기다리는 이벤트의 발생을 ISR에 의해 통보(rtl_event_post()) 받으면, 이벤트 관리자는 이를 sched_ops -> thread_release()를 통해 태스크 스케줄러에게 통보한다. 이어 스케줄러는 sv_ops -> job_arrive()을 통해 서버에게 새로운 작업이 도착했음을 알린다. 서버는 해당 알고리즘에 따라 비주기적 작업을 서비스 해준다. 비주기적 작업이 서비스될 때는 매번 사용 가능한 실행 예산이 주어지며, 이 시간을 초과하여 실행되는 경우에는 sv_ops -> budget_ exhaust()를 통해 스케줄링 Frame work으로부터 이 사실을 통보 받는다.

5. 제안 모델의 타당성 검증

본 절에서는 새로운 비주기적 태스크 서버를 하부의 주기적 태스크 스케줄러나 복잡한 커널 메커니즘 수정 없이 독립적으로 개발될 수 있는가를 확인하고, 또한 제안 모델상에서 비주기적 태스크를 스케줄링하는 오버헤드가 기존의 단일 스케줄러상의 비주기적 태스크 스케줄링 오버헤드와 비교해 볼 때 얼마만큼의 차이가 있는가를 확인함으로써 제안 모델의 타당성을 검증하고자 한다.

본 연구에서는 다양한 비주기적 태스크 서버를 지원할 수 있는 제안된 실시간 스케줄러 모델을 Real-Time Linux (RT-Linux)상에 시험적으로 구현하였다. 기반 RT-Linux는 Linux 2.2.14에 구현된 RT-Linux 2.2를 수정하여 사용했다. 본 연구에서는 기존에 구현된 태스크 스케줄러를 본 연구에서 제안된 비주기적 태스크 서버를 위한 스케줄러 모델을 수용하도록 수정/확장하였다.

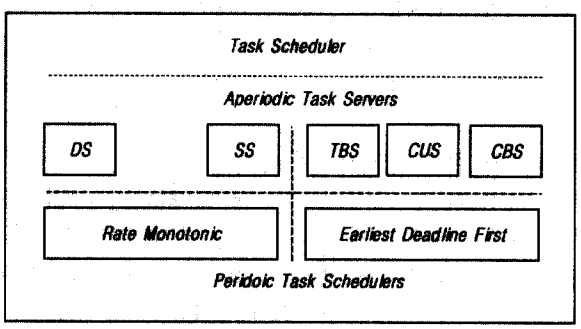
5.1 비주기적 태스크 서버의 개발 및 재구성 용이성

기존 연구[4, 5]에서 제안된 스케줄러 모델은 주기적 태스크를 스케줄링 할 수 있는 기본 틀만 제공한다. 따라서 주기적 경성/연성 태스크와 비주기적 경성/연성 태스크 등이 복합적으로 구성된 시스템을 지원하기 위해서는 주어진 스케줄링 Framework을 바탕으로 복합 태스크 스케줄링 알고리즘을 시스템 개발자가 직접 구현해야 하는 문제점이 있다. 즉, 기존의 주기적 태스크 스케줄러를 수정하여 비주기적 태스크 서버를 새로이 개발하여야 하며, 이를 위해서 기존 스케줄러를 완벽히 이해해야 한다. 또한 주기적 태스크 스케줄러 내에 구현된 비주기적 서버들은 기반 주기적 태스크 스케줄러와 밀접한 연관을 가지므로 필요에 따라 다른 특성을 가진 서버들로 교체해야 할 경우 스케줄러 전체를 수정해야 하는 어려움이 따른다.

따라서 본 연구에서는 이러한 복합 태스크 집합을 스케줄링할 수 있는 스케줄러를 보다 쉽게 개발할 수 있도록 기존 스케줄러 모델을 확장하였다. 이를 위해 기존 단일 태스크 스케줄러를 주기적 태스크 제어부와 비주기적 태스크 제어부로 분리하였으며((그림 1)참조), 이들간 내부 표준 인터페이스((그림 5)의 비주기적 태스크 서버 인터페이스 참조)를 새로이 정의하여 서로간 독립된 모듈로 개발 가능하도록 하였다. 즉, 제안 모델은 대부분의 실시간 커널에서 복잡하게 결합되어 하나의 스케줄러를 구성하던 잠재적 구성 요소들을 기능별로 명확히 구분한 후, 이를 독립된 모듈로 분리하고(하위단계의 스케줄링 Framework, 상위단계의 주기적 태스크 제어부와 비주기적 태스크 서버 제어부) 이들간 준수해야 할 내부 표준 인터페이스를 정의하였다.

