韓國 OA學會 論文誌
第7卷 第2號, 2002. 6.

2002-7-2-1-1

실시간 Linux 환경에서 효율적인 스케쥴링을 위한 선택 알고리즘의 구현

김 성 락*

An Implementation of Selection Algorithm for Efficient Scheduling on Real-Time Linux Environment

Seong-Rak Kim*

요 약

현재까지 실시간 Linux를 위한 스케쥴러는 RMS와 EDF 두 가지의 스케쥴러가 별도로 구현되어 있다. 이 두 가지 스케쥴러 중에서 사용자가 각각의 스케쥴링 알고리즘의 특성을 고려하지 않고 두 가지 방법을 선택하여 사용하고 있다. 이로 인해 실시간 시스템의 스케쥴링 가능성 검사의 미수행으로 종료시한 miss rate를 증가시키는 결과를 초래한다. 또한, 현재 실시간 Linux에서는 스케쥴 불가능한 태스크를 스케쥴함으로서 시스템이 정지되는 현상이 발생된다. 이러한 현상은 경성 실시간 시스템에서는 매우 치명적이다. 그러므로 본 논문에서는 이러한 단점들을 해결하기 위한 안정적인 스케쥴링 가능성 검사를 통하여 RMS와 EDF 스케쥴링 방법의 특성에 맞게 적절한 스케쥴러를 사용함으로서 종료시한을 보장하고 또한 스케쥴이 불가능한 경우 태스크 스케쥴로 인해 발생되는 시스템 정지 현상을 제거하였다. 이를 위해서 본 논문에서는 태스크 집합의 효율적인 관리를 위한 스케쥴링 가능성 검사 알고리즘과 스케쥴러 선택 알고리즘을 제시한다.

Abstract

By now, Schedulers for RMS and EDF are implemented for real-time Linux Scheduler. These Schedulers are used for do not consider there's characteristics. Missing Schedulability-test cause result that increase deadline miss rate. Also The present real-time Linux causes system halt Because of scheduling for unschedulable tasks. These appearances are very fatal for real-time system.

Therefor, In this paper, The peaceful schedulability-test use scheduler which is proper characteristics of RMS and EDF scheduling methods. This scheduler keeps deadline and eliminates system halt from scheduling unschedulable tasks. In this paper, we propose the schedulability-test algorithm and scheduler select algorithm for the effective management of tasks sets.

* 오산대학 정보관리과 조교수

논문접수 : 2002. 3. 29
심사완료 : 2002. 5. 28

Ⅰ. 서론

실시간 시스템은 실세계에서 어떤 시간 내에 사건을 발생하기 위해서 재실행되어야만 하는 비 실시간 시스템과는 다르다[1]. 전형적인 실시간 시스템은 외부 프로세스를 감시하고 제어하며, 시기 적절하게 외부 프로세스에 변화를 알리고 반응해야만 한다. 만약 외부 프로세스가 단순하다면, 외부 사건의 몇 가지 유형에 즉각적으로 반응하는 단일 마이크로컴퓨터로 만족된다.

그러나 많은 실시간 시스템은 더 복잡하며, 더 많은 프로세서와 소프트웨어구조, 여러 동작들간의 더 복잡한 조합을 요구한다[2]. 이는 가장 단순한 실시간 시스템을 제외한 모든 것에서 다양한 동작들에 대해 스케쥴링하는 것이 필요함을 의미한다. 실시간 시스템에서 수행되는 태스크들은 시간적 제약 조건을 갖는다[2][3]. 이러한 시간 제약 조건으로 가장 널리 사용되고 있는 것은 종료시한이다[4]. 실시간 태스크는 종료시한이 지켜지지 못할 경우 실행 결과에 대한 가치를 상실하게 되는데, 그 정도에 따라서 경성 실시간(hard real-time) 태스크와 연성 실시간(soft real-time) 태스크로 구분된다[5][6]. 경성 실시간 태스크는 종료시한이 지켜지지 못할 경우 실행 결과의 가치가 0 또는 음의 값을 갖게 된다. 반면 연성 실시간 태스크의 경우에는 그 가치가 종료시한을 넘어서도 점진적으로 감소하는 특징이 있다.

