경성 실시간 태스크를 위한 확장된 가능성 검사를 통한 비율단조 기반 스케쥴링 기법

신동헌[O] 이준택 조수현 김영학
금오공과대학교 컴퓨터공학과 대학원
{genius[O], next, shcho, yhkim}@cespc1.kumoh.ac.kr

The Scheduling Technique Based on Rate-Monotonic with Extended Schedulability Inspection for Periodic Task in Hard Real-Time System

Dong-Hern Shin[O] Joon-Taek Lee Soo-Hyun Cho Young-Hak Kim
Graduate School, Dept. of Computer Engineering, Kumoh National Institute of Technology

요 약

경성 실시간 시스템(Hard Real-Time System)에서는 주기 태스크들의 엄격한 마감시간(Deadline) 보장이 시스템의 성능을 좌우한다. 본 논문에서는 CPU의 이용률(Utilization)이 높아 비율단조 정책으로는 마감시간을 보장 할 수 없는 주기 태스크들을 위해 확장된 스케쥴 가능성 검사를 통해 수행할 태스크들의 공통 주기(L.C.M : Least Common Multiple)내에서 EDF(Earliest-Deadline First) 정책을 기반으로 마감시간 보장 수행패턴(Feasible Pattern)을 찾고, 이를 참조하여 우선순위를 고려하지 않고 태스크들을 강제 수행할 수 있게 하는 비율단조 기반의 스케쥴링 기법을 제안한다. EDF를 기반으로 생성된 패턴은 EDF 정책의 특성에 따라 CPU의 이용률을 100% 까지 가능하게 하며 패턴을 참조하여 강제 수행함으로써 EDF 정책이 갖는 실행시간 스케쥴링 오버헤드를 없앨 수 있다.

1. 서 론

실시간 시스템은 동작의 정확성이 논리적 정확성뿐만 아니라 시간적 정확성에서도 좌우되는 시스템을 의미한다. 실시간 시스템에 존재하는 시간 제약 조건은 마감시간으로 주어진다. 마감시간은 엄격성에 있어서 경성(Hard), 준경성(Firm), 연성(Soft) 마감시간으로 분류해 볼 수 있다. 실시간 시스템이 마감시간을 만족시키기 위해서는 고속의 계산을 요구하게 되지만, 고속의 계산이 실시간 시스템의 요구조건을 만족시키는 것은 아니다. 일반적으로 고속의 계산은 시스템의 평균 응답시간을 최소화하지만, 실시간 시스템에서 요구되는 예측가능성 즉, 태스크의 마감시간 보장을 의미하지 않는다[1].

본 논문에서는 확장된 스케쥴 가능성 검사를 수행하는 비율 단조 기반의 스케쥴링 기법을 제안한다. 이 스케쥴러는 확장된 스케쥴 가능성 검사를 통해 스케쥴은 가능하나 확실하게 마감시간을 보장할 수 없는, 즉, CPU의 이용률이 높은 태스크 셋을 찾아내고 검사 시에 미리 마감시간을 만족하는 수행패턴을 찾아 이를 참조하여 태스크들을 순서대로 수행할 수 있게 한다. 또한 불필요한 오버헤드를 줄이기 위해 비율 단조 방식으로 스케쥴이 가능하고 마감시간이 보장되는 즉, CPU의 이용률이 낮은 경우 확장 검사를 실시하지 않고 기존의 비율 단조 정책에 따라 각 태스크에 고정 우선순위를 할당하고 그에 따라 수행 할 수 있게 한다.

마감시간을 만족하도록 생성되는 수행패턴은 EDF 정책을 따를 경우와 동일한 순서가 된다. 따라서 실행시간에 우선순위를 계산하지 않아 스케쥴링 오버헤드가 없는 RM 정책의 장점과 높은 CPU 이용률에도 적용 가능한 EDF 정책의 장점을 모두 수용하고 있다.

2. 관련 연구

2.1 Rate-Monotonic (RM)

비율 단조 알고리즘은 최적의 정적 알고리즘으로 알려져 있고 많이 연구되고 사용되는 방법 중의 하나로 다음과 같은 가정 하에 가능하다[2][4].

1) 마감시간을 갖는 모든 태스크들은 주기적이며, 2) 각 태스크에 대한 다음 요청은 이전 요청이 완료된 후 가능하고, 3) 각각의 태스크들은 독립적이다. 4) 각 태스크에 대한 수행시간은 상수이며 수행시간 내에서 인터럽트는 없고, 5) 시스템에 있는 다른 비주기 태스크들은 고려하지 않고 주기 태스크 수행에 드는 비용을 제외한 모든 비용은 무시한다.

