

실시간 리눅스에서 선택 알고리즘을 이용한 스케줄링 성능평가

최정훈^o, 김경화, 김두상, 최대수, 임종규, 박한규, 구용완
 수원대학교 컴퓨터학과
 e-mail : rtlinux@naver.com

A Simulation of Scheduling using Selection Algorithm on Real-Time Linux.

Jung-Hoon Choi^o, Kyung-Hwa Kim, Doo-Sang Kim,
 Dae-soo Choi, Jong-Kyu Im, Han-Gue Park, Yong-Wan Koo
 Dept. of Computer, The University of Suwon

요약

리눅스는 POSIX.1b 표준의 일부 지원으로 연성 실시간 시스템을 지원한다. 현재 경성 실시간 시스템을 지원하기 위해 리눅스를 확장한 여러 운영체제가 개발된 바 있지만, 다양한 프로그래밍 인터페이스 및 스케줄러의 지원이 부족하다. 현재까지 실시간 리눅스를 위한 스케줄러는 RMS와 EDF 두가지의 스케줄러가 별도로 구현되어 있다. 이 두가지 스케줄러 중에서 사용자가 각각의 스케줄링 알고리즘의 특성을 고려하지 않고 두가지 방법을 선택하여 사용하고 있다. 본 논문에서는 이러한 단점들을 해결하기 위한 안정적인 스케줄링 가능성 검사를 통하여 RMS와 EDF 스케줄링 방법의 특성에 맞게 스케줄러 중에서 하나를 선택하여 사용함으로써 종료시한을 보장하고 또한 스케줄이 불가능한 경우 태스크 스케줄로 인해 발생하는 시스템 정지 현상을 제거하고 태스크 집합의 효율적인 관리를 위한 스케줄링 가능성 검사 알고리즘과 스케줄러 선택 알고리즘을 제시하여 스케줄링 알고리즘의 성능평가 결과를 제시한다.

1. 서론

기존 시분할 시스템은 시스템의 설계시 시스템의 전체 성능 향상과 빠른 평균 응답시간, 자원의 공정한 분배를 목적으로 하고 있지만 실시간 시스템에서는 태스크가 종료시한을 만족하여 수행할 수 있는지 여부가 가장 중요한 설계의 관건이 되고 있다. 시분할 시스템과는 달리 실시간 시스템에서는 시스템의 빠른 응답시간보다는 시간의 예측가능성을 높인, 즉 최악 수행시간이 어느 범위 이상을 넘어서지 않는다는 것을 보장하는데 더 관심을 가져야 한다. 또 자원의 공정한 분배보다는 자원의 안정된 분배를 더 중요하게 여긴다[1].

실시간 리눅스에서는 현재 실시간 태스크의 개발을 위한 인터페이스를 제공해주고 있지만, 아직 다양한 스케줄러의 지원이 부족하다[2]. 그러므로 본 논문에서는 실시간 리눅스에서 기본적으로 제공하는 비율 단조형 스케줄링과 종료시한 우선 스케줄링 알고리즘 기법을 사용하여 태스크 집합이 제출되었을 경우 각 태스크 집합에 적합한 스케줄러를 선택하도록 스케줄링 가능성 분석후 적합한 알고리즘을 시스템이 선정하도록 하며 안정된 스케줄링을 할 수 있도록 선택 알고리즘 기법을 제시한다.

본 논문의 구성을 보면 2장에서는 관련연구로 스케줄링 가능성(schedulability), 비율 단조형 스케줄링과 종료시한 우선 스케줄링 알고리즘을 살펴본다. 3장에서는 본 논문에서 제시한 선택 알고리즘의 실제 구현을 보여주고, 4장에서는 제시된 선택 알고리즘의 성능평가 매개변수에 따른 결과를 기술하고, 5장에서는 결론 및 향후 연구과제를 제시한다.

