스케줄러 선택기반의 실시간 리눅스의 성능분석☆

Performance Analysis of Scheduler Selection based Real-time Linux Systems

강 민 구*
Min-goo Kang

요 약

본 논문에서는 스케줄러 선택방식 기반의 실시간 리눅스 시스템에서 비율단조(RMS)와 마감시간우선(EDF) 중에서 사용자가 하나를 선택함으로서, 개선된 스케줄링 검사가 가능하고 태스크 특성에 맞는 스케줄링 알고리듬을 제안하였다. 스케줄러 선택방식의 성능분석을 위해 다양한 프로세서 이용률을 갖는 태스크의 평균 응답 시간과 마감시간에 따라 효율적인 태스크 스케줄링 방식의 성능을 분석하였다.

Abstract

In this paper, an effective task scheduling scheme was proposed for the flexible real time LINUX systems with the selection between EDF(earliest deadline first) and RMS(rate monotonic scheduling). It was known that many task scheduling schemes were analyzed according to the characteristics of scheduling schemes and the guarantee of an earliest deadline scheduler for process utilities.

☞ Keyword : real time LINUX, RMS Scheduler, EDF Scheduler, Task 실시간 리눅스, 비율단조(RMS) 스케줄러, 마감시간우선(EDF) 스케줄러, 태스크

1. 서 론

실시간 리눅스 시스템(RTOS)은 일반적인 시스템과는 달리 태스크의 수행결과의 정확도뿐만 아니라 태스크가 마감시간 내에 실행될 수 있는 지가 중요한 성능의 척도가 된다.

기존의 일반 운영체제의 스케줄링 방식은 시분할 환경에 적합하도록 고안 되었으며 스케줄러는 프로세스들의 우선순위를 주기적으로 재계산하여 각 프로세스들이 프로세서를 공평하게 공유할 수 있도록 설계하였다[1].

그러나, 실시간 시스템에서 이러한 비 실시간 스케줄링 방법으로는 시간의 제약성을 만족시킬 수 없다. 실시간 시스템의 스케줄링은 시간 제약성과 정확성을 보장하기 위해서 우선순위에 따르는 선점 스케줄링이 필요하다[2].

실시간 태스크들은 마감시간 내에 수행되는 것이 보장되고 수행이 될 수 있다는 것이 예측가능해야 한다. 현재 실시간 리눅스를 위한 스케줄러로는 비율단조(RMS) 스케줄러와 마감시간 우선(EDF)스케줄러 두 가지가 별도로 구현되어 있다. 이 두 가지 스케줄러 중에서 사용자는 두 가지 방법 중 하나를 선택하여 사용하고 있다[3].

이로 인해 실시간 시스템의 스케줄링 가능성 검사의 미수행으로 마감시간 실패율을 증가시키는 결과를 초래한다[4].

또한 현재 실시간 리눅스에서는 스케줄 불가능한 태스크들을 스케줄 함으로서 시스템 자체가 멈춰버리는 정지현상이 발생할 수 있다. 이러한 현상은 실시간 시스템에서는 매우 치명적이다[5]. 본 논문에서는 이러한 현상을 고려하여 보다 유연성 있고 개선된 스케줄링 가능성 검사를 통하

* 종신회원 : 한신대학교 정보통신학과 교수
kangmg@hs.ac.kr
[2006/11/14 투고 - 2006/12/14 심사 - 2006/01/25 심사완료]"
☆ 이 논문은 2006년도 한신 대학교 학술연구비 지원에 의하여 연구되었음

여 태스크의 특성에 맞는 스케줄링 방법을 선택하도록 하여 마감시간을 보장하고 효율적인 태스크 스케줄링 방법을 제안한다.

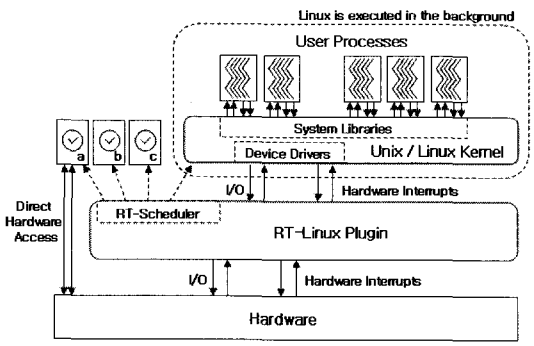
II. 실시간 리눅스에서 스케줄링 비교

2.1 실시간 리눅스 구현

실시간 리눅스에서 실시간 태스크를 스케줄링할 수 있는 메커니즘을 제공해 주고 있다.