기존 시분할 시스템은 시스템의 설계 시 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[7]. 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.

실시간 Linux에서는 현재 실시간 태스크의 개발을 위한 인터페이스를 제공해주고 있지만, 아직 다양한 스케쥴러의 지원이 부족하다. 그러므로 본 논문에서는 실시간 Linux에서 기본적으로 제공하는 비율 단조형 스케쥴링과 종료시한 우선 스케쥴링 알고리즘 기법을 사용하여 태스크 집합이 제출되었을 경우 각 태스크 집합에 적합한 스케쥴러를 제공하도록 스케쥴링 가능성 분석 후 적합한 알고리즘을 시스템이 선정하도록 하며 안정된 스케쥴링을 할 수 있도록 혼합 스케쥴링 기법을 설계한다.

Ⅱ. 관련 연구

지금까지 여러 종류의 실시간 스케쥴링 방법들이 개발되어 왔다. 이 장에서는 스케쥴 가능성검사와 실시간 Linux에서 지원하는 비율 단조형 스케쥴링 알고리즘과 종료시한 우선 스케쥴링 알고리즘에 관하여 살펴보기로 한다.

2.1 스케쥴링 가능성(Schedulability)

스케쥴링 측면에서 주어진 주기적 태스크 집합에 대하여 종료시한을 만족할 수 있는지의 여부를 조사하는 것을 스케쥴링 가능성 검사(schedulability test)라고 한다[8]. 만약 주어진 태스크 집합의 모든 태스크들이 종료시한 내에 실행을 완료할 수 있다면 그 태스크 집합은 "스케쥴링 가능하다"(schedulable)라고 한다[8][9].

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

2.2 표기법

본 논문에서 사용하게 될 표기들은 다음과 같다.

표 1. 본 논문에서 사용하는 표기법

표기법	의미
n	태스크 집합의 태스크 개수
τ_i	태스크 집합의 태스크 ($1 \le i \le n$)
C_i	τ_i의 최악경우의 수행 시간
T_i	τ_i의 주기
D_i	τ_i의 종료시한
I_i	τ_i의 위상(phase)

임의의 태스크 집합의 태스크 $\tau_1 \cdots, \tau_n$에 대해서, 각 태스크는 다음과 같이 네 가지 구성 요소로 나타낼 수 있다.

$(C_i, T_i, D_i, I_i), \quad 1 \le i \le n$

2.3 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 알려졌으며, 주기를 기초로 태스크에 정적 우선순위를 할당하고 짧은 주기의 태스크에 높은 우선 순위를 할당하며 태스크가 도착할 때 우선 순위가 할당되므로 우선 순위는 재 계산될 필요가 없다[10]. 이것은 단일 프로세서 상에서 고정 우선 순위 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 swit ching)과 태스크 스케줄링에 드는 비용은 무시한다.
③ 모든 태스크들은 서로 독립이다. 즉, 선행 제약(pre cedence constraints)은 존재하지 않는다.

RMS는 고정 우선 순위 방법으로 태스크의 우선 순위는 자신의 주기에 반비례한다. 따라서, 작업의 주기가 주어지면 자신의 우선 순위는 정적(static)으로 정해진다. 만일 태스크 τ_i의 주기가 τ_j보다 작다면, τ_i의 우선 순위는 τ_j보다 높다. 높은 우선 순위 태스크는 낮은 우선 순위 태스크를 항상 선점 할 수 있다. 따라서 태스크 집합 $\tau_1, \tau_2, \cdots, \tau_n$에 대해서, τ_1의 우선 순위가 가장 높고, τ_n의 우선 순위가 가장 낮다고 정의할 수 있다[11][12].