이러한 전략의 타당성을 확인하기 위해 본 연구에서는 사전에 정의된 표준 인터페이스를 준수하는 다양한 비주기적 태스크 서버들을 구현하여 보았다. 주기적 태스크 스케줄러로는 기존 실시간 시스템에서 많이 사용되는RM, EDF 스케줄러를 기반으로 하고, 비주기적 태스크를 서비스하는 TBS(total bandwidth server), CBS(constant bandwidth server), CUS(constant utilization server), DDS(dynamic deferrable server), DS(deferrable server), SS(sporadic server) 등의 다양한 서버들을 구현하였다. (그림 7)은 제안 모델의 타당성을 검증하기 위해 시험적으로 구현된 주기적 태스크 스케줄링 알고리즘별 적용 가능한 다양한 비주기적 서버들을 보이고 있다. 이중 DS와 SS는 RM 스케줄러를 기반으로 구현된 서버이고, 나머지 서버들은 EDF 스케줄러를 기반으로 구현되었다. 이러한 서버들은 복잡한 하위단계의 커널 모듈을 수정하지 않고도 제안 모델이 제공하는 스케줄링 Framework, 이벤트 제어부, 상위단계의 주기적 태스크 제어부에서 제공하는 서비스 함수들만을 이용하였다.

이상의 시험적 구현을 통해 기반이 되는 주기적 태스크 스케줄러에 영향을 받지 않고 제안 모델이 제공하는 서비스 함수와 동일한 비주기적 태스크 서버 인터페이스를 준수하면서 다양한 특성을 가진 서로 다른 서버들을 손쉽게 구현할 수 있다는 것을 확인하였다.



(그림 7) 시험적으로 구현된 비주기적 태스크 서버들

제안 모델의 또 하나의 특징은 제안된 모델을 바탕으로 다양한 주기적 태스크 스케줄러와 비주기적 태스크 서버들이 사전에 개발되어 있을 경우, 시스템 구축자는 필요에 따라 구축하고자 하는 실시간 시스템의 특성에 맞게 적절한

스케줄러와 서버들을 선택적으로 골라 시스템을 구성할 수 있다. 이는 제안 모델이 구성 요소들을 기능별로 독립된 모듈로 분리하여 지원하기 때문에 가능하며, 이로 인해 기존 개발된 소프트웨어 모듈들의 재사용과 시스템 재구성을 보다 용이하게 해주는 효과가 있다.

5.2 비주기적 태스크 스케줄링 오버헤드(overhead)

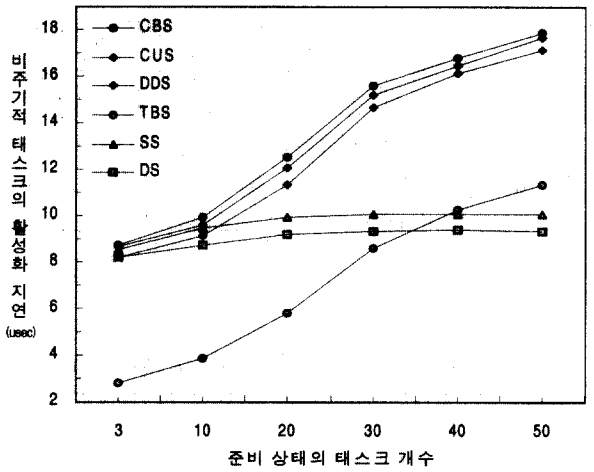
본 절에서는 단일 태스크 스케줄러상에서 주기적 태스크와 비주기적 태스크 서버를 복합적으로 스케줄링하는 기존 방법과 독립되고 구조화된 전략을 채택한 제안 모델의 스케줄링 오버헤드를 서로 비교함으로써 제안 모델의 타당성을 검증하는데 목적이 있다.

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[3]. 그러나 본 연구에서는 펜티엄 프로세서에서 제공하는 TSC(time stamp counter)를 이용하였다.