리눅스가 실시간 기능을 갖추려면 실제로 커널부터 다시 설계되어야 하겠지만, 이는 커널을 구성하는데 많은 시간을 필요로 하며, 리눅스 커널이 업그레이드 될 때마다 이를 신속히 반영하기 어렵고, 버그가 생길 가능성이 더 커지게 된다.

이에 실시간 리눅스는 커널을 다시 설계하지 않고 리눅스 소스 변경을 최소화하는 방향으로 진행되었다. 리눅스 아래에 실시간 리눅스 커널을 올리고 기존 리눅스 커널을 실시간 리눅스 커널에서 돌아가는 하나의 프로세스로 만들어서 실시간 태스크와 기존의 리눅스 커널이 공존하는 형태를 만들었다.



〔그림 1〕 실시간 리눅스의 커널 구성

실시간 리눅스 커널은 실시간 프로세스를 생성하고 이들을 스케줄링하며, 인터럽트가 발생하였을 때 이를 처리하며 리눅스와의 통신을 담당하는 역할을 한다. 기존 리눅스 커널은 일반 리눅스 프로세스를 관리하고, 자신의 인터럽트를 처리한다. 실시간 리눅스는 [그림 1]과 같이 구성되어 있다[1].

실시간 리눅스의 구현 특징은 다음과 같다[1].

- 리눅스 커널 태스크 : 실시간 리눅스는 기존 리눅스 커널을 실시간 리눅스 상에서 돌아가는 하나의 태스크로 생각한다. 여기에는 가장 낮은 우선순위가 부여되므로, 실행할 수 있는 실시간 태스크가 하나도 없을 때 실행된다.

- 스케줄링 : 실시간 리눅스는 현재 두 가지 스케줄링 방법을 제공한다. 하나는 우선순위 기반 선점 스케줄러로서 모든 태스크에 우선순위를 부여하고, 실행할 수 있는 태스크 중에서 가장 우선순위가 높은 태스크를 수행하며, 이보다 더 높은 우선순위를 갖는 태스크가 등장하면 이를 곧바로 수행한다.

다른 스케줄러는 마감시간 우선(EDF, Earliest Deadline First) 알고리듬을 사용하며, 우선순위가 아니라 마감시간을 기준으로 스케줄링 한다.[2].

2.2 실시간 스케줄링과 선택알고리듬 비교

스케줄링 측면에서 주어진 태스크 집합에 대하여 마감시간을 만족할 수 있는 지의 여부를 조사하는 것을 스케줄링 가능성 검사라고 한다.

만약 주어진 태스크 집합의 모든 태스크들이 마감시간 내에 실행을 완료할 수 있다면 그 태스크 집합은 스케줄링 가능하다고 한다. 스케줄링 가능성 검사를 예측하기 위해서는 간단한 수식을 통하여 판단 가능한 분석적 방법을 많이 사용한다.

분석적 방법을 통한 스케줄링 가능성 검사는 우선순위 구동 방식의 알고리듬들에 유용하게 사용할 수 있는 기법이다. 시스템 설계자는 설계 당시 시스템에 적합한 스케줄링 기법을 선택하고 아래 표와 같은 태스크 주기, 실행 시간, 마감 시간 등을 결정할 때, 분석적 방법을 이용하면 쉽게 시스템의 스케줄링 가능성 여부를 판단할 수 있다[1].

〔표 1〕 스케줄링 선택방법의 분석위한 표기법

표기법	의미
n	태스크 집합의 태스크 개수
t_i	태스크 집합의 태스크 $(1 \le i \le n)$
C_i	t_i의 수행 시간
T_i	t_i의 주기
D_i	t_i의 마감시간
W_i	최악의 경우의 수행 시간
U	프로세서 이용률(U)
S	태스크 전환 시 발생하는 오버헤드

2.2.1 비율 단조 스케줄링 (RMS)

비율 단조 스케줄링(RMS, Rate Monotonic Scheduling) 은 최적의 정적 알고리듬으로 알려졌다. 주기를 기초로 태스크에 정적 우선순위를 할당하고 짧은 주기의 태스크에 높은 우선순위를 할당하며 태스크가 도착할 때 우선순위가 할당되므로 우선순위는 재 계산될 필요가 없다.