독립적이라고 가정된 각 태스크는 수행 전에 정적으로 우선순위를 부여 받게 된다. 우선순위는 각 태스크의 주기를 기준으로 하며 주기가 짧을수록 높은 우선순위가 주어진다.

다음은 Liu와 Layland에 의해 제안된 스케쥴 가능성 검사 방법이다[4].

주기 태스크 집합 $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$가 고정 우선순위 방식으로 스케쥴링 된다고 가정 할 때, 우선순위 순으로 (τ_1이 가장 높다.) 정렬 되어있다고 가정한다. T_i는 τ_i의 주기, C_i는 수행시간(Completion time), C_i/T_i는 τ_i의 프로세서 이용률(Utilization, U_i)라 하면, 모든 태스크들의 CPU 이용률은 다음과 같다. $(1 \le i \le n)$

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots\ldots + \frac{C_n}{T_n} \qquad (1)$$

식 (1)의 이용률 U가 다음을 만족하면 태스크 집합 τ의

모든 태스크들은 동시에 마감시간을 만족할 수 있다.

$$U \leq n(2^{\frac{1}{n}} - 1) \qquad (2)$$

식 (2)에서 $n \rightarrow \infty$ 이면, $U = \ln 2$ (약 0.693~)이다. 즉, 주기를 기반으로 정적 우선순위를 부여하는 알고리즘에서는 CPU의 이용률이 최대 약 0.693(69.3%) 까지 이다. 따라서, 만약 태스크들의 CPU 이용률이 최대값보다 작으면 마감시간을 만족하지만 최대값보다 크고 1(100%) 보다 작으면 스케쥴링 가능한지 즉, 마감시간을 만족하는지 판단 할 수 없게 됨을 뜻한다. 태스크 실행 전에 우선순위가 할당 되기 때문에 우선순위 재계산을 위한 오버헤드가 없고 마감시간과 주기가 같을 때 최적이다.

2.2 Earliest-Deadline First (EDF)

마감시간 우선 알고리즘은 동적 우선순위 알고리즘으로는 최적으로 알려져 있다[3][4]. RM 정책과 달리 동적 우선순위를 할당하는 방식으로 태스크의 주기가 아닌 마감시간을 기준으로 임의의 실행시점에서 실행이 완료되지 않은 태스크들 중에서 가장 가까운 마감시간을 갖는 태스크에게 다음 우선순위를 할당한다. 즉, 마감시간이 짧을수록 높은 우선순위가 할당된다.

주기 태스크 집합 $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$가 동적 우선순위 방식으로 스케쥴링 된다고 가정 할 때, T_i는 τ_i의 주기, C_i는 수행 시간이라 하면, 다음 조건을 만족할 때 이 태스크들은 스케쥴 가능하다고 할 수 있다. $(1 \leq i \leq n)$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \ldots\ldots + \frac{C_n}{T_n} \leq 1 \qquad (3)$$

식 (3)에서 동적 우선순위 알고리즘은 CPU 이용률이 최대 1(100%)까지 가능함을 알 수 있다. 그러나 새로운 태스크를 할당 할 때 마다 우선순위를 계산하기 때문에 실행시간 스케쥴링 오버헤드가 크다. 마감시간이 주기와 동일하거나 주기에 비례하여 작을 경우 RM과 같고 마감시간이 주기보다 작을 때 최적이다.

2.3 Slack Stealing

슬랙은 CPU의 여유시간을 말한다. 즉, 주기 태스크들이 각자의 주기 내에서 실행을 완료하고 남은 시간을 말하는데 임의의 시간에서 실행되고 있는 주기 태스크들이 없을 때 비주기 태스크에게 할당하기 위해 주기 태스크로부터 슬랙을 빼앗아 비주기 태스크의 응답시간을 최소화 하고자 하는 방법이다[5].

3. 제안된 스케쥴 가능성 검사 기법

관련 연구를 바탕으로 확장된 스케쥴 가능성 검사 기법을 제안한다.

본 논문에서 제안하는 기법은 다음과 같은 제한 사항들을 갖는다. 태스크 수행에 필요한 비용을 제외한 모든 비용은 무시하고, 주기 태스크의 주기는 상수이며, 수행 중 인터럽트는 없고 마감시간은 주기보다 크거나 같다고 가정한다.