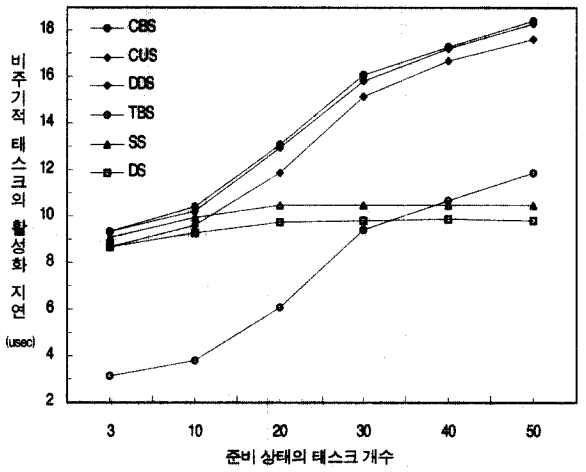
실험에서는 비주기적 태스크 서버로 앞서 개발 용이성을 확인하기 위해 시험적으로 구현된 TBS, CBS, CUS, DDS, DS, SS 등을 사용하였고, 기반 주기적 태스크 스케줄러로는 각각 RM과 EDF를 사용하였다. 스케줄링 오버헤드로 비주기적 태스크의 활성화/비활성화 지연을 측정하였다.

비주기적 태스크의 활성화 지연이란 비주기적 태스크에 이벤트가 전달되기(rtl_event_post() 호출) 직전부터 이 태스크가 최초로 서비스되기까지 소요된 총 시간을 말한다. 이 시간은 소프트웨어 이벤트를 비주기적 태스크에 전달하고, 이 태스크를 해당 서버의 비주기적 작업 큐에 삽입한 후, 이 중 가장 먼저 도착한 비주기적 태스크를 선택하여 서비스하는데 소요된 총 시간을 포함한다. 이 실험에서 비주기적 태스크 서버의 우선순위를 가장 높게 설정하였고, 비주기적 작업 큐는 항상 비어 있다. 반대로 비주기적 태스크의 비활성화 지연이란 서버가 현재의 비주기적 태스크를 모두 서비스하고 (pthread_wait_np() 호출), 그 다음 우선순위의 주기적 태스크로 문맥 교환을 실시하는데 소요된 총 시간을 의미한다. 이 시간은 서비스가 끝난 작업을 서버의 비주기적 작업 큐에서 삭제하고, 그 다음으로 높은 우선순위를 가진 주기적 태스크를 선택하는데 소요된 시간을 포함한다.

(그림 8)은 서버별 비주기적 태스크의 활성화 지연을 보여 준다. (그림 (a))는 제안 모델을 사용하지 않고 주기적 태스크 스케줄러 내부에 복합적으로 구현된 기존 비주기적 태스크 서버들의 스케줄링 오버헤드이며, (그림 (b))는 제안 모델을 기반으로 구현된 서버들의 스케줄링 오버헤드이다. 그림에서 x축은 준비 큐(서버의 비주기 작업 큐가 아님)에 이미 도착해 있는 주기적 태스크의 개수이고, y축은 태스크 개수별 활성화 지연 시간이다.