이것은 단일 프로세서 상에서 고정 우선순위 기반 선점형 스케줄링 방식으로 다음과 같은 가정을 하고 있다[3].

1. 태스크들은 주기적이고, 주기와 마감 시간은 같으며, 실행 중에 스스로 중지 되지 않는다.
2. 태스크들은 선점될 수 있고, 문맥 교환과 태스크 스케줄링에 드는 비용은 무시한다.
3. 모든 태스크들은 서로 독립이다. 즉, 선행 제약은 존재하지 않는다[4].

t_i의 최악의 경우의 수행 시간 (W_i)은 다음 [식1]과 같이 계산할 수 있다[3].

$$W_i(n+1) = C_i + \sum_{j<i} \left\lceil \frac{W_i(n)}{T_j} \right\rceil C_j, \quad W_i(0) = 0 \qquad [식1]$$

이 때, $W_i(n+1) = W_j(n)$ 이고 $W_i(n+1) \le T_i(n)$ 이면 t_i는 스케줄링 가능하고, 그렇지 않은 경우에는 t_i는 마감시간을 놓치게 된다.

2.2.2 마감 시간 우선 스케줄링 (EDF)

마감 시간 우선 스케줄링(EDF, Earliest Deadline Firs) 알고리듬은 동적 우선순위 스케줄링 방법으로 Liu와 Layland에 의해서 제시되었다.

EDF는 동적 알고리듬 중에서 선점 알고리듬으로는 최적인 것으로 알려져 있다.

태스크의 우선순위는 마감시간에 따라서 할당된다. 즉, 마감시간이 짧을수록 높은 우선순위가 할당되므로 임의의 순간에 실행되는 태스크는 실행이 완료되지 않은 태스크들 중에서 마감시간에 가까운 것이 선택된다. 또 정적 알고리듬과 달리 태스크의 우선순위가 시간에 따라 변하게 된다. Liu와 Layland는 EDF 알고리듬에 대해서도 스케줄링 가능성 기준을 제시하였다.

주기적으로 발생하는 독립적인 n개의 태스크에 대해, EDF 알고리듬의 스케줄링 가능판단에 대한 프로세서이용률(u)의 충분조건은 [식2]와 같다[3].

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \le 1 \qquad [식2]$$

이 식에서 보여주듯이 프로세서 이용률(u)이 최대 1(100%)까지 가능함을 알 수 있다.

따라서 태스크 전환시 발생하는 오버헤드(S)를 고려할 경우 태스크 집합의 프로세서 이용률(U)은 [식3]과 같다.

$$U = \sum_{i=1}^{n-1} \frac{C_i + 2S + I}{T_i} + \frac{C_j + 2S}{T_j}, \quad j = i+1 \qquad [식3]$$

주어진 태스크 집합에 대하여 RMS 알고리즘으로 스케줄링 가능하고 EDF 알고리즘보다 나은 성능을 이끌어내기 위한 조건은 다음 [식4]와 같다.

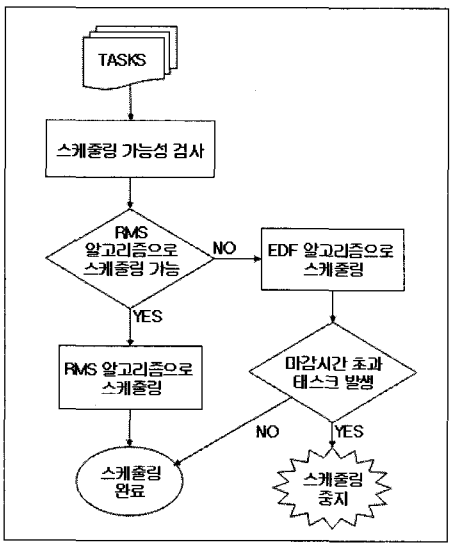
$$U = \sum_{i=1}^{n-1} \frac{C_i + 2S + I}{T_i} + \frac{C_j + 2S}{T_j} \le n(2^{1/n} - 1),$$
$$n = 1,2,\ldots, j = i+1 \qquad [식4]$$