<그림 1>은 본 논문에서 제안하고 있는 확장된 스케쥴 가능성 검사 기법을 나타내고 있다. 먼저, 임의의 시간에 주기 태스크 τ_i가 도착하면 τ_i와 동시에 스케쥴링이 되어야 할 이전 태스크들과 τ_i의 CPU 총 이용률 U를 구한다. 계산된 U가 1 보다 클 경우는 τ_i를 포함한 모든 태스크들의 U가 100%를 넘는다는 것을 의미하므로 τ_i는 스케쥴링 불가능하다는 것을 알 수 있다.

```
주기 태스크 τn 도착, (n>1)

U = Σ_{i=1}^{n} Ci/Ti   /* 이용률(Utilization) 계산*/

if(U ≤1){
    if(U ≤ n(2^(1/n) - 1)){
        RM 정책 스케쥴링
    }else{
        가능성 검사 대상 태스크들의 주기 값의 L.C.M 계
        산 /*공통 반복 주기*/
        T = (U *L.C.M) / 100 /*할당 가능한 총시간*/

        while(T){
            EDF 정책을 이용한 태스크 선택

            if(태스크 선택 불가능)
                스케쥴 불가능, 보류

            T -= 선택된 태스크의 수행 가능 시간량
            /* 선택된 태스크의 수행 가능 시간량
             과 태스크 번호를 테이블에 저장 */
        }

        테이블에 저장된 태스크 번호와 시간량을 이용하
        여 태스크 강제 수행.
    }
}else{
    스케쥴 불가능, 보류
}
```

그림 1. 제안된 스케쥴 가능성 검사 기법

그러나 U가 식 (2)를 만족할 경우 완벽하게 RM 정책으로 스케쥴링 가능하다는 것을 의미한다. 따라서 U가 1 보다 작고 식 (2)를 만족하지 않는 경우 즉, RM 정책으로는 마감시간을 보장 받지 못하는 경우이다. 본 논문에서는 이러한 경우를 위해 스케쥴 가능성 검사 기법을 확장한다. 먼저 이와 같은 경우에는 RM 정책으로는 마감시간을 보장 받을 수 없지만 U가 1 보다 작기 때문에 EDF 정책을 이용하면 보장 가능성이 커질 수 있다.

따라서 EDF 정책을 이용하여 마감시간을 보장할 수 있는 수행패턴과 수행가능 시간량을 계산한다. 그러나 EDF 정책이 실패하면 역시 스케쥴 불가능 하게 된다. 계산된 시간(U)을 실제 L.C.M까지의 실제 시간량으로 바꾸고 찾아진 태스크와 시간량의 쌍을 테이블에 유지 하면서 남은 시간량이 0이 될 때 까지 반복한다. 탐색이 한번 끝나면 각 태스크들의 L.C.M 을 공통주기로 하여 반복할 수 있는 마감시간 보장 수

행 패턴이 테이블에 저장된다[6].

이렇게 생성된 패턴을 우선순위로 그대로 적용하여 순차적으로 계산된 시간만큼만 실행된 후, 테이블에서 참조되고 있는 다음 태스크가 자동으로 선택되어 실행된다. 패턴 생성 후 공통 주기까지 실행하면 그 다음 공통 주기내에서 패턴을 그대로 다시 사용 할 수 있기 때문에 다른 태스크가 도착하지 않으면 더 이상의 스케쥴링 오버헤드는 없다.

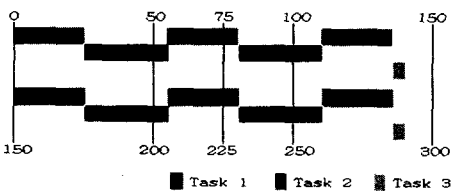

그림 2. 생성된 패턴의 반복

<그림 2>는 L.C.M을 주기로 패턴이 반복됨을 보여준다. Task 1, 2, 3의 주기와 수행시간은 각각 (50, 25), (75, 30), (150, 5)이다. 이용률 U = 95% 이고 확장된 가능성 검사를 통해 각 태스크 주기의 L.C.M인 150을 공통주기로 반복됨을 알 수 있다.