(a) 주기적 태스크 스케줄러 내부에 복합적으로 구현된 서버들



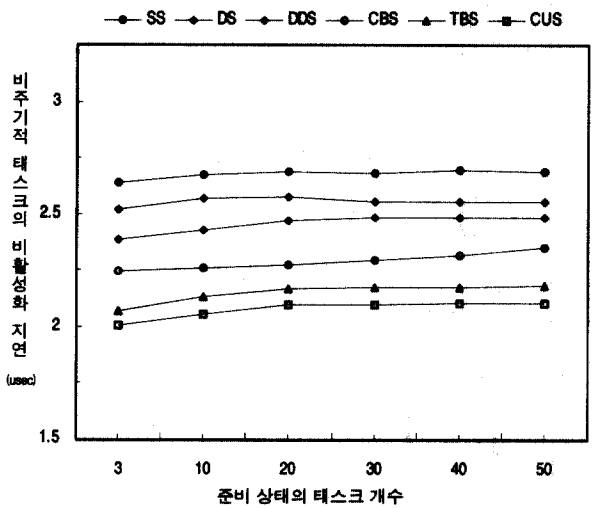
(b) 제안 모델을 기반으로 독립적으로 구현된 서버들

(그림 8) 서버별 비주기적 태스크의 비활성화 지연

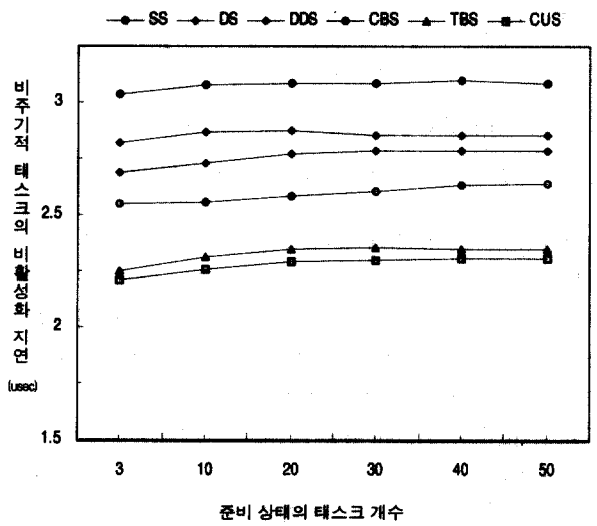
DS와 SS는 준비 상태의 태스크 개수와는 무관하게 일정한 활성화 시간을 보인 반면, TBS, CUS, CBS, DDS 등은 태스크 개수에 비례하여 증가한다. 이는 기반 스케줄러인 RM와 EDF에 의해 영향을 받기 때문이다. 즉, 해당 비주기적 태스크를 준비 큐에 삽입하는데 소요되는 시간 차이인 것으로 판단된다. RM은 태스크들이 항상 고정된 우선순위를 가지고 고정된 dispatching 시간을 보장하는 알고리즘[21]을 사용함으로써, 준비된 태스크의 개수와는 무관하게 항상 고정된 활성화 지연을 가진다. 그러나 EDF의 경우 동적 우선순위를 가지므로 최우선순위 태스크를 찾기 위해 준비된 태스크를 순차적으로 찾아야 한다. 따라서 준비된 태스크 수에 비례하여 증가한다. RM 기반에서는 DS 보다 SS가 더 많은 시간이 소요된다.

그러나 이는 1 msec 이내의 차이이므로 거의 동일한 시간을 가진다. EDF 기반 하에서는 전반적으로 TBS가 가장 짧은 활성화 지연을 보인다. 이는 TBS의 경우 실행 예산을 설정할 필요가 없기 때문이다.

전반적으로 제안 모델상의 스케줄링 오버헤드(그림 b)가 기존 스케줄러(그림 a)보다 0.3~0.6 msec가 더 소요된다. 이는 기존 스케줄러를 다시 주기적 태스크 제어부와 비주기적 태스크 제어부로 분리함으로써 발생하는 오버헤드인 것으로 분석된다. 이는 비주기적 태스크의 스케줄링의 경우 주기적 태스크 제어부에서 직접 처리하지 않고 스케줄링 권한을 비주기적 태스크 제어부로 넘기는 과정에서 발생하는 오버헤드이다.



(a) 주기적 태스크 스케줄러 내부에 복합적으로 구현된 서버들



(b) 제안 모델을 기반으로 독립적으로 구현된 서버들

(그림 9) 서버별 비주기적 태스크의 비활성화 지연

(그림 9)는 서버별 비주기적 태스크의 비활성화 지연을 보여 준다. 비활성화 지연은 준비 상태의 태스크 개수와는 무관하다는 것을 알 수 있다. 이는 [4,5]에서 주기적 태스크의 비활성화 지연에서 확인한 바와 같이 RM와 EDF 모두 동일한 비활성화 지연을 보인 것과 같은 맥락이라 판단된다. RM 기반의 DS와 SS가 대체로 높게 나타나는데, 이는 실행 예산의 소비 및 보충이 RM 기반의 서버가 더 복잡하기 때문이다. 활성화 지연과 마찬가지로 전반적으로 제안 모델상의 비활성화 지연(그림 b)이 주기적 태스크 스케줄러 내부에 복합적으로 구현된 기존 서버상에서의 비활성화 지연보다 0.2~0.3 msec가 더 소요된다. 이 역시 주기적 태스크 제어부에서 직접 처리하지 않고 비주기적 태스크의 비활성화 작업을 비주기적 태스크 제어부로 넘기는 과정에서 발생하는 오버헤드인 것으로 분석된다.