또한, EDF 알고리듬은 동적 우선순위 알고리듬들 중에서 최적으로, EDF 알고리듬으로 스케줄링 될 수 없으면 다른 동적 우선순위 알고리듬으로도 스케줄링될 수 없다. 그러나 새로운 태스크를

할당할 때 마다 우선순위를 계산하기 때문에 실행시간 스케줄링 오버헤드가 큰 단점이 있다. 마감시간이 주기와 동일하거나 주기에 비례하여 작을 경우 RMS와 효율이 같고 마감시간이 주기보다 작을 때 최적이다.

2.3 선택 스케줄러 알고리듬 설계

본 논문에서 스케줄러 선택은 스케줄링 가능성 검사 알고리듬에서 얻어지는 프로세서 이용률을 기반으로 수행하며, 시스템에 영향을 미치지 않도록 그림과 같이 간결하게 설계하도록 제안하였다[6].



〔그림 2〕 스케줄링 관리자의 동작 흐름도

2.4 EDF 스케줄러 확장 알고리듬 설계

RMS 알고리듬으로 스케줄링이 가능한 충분조건은 $U = 0.693$ 으로서, 프로세서 이용률이 0.693 이하일 경우 RMS 알고리듬을 선택하고, 그 이상일 경우 별도로 최악의 수행 시간을 계산하지 않고 EDF 알고리듬을 제안하였다[6].

```
U = 태스크 집합의 프로세서 이용률 계산
if( U ≤ 0.693 )
    RMS로 스케줄링
else
    EDF로 스케줄링
```

RMS 알고리듬으로 안정적인 스케줄링을 보장할 수 없는, 프로세서 이용률이 0.693을 초과하는 상황에 대해서 EDF 알고리듬을 이용하여 스케줄링한다.

하지만, EDF 알고리듬으로도 프로세서 이용률이 100%를 넘어갈 경우 안정적인 스케줄링을 보장할 수 없다. 따라서 EDF 알고리듬으로 스케줄링 할 때, 각각의 태스크가 실행에 들어가기 전에 마감시간이 초과되는 지를 체크하여 마감시간을 초과하는 상황이 예상될 경우 스케줄링을 중지한다.

III. 시뮬레이션 및 결과고찰

실시간 리눅스 상에서 실시간 태스크들의 스케줄링 가능성을 검사하여 특성에 맞는 스케줄러를 선택하는 스케줄러 관리자의 스케줄링 결과를 기존의 RMS와 EDF 알고리듬과 비교 평가한다.

선택적 알고리듬의 성능분석을 시뮬레이션으로 위한 운영체제로 RT Linux 3.1과 Linux 커널 버전 2.4.32를 사용하였으며, 스케줄러는 실시간 리눅스에서 기본적으로 제공하는 RMS 스케줄러와, Practicia Balbastre와 Ismael Ripoll에 의해 구현된 EDF 스케줄러를 인스톨함으로서 프로세서 이용률에 따른 평균응답시간의 성능을 분석하였다.

3.1 프로세스 이용률에 따른 성능분석

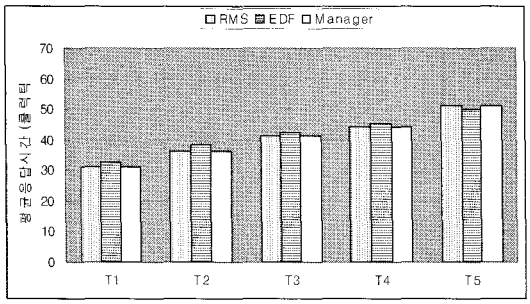
스케줄러는 실시간 리눅스에서 제공하는 RMS 스케줄러와, EDF 스케줄러를 인스톨하여 사용한다.

시뮬레이션을 위한 마감시간과 주기가 같은 경우와 마감시간과 주기가 다른 경우로 나누어 각각 다양한 프로세서 이용률을 가지는 태스크 집합을 준비하고, 성능분석을 위한 평균 응답 시간과 마감시간을 비교한다.