2. 관련연구

2.1 스케줄링 가능성(Schedulability)

스케줄링 가능성 검사[2]를 예측하기 위해서는 간단한 수식을 통하여 판단 가능한 분석적 방법(analytical method)을 많이 사용한다. 분석적 방법을 통한 스케줄링 가능성 검사는 우선순위 구동 방식의 알고리즘들에 유용하게 사용할 수 있는 기법이다. 시스템 설계자는 설계 당시 시스템에 적합한 스케줄링 기법을 선택하고 태스크 주기, 실행시간, 종료시한 등을 결정할 때, 분석적 방법을 이용하면 쉽게 시스템의 스케줄링 가능성 여부를 판단할 수 있다. 이러한 이유로 단일 프로세서 시스템과 통계적으로 설정되는 다중 프로세서 시스템에서 우선순위 구동 알고리즘이 많이 사용되고 있다.

2.2 비율 단조(Rate-Monotonic) 스케줄링 알고리즘

Rate-Monotonic Scheduling(이하 RMS)은 실시간 스케줄링에 관한 연구 중 가장 많이 연구되고 사용되는 방법 중의 하나이다. 본 알고리즘은 최적의 정적 알고리즘으로 알려져 있으며, 주기를 기초로 태스크에 정적 우선순위를 할당하고 짧은

주기의 태스크에 높은 우선순위를 할당하며 태스크가 도착할 때 우선순위가 할당되므로 우선순위는 재계산될 필요가 없다[3]. 이것은 단일 프로세서 상에서 고정 우선순위 기반 선점형 스케줄링(preemptive scheduling based on fixed priority)방식으로 다음과 같은 가정을 하고 있다.

- ① 태스크들은 주기적이고, 주기와 종료시한은 같으며, 실행 중에 스스로 중지(suspend)되지 않는다.
- ② 태스크들은 선점될 수 있고, 문맥교환(context switching)과 태스크 스케줄링에 드는 비용은 무시한다.
- ③ 모든 태스크들은 서로 독립이다. 즉, 선행 제약(precedence constraints)은 존재하지 않는다.

RMS는 고정 우선순위 방법으로 태스크의 우선순위는 자신의 주기에 반비례한다. 따라서, 작업의 주기가 주어지면 자신의 우선순위는 정적(static)으로 정해진다. 만일 태스크 τ_i 의 주기가 τ_j 보다 작다면, τ_i 의 우선순위는 τ_j 보다 높다. 높은 우선순위 태스크는 낮은 우선순위 태스크를 항상 선점할 수 있다. 따라서 태스크 집합 $\tau_1, \tau_2, \dots, \tau_n$ 에 대해서, τ_1 의 우선순위가 가장 높고, τ_n 의 우선순위가 가장 낮다고 정의할 수 있다.

태스크가 주기적이기 때문에, 매 주기마다 한번씩 수행된다. 태스크의 매 수행을 작업(job)이라 하자. τ_i 의 j번째 작업은 $I_i + (j-1)T_i$ 에 실행 준비될 것이고, C_i만큼 실행된 후, $I_i + (j-1)T_i + D_i$ 안에 실행을 완료할 것이다. 태스크 집합이 주어졌을 때, 모든 태스크들이 그들의 종료시한을 모두 만족하면, 그 태스크 집합은 "스케줄링 가능하다(schedulable)"라고 하며, RMS에서 스케줄링 가능성 필요충분조건은 표1과 같다.

<표 1> RMS에서 스케줄링 가능성 필요충분조건

No. of Task	프로세서 이용률
1	1.00
2	0.828
3	0.780
4	0.757
5	0.743
↓	↓
∞	0.693

2.3 종료시한 우선(Earliest-Deadline First) 스케줄링 알고리즘
 RMS 알고리즘과는 달리 Earliest-Deadline First(이하 EDF) 알고리즘은 동적 우선순위 스케줄링 방법으로 Liu와 Layland에 의해서 제시되었다[6]. EDF는 동적 알고리즘으로 선형 알고리즘들 중에서 최적적인 것으로 알려져 있다. 태스크의 우선순위는 종료시한에 따라서 할당된다. 즉, 종료시한이 짧을수록 높은 우선순위가 할당되므로 임의의 순간에 실행되는 태스크는 실행이 완료되지 않은 태스크들 중에서 종료시한에 가까운 것이 선택된다. 또 정적 알고리즘과 달리 태스크의 우선순위가 시간에 따라 변하게 된다. EDF 알고리즘에서는 다음과 같은 조건이 만족되면 주기적으로 발생하는 독립적인 n개의 태스크들은 스케줄 가능하다고 할 수 있다. Liu와 Layland는 EDF 알고리즘에 대해서도 스케줄링 가능성 기준을 제시하였다. $D_i = T_i$ 인 주기적인 태스크들에 대해서, EDF 알고리즘에 의해서 스케줄링 가능성의 필요충분조건은 다음과 같다.