이상의 비주기적 태스크의 활성화/비활성화 지연 실험 결과에서 제안 스케줄러 모델이 스케줄링 Framework을 기반으로 주기적 태스크 스케줄러와 비주기적 태스크 서버를 분리하여 독립된 스케줄링 단위로 존재할지라도 스케줄링 오버헤드는 크지 않다는 것을 확인할 수 있다. 이는 제안된 모델을 기반으로 다양한 스케줄러와 비주기적 태스크 서버를 구현한다 해도 실행시의 오버헤드는 크지 않은 반면, 앞 절에서 논의한 바와 같이 시스템 재구성 및 새로운 스케줄러 개발을 효과적으로 지원할 수 있다는 것을 의미한다.

6. 결 론

본 연구에서는 상위 단계의 태스크 스케줄러와 하위 단계의 스케줄링 Framework으로 구성된 기존 제안 스케줄러 모델을 수정하여, 다양한 비주기적 태스크 서버들을 지원할 수 있는 확장된 스케줄러 모델을 제안하였다. 제안 모델은 기존 태스크 스케줄러를 다시 주기적 태스크 제어부와 비주기적 태스크 제어부로 분리하였으며, 이들간 내부 표준 인터페이스를 새로이 정의하였다. 제안 모델은 대부분의 실시간 커널에서 복잡하게 결합되어 하나의 커널 스케줄러를 구성하던 구성 요소들을 기능별로 재구성이 가능하도록 명확하게 구분하였다.

이러한 구조는 policy/mechanism separation[19] 기법을 제공한다. 상위 계층의 태스크 스케줄러는 정책(policy)을 정의하고, 하위 계층의 스케줄링 Framework은 메커니즘(mechanism)을 제공한다. 스케줄링 정책을 스케줄링 메커니즘으로부터 분리함으로써, 스케줄링 Framework 인터페이스만 준수한다면, 하위 단계의 복잡한 커널 메커니즘을 수정하지 않고도 새로운 정책들을 쉽게 개발할 수 있다. 또한 시스템 재구성을 보다 유연하게 해준다. 태스크 스케줄러를 재구성 가능한 두 단계 모듈(주기적 이벤트 제어부와 비주기적 이벤트 제어부)로 분리함으로써, 응용의 특성에 맞게 시스템을 유효 적절하게 재구성할 수 있다. 즉, 주기

적 태스크 집합으로 구성된 응용일 경우 이에 적합한 주기적 태스크 스케줄러만 선택하여 사용할 수 있고, 여기에 비주기적 태스크들도 함께 스케줄링 해야 할 경우 해당 주기적 태스크 스케줄러를 기반으로 하는 비주기적 태스크 서버를 추가함으로써 이를 지원할 수 있다.

본 연구에서 제안한 스케줄러 모델은 단일 프로세서상에서의 스케줄링을 전제로 하고 있다. 따라서 본 연구팀은 현재 SMP(symmetric multiprocessor) 시스템을 위한 제안 모델의 확장 작업을 추진하고 있으며, 확장 모델이 완성되면 동일한 스케줄링 Framework을 사용하면서 CPU별 서로 다른 태스크 집합과 태스크 스케줄러를 활용할 수 있게 된다. 실시간 시스템 설계시 고려해야 할 또 다른 중요한 요소는 시스템 자원의 효율적인 사용이다. 교착상태를 방지하고 우선순위 역전(priority inversion)으로 인한 블록킹(blocking) 시간을 한정시키기 위해, priority inheritance protocol, priority ceiling protocol, stack resource policy 등과 같은 여러 자원 관리 정책들이 제안되었다. 경성 태스크를 위한 자원 관리 정책은 기반 스케줄링 알고리즘과 별개로 지원될 수 없으며, 스케줄링 알고리즘 설계시 함께 고려되어야 한다. 따라서 본 연구에서는 제안된 스케줄러 모델에서 재구성을 지원하면서 자원과 태스크를 통합적으로 스케줄링 할 수 있는 모델에 대해 연구하고 있다.