3.1.1 프로세스 이용률:0.626(주기=마감시간)

[실험 1]에서는 기존의 RMS와 EDF 알고리듬으로 스케줄링을 한 경우의 평균 응답시간을 구하고 스케줄러 선택 알고리듬이 스케줄러를 선택하여 스케줄링을 한 경우를 비교하였다. RMS 알고리듬의 성능이 EDF 알고리듬 보다 약간 좋음을 알 수 있다.

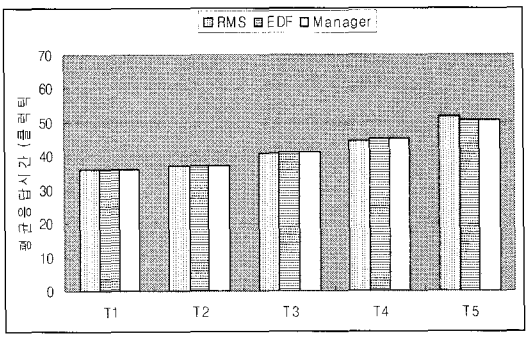
RMS의 결과와 매니저(Manager)의 결과가 동일함을 알 수 있다. 이는 스케줄링 가능성 검사에서 RMS 알고리듬의 충분조건인 $U \le n(2^{\frac{1}{n}}-1)$이 넘지 않기 때문에, RMS 알고리듬을 선택하여 스케줄링 되었다고 볼 수 있다.



〔그림 3〕 프로세스이용률 0.626(주기=마감시간)

3.1.2 프로세스 이용률:0.756(주기=마감시간)

[실험 2]에서는 태스크 집합의 프로세서 이용률이 RMS 알고리듬의 충분조건인 0.693을 넘었기 때문에, RMS로 스케줄링 할 경우 태스크들이 마감시간 내에 수행을 완료하는 것을 보장하지 못한다.



〔그림 4〕 프로세스이용률 0.756(주기=마감시간)

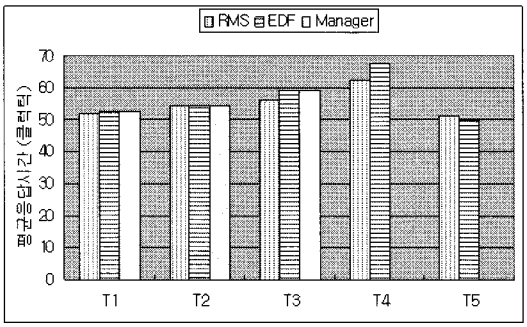
그러나 RMS 스케줄링 경우에도 태스크 집합에서 마감시간을 넘겨 스케줄링 된 태스크는 없었다.

스케줄러 선택 알고리듬은 EDF 알고리듬을 선택하여 스케줄링 하였다. EDF 알고리듬의 경우 스케줄링 가능한 구간이 $0 \le U \le 1$이기 때문에 충분히 스케줄링 안정성을 보장하며, RMS 알고리듬에 비하여 크게 성능이 떨어지거나 하는 일은 없었다.

3.1.3 프로세스 이용률:1.101(주기=마감시간)

[실험 3]에서는 태스크 집합에 대한 프로세서 이용률이 100%를 초과하고 있다. 이 경우 RMS와 EDF 알고리듬 모두 태스크들의 마감시간 내 스케줄링을 보장할 수 없다. RMS 알고리듬과 EDF 알고리듬 모두 태스크 4와 5에서 마감시간 위반이 발생한다. 스케줄러 선택 알고리듬은 마감시간을 위반하는 태스크가 발생할 경우 시스템의 안정성 확보를 위해 스케줄링을 중지한다.

이를 위해 EDF 스케줄러를 확장하였기 때문에 마감시간을 위반하게 되는 태스크 4 이전의 태스크의 경우 EDF 알고리듬과 동일한 성능을 보여주게 된다.