태스크가 주기적이기 때문에, 매 주기마다 한번씩 수행된다. 태스크의 매 수행을 작업(job)이라 하자. τ_i의 j 번째 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 한다.

표 2. 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

2.4 종료시한 우선(Earliest-Deadline First) 스케줄링 알고리즘

RMS 알고리즘과는 달리 Earliest-Deadline First (이하 EDF) 알고리즘은 동적 우선 순위 스케줄링 방법으로 Liu와 Layland에 의해서 제시되었다[5]. EDF는 동적 알고리즘으로 선점 알고리즘들 중에서 최적인 것으로 알려져 있다. 태스크의 우선 순위는 종료시한에 따라서 할당된다. 즉, 종료시한이 짧을수록 높은 우선 순위가 할당되므로 임의의 순간에 실행되는 태스크는 실행이 완료되지 않은 태스크들 중에서 종료시한에 가까운 것이 선택된다. 또 정적 알고리즘과 달리 태스크의 우선 순위가 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[13][14].

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

또한 EDF 알고리즘은 동적 우선순위 알고리즘들 중에

서 최적이다. 즉, EDF 알고리즘으로 스케쥴링 될 수 없으면 다른 동적 우선 순위 알고리즘으로도 스케쥴링 될 수 없다.

표 3. RMS와 EDF 스케쥴링 알고리즘의 장단점

	비율단조 (Rate Monotonic)	종료시한 우선 (Earliest Deadline First)
특징	-정적 우선 순위 스케쥴링 방식 -경성 실시간용 -선점형 -고정 우선 순위 기법 -스케쥴 가능성 : $n(2^{1/n}-1)$ -태스크의 우선 순위는 주기가 짧은 작업이 최우선 →정적 알고리즘 중 최적임이 증명됨	-동적 우선 순위 스케쥴링 방식 -경성 실시간용 -선점형 -스케쥴링 시점에서 볼 때 종료시한 까지의 거리가 짧은 태스크가 더 높은 우선 순위를 배정하여 스케쥴링 →동적 알고리즘중 최적임이 증명됨
특이사항	-종료시한이 주기와 같은 태스크 모델에서 최적임이 증명됨 -종료시한이 주기보다 작은 경우는 최적이지 못함 -약 88% 실용성 → 실시간 시스템 구현시 간단한 특성으로 많이 사용	-종료시한이 주기와 동일 하거나 주기에 비례하여 작을 경우 비율단조 스케쥴링 알고리즘과 같다. - 종료시한이 주기보다 작은 경우 최적
제약조건	-서로 독립적인 주기 태스크만을 고려 -태스크간의 선행관계가 없음 -문맥교환 및 태스크 스케쥴링 비용 무시 -우선 순위 역전 현상은 발생가능	- 스케쥴링 오버헤드

Ⅲ. 실시간 Linux의 효율적인 스케쥴링을 위한 선택 알고리즘 구현

본 장에서는 실시간 Linux에서 효율적인 태스크 관리를 위해 RMS와 EDF 스케쥴링 기법을 선택하여 효율적인 스케쥴링을 하도록 선택 알고리즘을 구현한다.

3.1 선택 알고리즘의 설계 및 구현

실시간 운영체제에서 태스크의 스케쥴링은 매우 중요한 역할을 수행한다. 실시간 시스템에서는 각각의 태스크들마다 종료시한이 주어져 있고 이 태스크들이 종료시한 내에 수행되지 못하는 경우에는 큰 피해를 입을 수 있으므로 반드시 지켜져야 한다. 지금까지 종료시한을 보장하기 위한 여러 스케쥴링 방법들이 연구되어 왔으며 본 논문에서는 이런 방법 중 우선 순위가 고정적으로 주어진 주기 태스크 집합들에 대한 정적 스케쥴링 알고리즘들 중 최적 알고리즘이라 알려진 RMS와 태스크의 수행 중에 종료시한에 가까울수록 우선 순위를 높게 해주는 동적 우선 순위 할당 방법을 사용하는 EDF에 대해서 다루었다. 또한 기존의 RMS나 EDF에서 실시간 태스크들의 집합이 종료시한을 보장받는가를 검사하는 방식을 확장하여 좀 더 정확한 스케쥴링 검사방법을 제시하며, 실시간 Linux 운영체제 상에서 확장된 스케쥴링 가능성 검사를 사용하여 RMS와 EDF를 기반으로 스케쥴하는 선택 알고리즘을 설계하고 구현한다.