$$U = \sum U_i = \sum \frac{C_i}{T_i} \leq 1$$

또한 EDF 알고리즘은 동적 우선순위 알고리즘들 중에서 최적이다. 즉, EDF 알고리즘으로 스케줄링 될 수 없으면 다른 동적 우선순위 알고리즘으로도 스케줄링 될 수 없다.

3. 실시간 리눅스의 선택 알고리즘과 스케줄링 가능성 검사 알고리즘

3.1 선택 알고리즘의 구현

실시간 운영체제에서 태스크의 스케줄링은 매우 중요한 역할을 수행한다. 실시간 시스템에서는 각각의 태스크들마다 종료시한이 주어져 있고 이 태스크들이 종료시한 내에 수행되지 못하는 경우에는 큰 피해를 입을 수 있으므로 반드시 지켜야 한다. 지금까지 종료시한을 보장하기 위한 여러 스케줄링 방법들이 연구되어 왔으며[4][5] 본 논문에서는 이런 방법 중 우선순위가 고정적으로 주어진 주기 태스크 집합들에 대한 정적 스케줄링 알고리즘들 중 최적 알고리즘이라 알려진 RMS와 태스크의 수행중에 종료시한에 가까울수록 우선순위를 높게해주는 동적 우선순위 할당 방법을 사용하는 EDF에 대해서 다루었다. 또한 기존의 RMS나 EDF에서 실시간 태스크들의 집합이 종료시한을 보장받지 못하는 것을 검사하는 방식을 확장하여 좀 더 정확한 스케줄링 검사방법을 제시하며, 실시간 리눅스 운영체제 상에서 확장된 스케줄링 가능성 검사를 사용하여 RMS와 EDF 중에서 하나를 선택하여 스케줄링하는 선택 알고리즘을 제시하고 스케줄링 알고리즘의 성능 평가 결과를 제시한다.

3.2 스케줄링 가능성 검사 알고리즘의 구현

현재까지 실시간 리눅스를 위한 스케줄러는 RMS와 EDF 두가지의 스케줄러가 별도로 구현되어 있다. 이 두 가지 스케줄러 중에서 사용자가 각각의 스케줄링 알고리즘의 특성을 고려하지 않고 두 가지 방법을 사용자 수준에서 선택하여 사용하고 있다[7]. 이로 인해 종료시한 miss rate를 증가시키는 결과를 초래한다. 또한 현재 실시간 리눅스에서는 스케줄 불가능 태스크에 대한 스케줄로 인하여 시스템이 정지되는 현상이 발생된다. 이러한 현상은 경쟁 실시간 시스템에서는 매우 치명적이다[8].

그러므로 본 논문에서는 이러한 단점들을 해결하기 위한 안정적인 스케줄링 가능성 검사를 통하여 RMS와 EDF 스케줄링 방법의 특성에 맞게 적절한 스케줄러를 사용함으로써 종료시한을 보장하고 또한 스케줄이 불가능한 경우 태스크 스케줄로 인해 발생하는 시스템 정지 현상을 제거하였다. 위에서 제시한 방법들의 알고리즘은 다음과 같다.

```

<알고리즘 1> 스케줄링 가능성 검사 알고리즘
double Schedulable_check(NTASKS)
{
    /* 태스크 집합에 대해 주기와 종료시한을 검사하여 동일할까를 판단 */
    for(i=0; i<NTASKS; i++) {
        if(Parameters[i].period == Parameters[i].deadline) looping;
        else { RMS = FALSE; break; }
    }
    /* 태스크 집합에 대한 스케줄 가능성 검사 */
    if(RMS) { /* RMS인 경우 스케줄링 가능성 검사 */
        for(i=0; i<NTASKS; i++)
            Util += Parameters[i].compute/Parameters[i].period;
    } else { /* EDF인 경우 스케줄링 가능성 검사 */
        for(i=0; i<NTASKS; i++)
            Util += ((Parameters[i].compute)*(Parameters[i].period -
                Parameters[i].deadline)) / Parameters[i].period;
    }
}
return (Util);
    
```