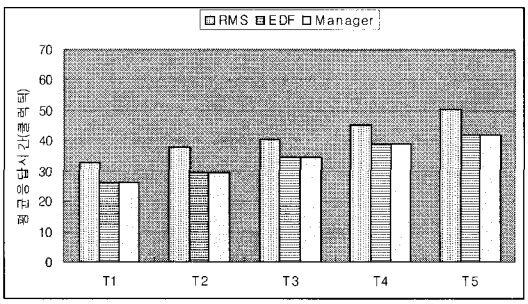


〔그림 5〕 프로세스이용률 1.101(주기=마감시간)

3.1.4 프로세스이용률:0.626(주기≠마감시간)

[실험 4]에서는 주기와 마감시간이 다르므로 [그림 6]에서의 결과와 같이 EDF 알고리듬이 RMS 알고리듬 보다 더 좋은 평균 응답시간을 보였다.

따라서 스케줄러 선택 알고리듬에서는 주기와 마감시간이 다른 태스크 집합에 대해서는 EDF 알고리듬으로 스케줄링을 할 수 있으므로 평균 응답 시간도 EDF 알고리듬과 동일하게 나타났다.

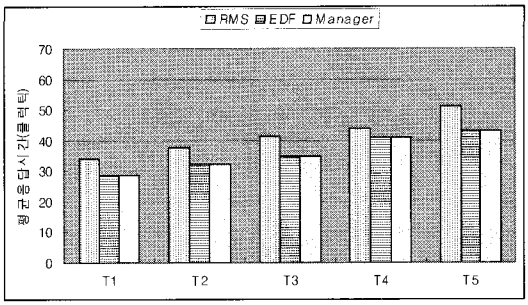


〔그림 6〕 프로세스이용률 0.626(주기≠마감시간)

3.1.5 프로세스이용률:0.756(주기≠마감시간)

[실험 5]는 태스크 집합의 프로세서 이용률이 실험 2에서처럼 0.760인 경우 RMS 알고리듬으로는 태스크들의 마감시간 내에 수행을 보장하지 못한다.

하지만 여기서도 [실험 2]에서와 같이 RMS 알고리듬으로 스케줄링 하더라도 마감시간을 위반하는 태스크는 없었다. [실험 5]에서 스케줄러 선택 알고리듬은 태스크 집합의 프로세서 이용률이 RMS 알고리듬의 충분조건인 0.693을 넘었고, 주기와 마감시간이 다르게 설정되었으므로 [그림 7]과 같이 EDF 알고리듬을 선택하여 스케줄링할 수 있다.



〔그림7〕프로세스이용률0.756(주기≠마감시간)

3.2 스케줄러 선택 알고리듬의 성능고찰

RMS 알고리듬은 마감시간이 주기와 같은 태스크 모델에서 최적이지만, 마감시간이 주기보다 작은 경우에는 최적이지 못하고, 프로세서 이용률도 평균 88%라는 제약이 존재한다.

EDF 알고리듬은 마감시간이 주기와 동일한 경우 RMS 알고리듬의 효율과 같고, 마감시간이 주기보다 작은 경우 최적이며, 프로세서 이용률을 거의 100%까지 이용할 수 있다.

하지만, 동적 우선순위 스케줄링 방법인 EDF 알고리듬은 스케줄링 오버헤드가 존재하기 때문에 마감시간이 주기와 동일할 경우 정적 우선순위 스케줄링 방법인 RMS 알고리듬보다 효율이 떨어질 수 밖에 없다. 따라서 마감시간과 주기가 같지 않은 경우와 스케줄링 오버헤드를 고려하여 스케줄링 가능성 검사 알고리듬을 구현하였다.

이를 이용하여 RMS 알고리듬으로 스케줄링이 가능한 경우 RMS 알고리듬으로 스케줄링을 하고, 그렇지 못할 경우 EDF 알고리듬으로 스케줄

링을 하도록 선택하는 방법으로 어떠한 특성의 태스크 집합이라도 마감시간을 보장하여 스케줄이 가능하다. [표 2]는 RMS과 EDF 알고리듬의 비교와 본 논문에서 제안하는 스케줄링 관리자의 성능분석 결과이다.

〔표 2〕 프로세스 이용률에 따른 스케줄링 관리자의 성능분석 결과

	이용률	주기: 마감시간	RMS	EDF	관리자가 선택한 알고리듬
실험1	0.626	=	○	△	RMS
실험2	0.756	=	○	△	EDF
실험3	1.101	=	X	X	-
실험4	0.626	≠	△	○	EDF
실험5	0.756	≠	△	○	EDF
○ 스케줄링 가능, 최적의 성능 △ 스케줄링 가능 X 안정적인 스케줄링 불가 - 스케줄링 중					

IV. 결론

본 논문에서는 실시간 태스크들을 스케줄링 하기 위해서 스케줄링 관리자를 설계하였으며, 마감시간 내에 태스크의 수행을 보장하기 위해 EDF 스케줄러를 확장하여 시스템 정지와 같은 위험한 상황을 미리 예측 가능하도록 구현하였다.