3.2 혼합된 스케쥴링 가능성 검사

현재까지 실시간 Linux를 위한 스케쥴러는 RMS와 EDF 두 가지의 스케쥴러가 별도로 구현되어 있다. 이 두 가지 스케쥴러 중에서 사용자가 각각의 스케쥴링 알고리즘의 특성을 고려하지 않고 두 가지 방법을 선택하여 사용하고 있다. 이로 인해 실시간 시스템의 스케쥴링 가능성 검사의 미 수행으로 종료시한 miss rate를 증가시키는 결과를 초래한다. 또한, 현재 실시간 Linux에서는 스케쥴 불가능한 태스크에 대한 스케쥴로 인하여 시스템이 정지되는 현상이 발생된다. 이러한 현상은 경성 실시간 시스템에서는 매우 치명적이다.

그러므로 본 논문에서는 이러한 단점들을 해결하기 위한 안정적인 스케쥴링 가능성 검사를 통하여 RMS와 EDF 스케쥴링 방법의 특성에 맞게 적절한 스케쥴러를 선택함으로서 종료시한을 보장하고 또한 스케쥴이 불가능한 경우 태스크 스케쥴로 인해 발생되는 시스템 정지 현상을 제거하였다. 위에서 제시한 방법들의 알고리즘은 다음과 같다.

```
double Schedulable_check(NTASKS)
{
/* 태스크 집합에 대해 주기와 종료시한을 검사하여 동일한가를 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);
}
```

그림 1. 스케줄링 가능성 검사 알고리즘

```
Scheduler_Select()
{
  Util = Schedulable_check(NTASKS);
  /* 태스크 집합에 대한 스케줄 가능성 검사 */
  if( Util >=0 && Util <= NTASKS * ( POW(2, (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; clean_module(); }
/* 스케줄 불가능, 태스크를 큐에서 제거하여 리눅스 커널 모드로 전환하여
시스템 정지 현상 예상 */

  if( SCHED_OK && RMS)        /* RMS로 태스크 스케줄링 */
    {
      for (x=0; x<NTASKS; x++)   /* 태스크 구조체 초기화 */
        mon_rt_task_init(&(tasks[x]),  mon_fun, Parameters[x].compute ,
        3000, x+1);
      for (x=0; x<NTASKS; x++)
      /* 태스크에 주기를 부여하여 태스크 생성 */
        mon_rt_task_make_periodic(&(tasks[x]), now+(RTIME)1000,
         (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  ,
        3000, x+1, x+1);
      for (x=0; x<NTASKS; x++)
        edf_rt_task_make_periodic(&(tasks[x]), now+(RTIME)1000,
            (RTIME)Parameters[x].period,
             (RTIME)Parameters[x].deadline);
           }
    return 0;
    }
```

그림 2. 스케줄 가능성 결과에 따른 스케줄러 선택 알고리즘

Ⅳ. 성능평가

성능평가에 사용된 시스템 환경은 IBM PC Pentium 급에서 Linux 커널 2.0.29버전을 기반으로 하였으며, 실시간 Linux에서 제공하는 비율단조형 스케줄러와 Ismael Ripoll에 의해 구현된 종료시한 우선 순위 스케줄러를 인스톨하였다. 성능평가에 사용된 태스크 집합에 대한 매개변수는 표 4와 표 5에 제시되었다.