<알고리즘 2> 스케줄 가능성 결과에 따른 스케줄러 선택 알고리즘

```

Scheduler_Select()
{
    Util = Schedulable_check(NTASKS); /* 태스크 집합에 대한 스케줄 가능성 검사 */
    if (Util >= 0 && Util <= NTASKS * (POM2 / (1/NTASKS)) - 1) )
        SCHED_OK = TRUE; /* RMS, EDF 스케줄 가능 */
    else if (Util >= 0 && Util <= 1) SCHED_OK = TRUE; /* EDF 스케줄 가능 */
    else { SCHED_OK = FALSE; deadmodule(); }
    /* 스케줄 불가능, 태스크를 큐에서 제거하여 리눅스 커널 모드로 전환하여 시스템 정지 현상 예방 */

    if (SCHED_OK && RMS) /* RMS로 태스크 스케줄링 */
        for (x=0; x<NTASKS; x++) /* 태스크 구조체 초기화 */
            man_rt_task_init(&tasks[x]), man_fun.Parameters[x].compute = 300, x+1);
        for (x=0; x<NTASKS; x++) /* 태스크에 주기를 부여하여 태스크 생성 */
            man_rt_task.make_periodic(&tasks[x]), now+(RTIME)100,
                (RTIME) Parameters[x].period);
    } else { /* EDF로 태스크 스케줄링 */
        for (x=0; x<NTASKS; x++)
            edf_rt_task_init(&tasks[x]), edf_fun.Parameters[x].compute = 300, x+1, x+1);
        for (x=0; x<NTASKS; x++)
            edf_rt_task.make_periodic(&tasks[x]), now+(RTIME)100,
                (RTIME)Parameters[x].period, (RTIME)Parameters[x].deadline);
    }
    return 0;
}
    
```

4. 성능평가

성능평가에 사용된 시스템 환경은 IBM PC Pentium급에서 리눅스 커널 2.0.29버전을 기반으로 하였으며, 실시간 리눅스에서 제공하는 비윤단조형 스케줄러와 Ismael Ripoll에 의해 구현된 종료시한 우선순위 스케줄러를 인스톨하였다. 성능평가에 사용된 태스크 집합에 대한 매개변수는 <표 3>과 <표 4>에 제시되었다.

<표 3> 성능평가 매개변수 (주기=종료시한)

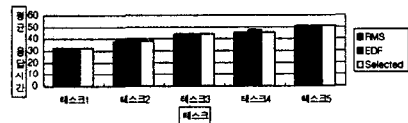
태스크	1	2	3	4	5	이름
기간	20	30	40	50	60	0.719
비율	200	300	400	500	700	
비율	200	300	400	500	700	0.760
비율	200	300	400	500	700	0.837
비율	200	300	400	500	700	1.040
비율	200	400	300	600	700	

<표 4> 성능평가 매개변수 (주기≠종료시한)

태스크	1	2	3	4	5	이름
기간	20	30	40	50	60	0.719
비율	200	300	400	500	700	
비율	200	300	400	500	700	0.760
비율	200	300	400	500	700	0.837
비율	200	400	300	600	700	

4.1 종료시한과 주기가 동일한 경우

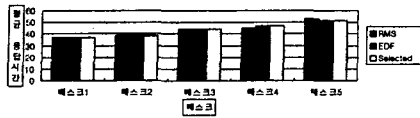
본 절에서는 RMS와 EDF에 대한 상대적인 평균 응답시간을 비교하기 위하여 EDF인 경우 각 작업의 종료시한을 그 작업의 주기와 동일하게 설정하여 실험을 하였다. 실시간 리눅스에서 제공하는 RMS는 실시간 태스크들이 먼저 처리한 후 기존의 리눅스 커널을 낮은 우선순위 실시간 태스크로 취급한다. 즉, 리눅스는 단지 실시간 시스템이 아무 것도 처리하지 않을 때 수행된다. 따라서, 실시간 태스크 폭주로 리눅스 커널 모드로 되돌아 갈 수 없는 경우가 발생할 수 있어 시스템이 불안정하다. (그림 3) 이용률이 0.719인 경우의 평균 응답시간(단위: 클럭)