4. 평가

<그림 3>의 빗금 친 부분은 본 논문에서 제안하고 있는 기법이 적용되는 범위를 나타내고 있다. (식 (2))

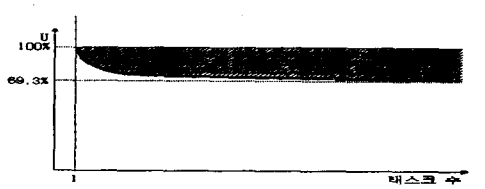

그림 3. 제안된 기법의 적용 범위

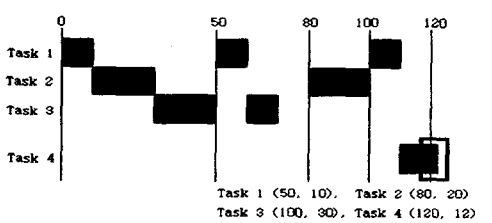

그림 4. RM 정책 결과

<그림 4>는 주어진 주기와 수행시간에 따른 태스크들의 스케쥴 가능여부를 보여준다.

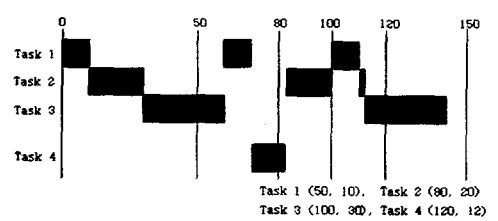

그림 5. 제안된 기법에 의한 EDF 정책 결과

<그림 4>에서 태스크 1, 2, 3은 식 (2)에 의해 마감시간을 보장 받는다.(U = 75%) 그러나 태스크 4가 추가될 경우 보장 받지 못한다.(U = 85%) 따라서 RM 정책 적용 시 <그림 4>의 결과와 같이 실패한다. 마감시간을 보장 받지 못할 경우 제안된 기법에서는 EDF 정책을 적용하여 <그림 5>와 같이 각 태스크들의 마감시간을 보장 할 수 있다. <그림 2>의 경우처럼, 생성된 수행 패턴은 각 태스크 주기의 L.C.M을 공통주기로 반복 되며 미리 생성된 패턴을 참조하여 수행 하므로 실시간 스케쥴링 오버헤드가 없다.

5. 결론 및 향후 연구 과제

본 논문에서는 RM 정책을 기반으로 하는 정적 우선순위 할당 알고리즘에서, 좀 더 정확하고 효율적인 스케쥴을 하기 위해 확장된 스케쥴 가능성 검사 기법을 제안 하였다. RM 정책으로 마감시간을 보장 할 수 없는 태스크들을 위해 미리 수행 패턴을 생성함으로써 정적 우선순위 알고리즘이 갖는 정확성과 신속함을 유지하면서 동적 우선순위 알고리즘이 갖는 높은 이용률을 얻을 수 있다. 그러나 몇 가지 제한된 가정 하에서 논의된 사항이다.

각 태스크 주기의 L.C.M 값을 이용하여 공통 주기를 찾아 반복 패턴을 찾음으로써 스케쥴링 오버헤드를 줄이고 이 값이 작을수록 최적이라 할 수 있다. 하지만 값이 커짐에 따라 오버헤드는 증가한다. 따라서 실제 예측 가능한 시스템을 위한 실질적인 논의가 뒤따라야 할 것이고, 다양한 실제 환경에서의 성능평가 또한 이루어져야 할 것이다.

참 고 문 헌

[1] N.Audsley, A. Burns, M. Richardson, and A. Wellings, Hard Real-Time Scheduling : The Deadline-Monotonic Approach, Proceedings of IEEE Workshop on Real-Time Operating Systems and Software, pp. 133-137, 1991

[2] John Lehoczky, Lui Sha and Ye Ding," The Rate Monotonic Scheduling Algorithm : Exact Characteriz-ation And Average Case Behavior" , Proc. Of IEEE Real-Time Systems Symposium, pp.166-171, Dec. 1989.

[3] H. Chetto and M, Chetto," Some Results of the Earliest Deadline Scheduling Algorithm" , IEEE Trans. On Software Eng., Vol.15, No.10, pp.1216-1269, Oct. 1989.

[4] C. Liu and J. Layland," Scheduling algorithms for multi-programming in hard real-time environment" , Journal of ACM, pp.46-61, Jan. 1973.

[5] Davis, R. I." Approximate Slack Stealing Algorithm for Fixed Priority Preemptive Systems" , Dept. of Computer Science University of York, UK, Report Number YSC-93-217

[6] S.T. Cheng and A.K. Agrawala. "Allocation and scheduling of real-time periodic tasks with relative timing constraints" In Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applicatio ns, J 1997.