표 4. 성능평가 매개변수(주기=종료시한)

실험별 \ Task		TASK 집합					이용률
		τ_1	τ_2	τ_3	τ_4	τ_5	
실험1	C_i	50	60	70	80	90	0.719
	T_i	300	400	500	600	700	
실험2	C_i	58	60	70	80	100	0.760
	T_i	300	400	500	600	700	
실험3	C_i	70	80	90	100	120	0.937
	T_i	300	400	500	600	700	
실험4	C_i	80	90	100	110	120	1.046
	T_i	300	400	500	600	700	

표 5. 성능평가 매개변수 (주기≠종료시한)

실험별 \ Task		TASK 집합					이용률
		τ_1	τ_2	τ_3	τ_4	τ_5	
실험5	C_i	20	30	40	50	60	0.719
	D_i	280	362	450	560	700	
	T_i	300	400	500	600	700	
실험6	C_i	20	30	40	50	60	0.760
	D_i	265	366	450	560	700	
	T_i	300	400	500	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 Linux에서 제공하는 RMS는 실시간 태스크들을이 먼저 처리한 후 기존의 리눅스 커널을 낮은 우선 순위 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.

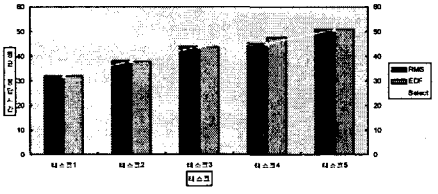

그림 3. 이용률이 0.719인 경우의 평균 응답시간
(단위: 클럭틱)

그림 3은 일련의 태스크 집합에 대해 프로세서 이용율이 0.719인 경우 평균 응답시간을 보여주고 있다. 여기서, 먼저 기존의 RMS와 EDF로 스케줄링을 한 경우의 평균 응답시간을 구하고 제안한 선택 알고리즘 기법 으로 스케줄러를 선택하여 스케줄링을 한 경우를 비교하였다. EDF인 경우 평균 응답시간 시간을 RMS와 비교하기 위해서는 EDF의 태스크 종료시한과 주기를 동일한 경우로 설정해야 한다. 표 4에서 설명한 것처럼 EDF의 경우 주기와 종료시한이 동일한 경우 RMS의 특성을 따르기 때문에 평균 응답시간이 거의 동일하게 계산되었다. 그림 3에서 제안한 기법의 평균 응답시간은 RMS의 평균응답시간과 동일함을 알 수 있다. 이는 제안한 혼합 스케줄링 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 있다.

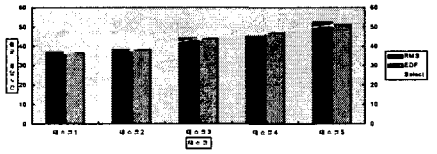

그림 4. 이용률 0.760인 경우의 평균 응답시간
(단위: 클럭틱)

그림 4에서는 태스크 집합의 프로세서 이용률 0.760으로 스케줄링 가능한 필요 충분조건인 0.743을 넘었기 때문에 RMS로 스케줄링할 경우 태스크들의 종료시한내에 수행을 보장하지 못한다. 따라서, 제안한 선택 알고리즘 기법은 EDF 기법을 선택하여 스케줄링 되었다. EDF의 경우 스케줄링 가능한 구간이 $0 \le i \le 1$이기 때문에 충분히 스케줄링 안정성을 보장한다. RMS인 경우 다행히 주어진 태스크 집합에서 종료시한을 넘겨 스케줄링된 것은 없었다.