(그림 3)은 일련의 태스크 집합에 대해 프로세서 이용률이 0.719인 경우 평균 응답시간을 보여주고 있다. 여기서, 먼저 기존의 RMS와 EDF로 스케줄링을 한 경우의 평균 응답시간을 구하고 제안한 선택 알고리즘 스케줄러를 선택하여 스케줄링을 한 경우를 비교하였다. EDF인 경우 평균 응답시간을 RMS와 비교하기 위해서는 EDF의 태스크 종료시한과 주기를 동일한 경우로 설정해야 한다. <표 3>에서 설명한 것처럼 EDF의 경우 주기와 종료시한이 동일한 경우 RMS의 특성을 따르기 때문에 평균 응답시간이 거의 동일하게 계산되었다. (그림 3)에서 제안한 기법의 평균 응답시간은 RMS의 평균 응답시간과 동일함을 알 수 있다. 이는 제안한 선택 알고리즘 기법이 먼저 태스크 집합에 대해 스케줄링 가능성 검사를 행함으로써, 즉 $n(2^{\frac{1}{n}} - 1)$ 을 계산하여 스케줄 가능한 필요 충분조건인 0.743을 넘지 않으므로 제안한 선택 알고리즘은 RMS를 선택하여 스케줄링이 되었음을 볼 수 있

다.

(그림 4) 이용률 0.760인 경우의 평균 응답시간(단위: 클럭틱)



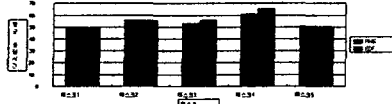
하지만, (그림 4)에서는 태스크 집합의 프로세서 이용률 0.760으로 스케줄링 가능한 필요 충분조건인 0.743을 넘었기 때문에 RMS로 스케줄링할 경우 태스크들의 종료시한내에 수행을 보장하지 못한다. 따라서, 제시한 선택 알고리즘에서는 EDF기법을 이용하여 스케줄링되었다. EDF의 경우 스케줄링 가능한 구간이 $0 \leq i \leq 1$ 이기 때문에 충분히 스케줄링 안정성을 보장한다. RMS인 경우 다행히 주어진 태스크 집합에서 종료시한을 넘겨 스케줄링된 것은 없었다.

(그림 5) 이용률이 0.937인 경우의 평균 응답시간(단위: 클럭틱)



(그림 5)는 태스크 집합에 대한 프로세서 이용률이 거의 100%에 근접하고 있다. 이러한 경우 제안한 선택 알고리즘 기법을 통해 스케줄링하게 되면 RMS 대신에 EDF를 선택한다. RMS로 스케줄링 할 경우 종료시한을 초과하는 경우가 발생할 수 있는데, 실제 RMS로 한 경우 종료시한을 위반한 태스크가 1개 발생하였음을 확인할 수 있었다. 그러나 EDF에서는 1개의 태스크도 종료시한을 위반함 없이 스케줄링 되었다. 역시 스케줄링 가능한 구간이 $0 \leq i \leq 1$ 에 포함되므로 안정된 스케줄링이 되었다. 한가지 고려할 사항은 평균 응답 시간에서 RMS가 EDF보다 빠르게 나타났지만 종료시한을 위반한 태스크가 발생했으므로 별 의미가 없다 하겠다.

(그림 6) 이용률이 1.046인 경우의 평균 응답시간(단위: 클럭틱)

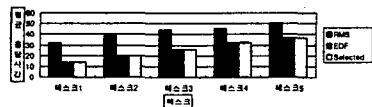


(그림 6)은 태스크 집합에 대한 프로세서 이용률이 100%를 초과하고 있다. 이 경우 RMS와 EDF 모두 태스크들의 종료시한내 스케줄링을 보장할 수 없다. 본 연구에서 제안한 선택 알고리즘 기법은 프로세서 이용률이 100%를 초과하면 시스템의 안정성 확보를 위해 실시간 스케줄링을 하지 않고 리눅스 커널 모드로 되돌아간다. 따라서, (그림 6)은 제안한 선택 알고리즘을 이용하였을 경우 스케줄링에 대한 평균 응답 시간은 나타나지 않았다.