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

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

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

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

[7] M. Spuri and G. C. Buttazzo, "Efficient Aperiodic Service under the Earliest Deadline Scheduling," *Proc. of Real-Time Systems Symposium*, pp.2-11, Dec. 1994.

[8] 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.

[9] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *The Journal of Real-Time Systems*, No.1, pp.27-60, 1989.

[10] T. M. Ghazalie and T. P. Baker, "Aperiodic Servers in a Deadline Scheduling Environment," *The Journal of Real-Time Systems*, 9, 1995.

[11] J. P. Lehoczky, L. Sha, and J. K. Stronider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proc. of Real-Time Systems Symposium*, pp.261-270, Dec. 1987.

[12] J. P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed Priority Preemptive Systems," *Proc. of Real-Time Systems Symposium*, pp.110-123, 1992.

[13] S. Ramos-Thuel and J. P. Lehoczky, "On-line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems," *Proc. of Real-Time Systems Symposium*, pp.160-171, 1993.

[14] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling Slack Time in Fixed Priority Preemptive Systems," *Proc. of Real-Time Systems Symposium*, pp.222-231, Dec. 1993.

[15] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, Vol.15, No.10, pp.1261-1269, 1989.

[16] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic Scheduling of Real-Time Tasks under Precedence Constraints," *The Journal of Real-Time Systems, 2*, pp.181-194, 1990.

[17] Z. Deng and Jane W. S. Liu, "Scheduling Real-Time Applications in an Open Environment," *Proc. of the 18th IEEE Real-Time Systems Symposium*, pp.308-319, Dec. 1997.

[18] L. Abeni and G. C. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems," *Proc. of Real-Time Systems Symposium*, Dec. 1998.

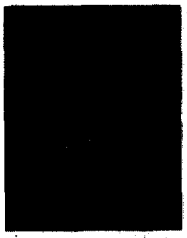
[19] R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Sulf, "Policy/Mechanism Separation in HYDRA," *Proc. of Symposium on Operating Systems Principles*, Nov. 1975.

[20] T. P. Bake and A. Shaw, "The Cyclic Executive Model and Ada," *Proc. IEEE Real-Time Systems Symposium*, pp. 120-129, Dec. 1988.

[21] Jean J. Labrosse, *MicroC/OS-II The Real-Time Kernel*, R&D Books, pp.75-104, 1999.

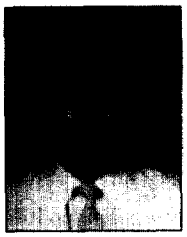
[22] Jane W. S. Liu, *Real-Time Systems*, Prentice-Hall, pp. 190-276, 2000.

[23] IEEE, *Information technology - Portable Operating System Interface (POSIX) Part 1 : System Application Program Interface (API) [C Language]*, ISO/IEC 9945-1 : 1966(E); ANSI/IEEE Std 1003.1, 1996.



심 재 흥

e-mail : jhshim@cesys.ajou.ac.kr
1987년 서울대학교 전산과학과 졸업(학사)
1989년 아주대학교 컴퓨터공학과 졸업(석사)
1989년~1994년 서울시스템 공학연구소
2001년 아주대학교 컴퓨터공학과 졸업 (박사)
2001년~현재 아주대학교 정보통신전문대학원, Post Doc
관심분야 : 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템

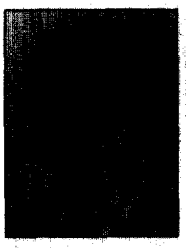


김 영 일

e-mail : yikim@kerinet.re.kr
1981년 충남대학교 전자교육공학과 졸업 (학사)
1992년 아주대학교 전자계산학과 졸업(석사)
1989년~1998년 경기도과학교육원, 경기도 교육정보연구원
1999년~현재 경기도교육청 장학사
1995년~현재 아주대학교 컴퓨터공학과 박사과정(수료)
관심분야 : 운영 체제, 분산시스템, 실시간 및 멀티미디어시스템

최 경 희

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



정 기 현

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

유 해 영

e-mail : yoohy@dankook.ac.kr
1979년 단국대학교 수학과 졸업(학사)
1981년 단국대학교 수학과 졸업(석사)
1994년~현재 아주대학교 컴퓨터공학과 박사과정 수료
1983년~현재 단국대학교 전산통계학과 교수
관심분야 : 소프트웨어 공학, 시스템 프로그램 등