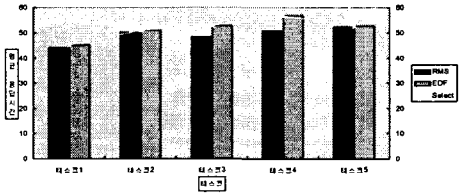

그림 5 이용률이 0.937인 경우의 평균 응답시간
(단위: 클럭틱)

그림 5는 태스크 집합에 대한 프로세서 이용률이 거의 100%에 근접하고 있다. 이러한 경우 제안한 선택 알고리즘 기법을 통해 스케줄러를 선택하여 스케줄링 하게되면 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 EDF 에서는 1개의 태스크도 종료시한을 위반함 없이 스케줄링 되었다. 역시 스케줄링 가능한 구간이 $0 \le i \le 1$에 포함되므로 안정된 스케줄링이 되었다. 한가지 고려할 사항은 평균 응답시간에서 RMS가 EDF 보다 빠르게 나타났지만 종료시한을 위반한 태스크가 발생했으므로 별 의미가 없다 하겠다.

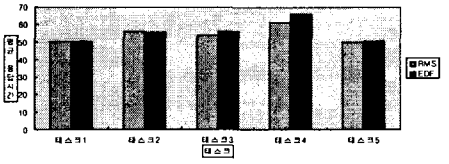

그림 6. 이용률이 1.046인 경우의 평균 응답시간
(단위: 클럭틱)

그림 6은 태스크 집합에 대한 프로세서 이용율이 100%를 초과하고 있다. 이 경우 RMS와 EDF 모두 태스크들의 종료시한 내에 스케줄링을 보장할 수 없다. 본 논문에서 제안한 선택 알고리즘 기법은 프로세서 이용율이 100%를 초과하면 시스템의 안정성 확보를 위해 실시

간 스케쥴링을 하지 않고 리눅스 커널 모드로 되돌아간다. 따라서, 그림 6은 제안한 선택 알고리즘기법이 스케쥴러를 선택하지 않아서 평균 응답시간은 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.

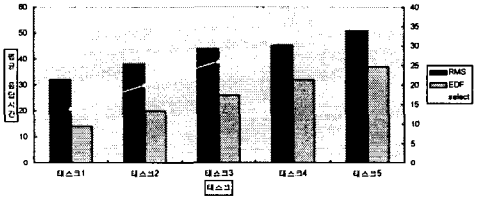

그림 7. 이용률이 0.719인 경우의 평균응답시간
(단위: 클럭틱)

그림 7은 태스크 집합 프로세서 이용율이 그림 3에서처럼 0.719인 경우이다. 이 경우 태스크 집합에 종료시한이 주기 이전에 설정된 경우 RMS로는 스케쥴링 될 수 없고 EDF로 스케쥴링이 되어야 한다. 본 성능평가 결과 EDF인 경우가 RMS보다 높은 평균응답시간을 보였다. 따라서, 혼합 스케쥴링 기법을 사용하는 제안한 선택 알고리즘은 주기와 종료시한이 다른 태스크 집합에 대해서는 EDF로 선택하여 스케쥴링을 할 수 있으므로 평균응답시간도 EDF와 동일하게 나왔다.

그림 8은 태스크 집합 프로세서 이용율이 그림 3에서처럼 0.760인 경우이다. 이는 RMS인 경우 스케쥴링의 필요충분조건을 만족하지 못하는 경우이다. 이 경우 역시 주기와 종료시한이 달리 설정되었으므로 혼합 스케쥴링시 EDF로 스케쥴링이 되며, 평균 응답시간에서도 월등한 평균 응답시간을 보였다.