4.2 종료시한과 주기가 다른 경우

본 절에서는 종료시한이 주기 이전에 주어진 EDF에 대한 평균응답시간을 비교하기 위하여 각 태스크의 종료시한을 달리하여 실험하였다. 평균응답시간에 있어 RMS와의 상대평가를 위하여 EDF의 종료시한을 설정할 때, 앞 절의 실험에서 사용했던 태스크 집합의 프로세서 이용률에 기준을 두어 설정하였다.

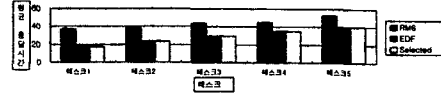
(그림 7) 이용률이 0.719인 경우의 평균응답시간(단위: 클럭틱)



(그림 7)은 태스크 집합 프로세서 이용률이 (그림 3)에서처럼 0.719인 경우이다. 이 경우 태스크 집합에 종료시한이 주기 이전에 설정된 경우 RMS로는 스케줄링 될 수 없고 EDF로 스케줄링이 되어야 한다. 본 성능평가 결과 EDF인 경우가 RMS보다 높은 평균응답시간을 보였다. 따라서, 선택 알고리즘 기법을 사용하는 스케줄러는 주기와 종료시한이 다른 태스크 집합에 대해서는 EDF로 스케줄링을 할 수 있으므로

평균응답시간도 EDF와 동일하게 나왔다.

(그림 8) 이용률이 0.760인 경우의 평균 응답시간(단위: 클럭틱)



(그림 8)은 태스크 집합 프로세서 이용률이 (그림 4)에서처럼 0.760인 경우이다. 이는 RMS인 경우 스케줄링의 필요충분조건을 만족하지 못하는 경우이다. 이 경우 역시 주기와 종료시한이 달리 설정되었으므로 선택 알고리즘 기법을 사용하여 스케줄링시 EDF로 스케줄링이 되며, 평균응답시간에서도 (그림 7)과 같이 월등한 평균 응답시간을 보였다.

5. 결론 및 향후 연구과제

본 논문에서는 안정적인 스케줄링 가능성 검사를 통하여 RMS와 EDF 스케줄링 방법중에서 시스템이 하나의 스케줄러를 선택하여 스케줄링 함으로서 종료시한을 보장하며, 스케줄링 가능성 검사 알고리즘과 스케줄러 선택 알고리즘을 제시하여 스케줄링 알고리즘의 성능평가 결과를 제시하였다.

향후 연구과제로는 문맥 교환 오버헤드가 많은 EDF 보다는 RMS를 선호하려고 노력하여야 하겠으며, 비주기적인 태스크를 실시간 태스크의 수행후 여분의 슬랙(slack)에서 실행할 수 있도록 기존의 여러 비주기적 실시간 스케줄링 알고리즘을 실시간 리눅스에 포함하여 본 논문에서 구현한 스케줄링 가능성 검사 방법을 보다 확장하여 스케줄러를 선택가능 하도록 한다면 향상된 실시간 리눅스가 될 것이다.

참고 문헌

- [1] 안병철, "PC환경 실시간 운영체제를 위한 멀티태스킹 커널의 설계 및 구현", 영남대학교, 1996.
- [2] An Fredette and Rcleaveland, "A Generalized to Real-Time Schedulability Analysis", 10th workshop on real-time operating system and software, 1993.
- [3] John Lehoczky, Lui Sha and Ye Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior", Proc. IEEE Real-Time Systems Symposium, pp.166-171, Dec. 1989.
- [4] Lehoczky, J.P., Sha, L., Strosnider, J.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.
- [5] Liu, C. L. and Layland, J. W., "Scheduling algorithm for multiprogramming in a hard real-time environment", JACM, Vol.20, pp.46-61, 1973.
- [6] Chetto. H and Chetto. M, "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, vol.15, no.10, pp.1261-1269, Oct. 1989.
- [7] 박정훈, "스케줄링 정책의 다양성을 지원하는 실시간 커널의 설계 및 구현", 서울대학교, 1997.
- [8] Barabanov, M., "A Linux-based Real-Time Operating System", <http://rtlinux.cs.nmt.edu/~baraban/thesis/thesis.htm>, Jun. 1997.