또한, 실시간 리눅스가 보다 동적인 상황에 적응이 가능하도록, RMS 알고리듬과 EDF 알고리듬을 태스크 집합의 특성에 따라 유연하게 선택함으로써 안정적인 스케줄링과 더 나은 성능을 이끌어 낼 수 있도록 선택 알고리듬을 제안하였다.

제안한 선택 알고리듬의 결과로 실행시간 오버헤드를 가지는 동적 스케줄링 알고리듬인 EDF 알고리듬보다 정적 스케줄링 알고리듬인 RMS 알고리듬을 선호하도록 하였고, RMS 알고리듬으로 스케줄링이 불가능한 경우 EDF 알고리듬을 선택하도록 하였다.

스케줄링 오버헤드와 마감시간이 항상 주기의 끝이 아닌 경우를 고려한 스케줄링 가능성 검사 방법에 의한 성능분석을 위해 컴퓨터 시뮬레이션을 통한 결과를 분석결과 본 논문에서 제안한 스케줄링을 선택하는 방법은 어떠한 특성의 태스크 집합이라도 마감시간을 보장하는 스케줄이 가능하게 되었다.

향후 스케줄링 가능성 검사 알고리듬은 스케줄러 선택 알고리듬의 오버헤드는 무시할 수 있는 수준이었지만, 오버헤드를 줄일 수 있는 최적화 기법의 연구가 계속되어야 할 것이다.

참 고 문 헌

[1] Fredette A. and Cleaveland R., "A Generalized to Real-Time Schedulability Analysis.", 10th workshop on real-time operating system and software, 1993.

[2] John L., Sha L., Strosnider K. and Tokuda H., "Fixed Priority Scheduling Theory for Hard Real-Time Systems.", Foundations of real-time computing : Scheduling and Resource Management, pp.1-30., Mar., 1990.

[3] Warren C., "Rate Monotonic Scheduling.", IEEE Micro, pp.34-38, Jun., 1991.

[4] Liu J. W. S., "Real-Time Systems", Prentice Hall, 2000.

[5] Kevin M., "Preemptible Linux-a reality check.", Articles and whitepapers about Linux in embedded applications of LinuxDevice.com, Sep., 2001.

[6] Daniel P. and Marco C., "Understanding the Linux Kernel.", O'Reilly, 2001.

[7] 심형용, 김지환, 선동국, 김성조,"RTOS를 위한 TCP/IP 프로토콜 스택의 구현," 한국정보과학회 추계 학술발표논문집, 제29권제2호, pp427-429, 한국정보과학회, 2002.10

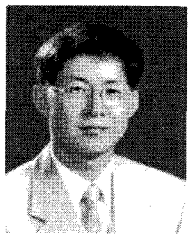
[8] 김방현, 류성준, 김종현, 남영광, 이광용 "실행시간 추정가능한 RTOS 시뮬레이터의 구

현",한국시뮬레이션학회 춘계학술대회 논문집, pp125-129, 2002.5

[9] 김방현, 김태규, 김종현, "리눅스 프리웨어를 이용한 SIOS의 구현",한국정보과학회 병렬처리시스템연구회 학술발표회 논문집 제15권 제1호, pp173-182, 2004.5

[10] 정종칠, "실시간 리눅스에서 효과적인 태스크 스케줄링을 위한 스케즐링 관리자의 설계 및 구현,"한신대학교대학원 석사학위논문, 2006.8

◐ 저 자 소 개 ◑



강민구(Min-goo, Kang)
1986년 연세대학교 전자공학과(공학사)
1989년 연세대학교 대학원 전자공학과(공학석사)
1994년 연세대학교 대학원 전자공학과(공학박사)
2000~현재 한신대학교 정보통신학과 교수
관심분야 : 정보통신시스템 etc.
E-mail : kangmg@hs.ac.kr