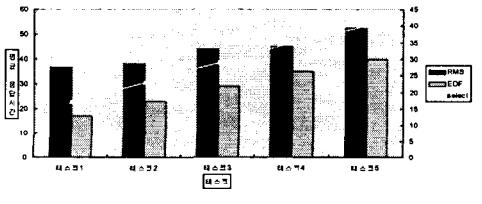

그림 8. 이용률이 0.760인 경우의 평균 응답시간
(단위: 클럭틱)

V. 결론 및 향후 연구과제

실시간 운영체제에서 태스크 스케쥴링은 매우 중요한 역할을 수행한다. 실시간 시스템에서는 각각의 태스크들마다 종료시한이 주어져 있고 이 태스크들이 종료시한 내에 수행되지 못하는 경우에는 큰 피해를 입을 수 있으므로 반드시 지켜져야 한다. 지금까지 종료시한을 보장하기 위한 여러 스케쥴링 방법들이 연구되어 왔다. 특히 본 논문에 사용된 실시간 Linux는 POSIX.1b가 일부분 구현되어있는 Linux를 기반으로 하여 경성 실시간 시스템을 구성한다.

현재까지 실시간 Linux를 위해서 두개의 스케쥴러가 구현되어 있다. RMS방법과 EDF방법이 그것이다. 이 두가지 스케쥴러 중에서 사용자가 각각의 스케쥴링 알고리즘 특성을 고려하지 않고 개별적으로 선택하여 사용하고 있다. 이는 실시간 시스템의 스케쥴 가능성 검사를 미수행 함으로써 종료시한 miss rate를 증가시키는 결과를 초래한다. 또한 현재 실시간 Linux에서는 스케쥴 불가능한 태스크에 대한 태스크 스케쥴링으로 인해 시스템이 정지되는 현상이 발생된다. 이러한 현상은 경성실시간 시스템에서는 매우 치명적이다.

본 논문에서는 이러한 단점들을 해결하기 위한 안정된 스케쥴 가능성 검사 방법을 제시하고, 안정된 스케쥴 가능성 검사를 통하여 RMS방법과 EDF방법의 특성에 맞게 적절한 스케쥴러를 선택하여 사용함으로서 종료시한을 보장하고, 또한 스케쥴이 불가능한 경우 태스크 스케쥴로 인해 발생되는 시스템 정지 현상을 제거하였다.

향후 연구과제로서 비주기적인 태스크를 실시간 태스크의 수행후 여분의 슬랙(slack)에서 실행할 수 있도록 기존의 여러 비주기적 실시간 스케쥴링 알고리즘을 실시간 Linux에 포팅하여 본 논문에서 구현한 스케쥴 가능성 검사 방법을 보다 확장하여 스케쥴러를 선택가능 하도록 한다면 향상된 실시간 Linux가 될 것이다.

참고 문헌

[1] 안병철, "PC환경 실시간 운영체제를 위한 멀티태스킹 커널의 설계 및 구현", 영남대학교, 1996.

[2] An Fredette and R.cleaveland, "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.

[9] 김인국, "복합 태스크 모델에 대한 효율적인 실시간 스케줄링", 한국정보처리학회 논문지 제3권, 제6호, pp1568-1579, Jun. 1997.

[10] Mckusick, M.K., Bositic, K., Karels, M.J. and Quarterman, J.S., "The Design and Implementation of The 4.4 BSD Unix Operating System", Addison-Wesley, 1996.

[11] Wind River Systems. Inc., 1010 Atlantic Avenue, Alameda, CA 94501-1147, USA. VxWORKS Programmer's Guide 5.1, Dec. 1993.

[12] Sha, L., Rajkumar, R. and Lehoczky, J.P., "Priority inheritance protocols : An approach to real-time synchronization", IEEE Transactions on Computers, 39(9), pp.1175-1185, Sept. 1990.

[13] S-C Cheng, et al, "Scheduling Algorithms for Hard Real-Time Systems - A Brief Survey", Tutorial : Hard Real-Time Systems, IEEE Computer Society Press, 1988.

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

저자소개

김 성 락

1984년 울산대학교 전자계산학과 졸업(학사)

1989년 한양대학교 산업대학원 전자계산학전공(석사)

2000년 수원대학교 컴퓨터학과 박사과정 수료

1996년~현재 오산대학 정보관리과 조교수

관심분야 : 분산운영체제, 웹프로프로그래밍 등