멀티프로세서 환경에서 슬랙 체크테이블(SCT)을 이용한 비주기 태스크 스케쥴링 알고리즘

서순영°, 임종규, 구용완
수원대학교 전자계산학과

Aperiodic Task Scheduling Algorithm using Slack Check Table in Multiprocessor Environment

Soon-Young Seo°, Jong_Kyu Lim, Young-Wan Koo
Dept. of Computer Science, University of Suwon

요 약

단일프로세서 환경에서 주기 태스크의 시간 제약 조건을 만족시키면서 비주기 태스크의 평균 서비스 시간을 최소화하는 연구가 계속 진행되어 왔으나, 최근의 시스템은 여러 개의 프로세서를 병렬처리하여 프로그램의 처리속도 향상을 꾀하는 멀티프로세서 환경으로 전환되어 가는 추세다. 따라서, 멀티프로세서 환경에서의 태스크 스케쥴링에 관한 다양한 연구가 필요하게 되었다. 일반적으로 멀티프로세서 환경에서는 주기 태스크를 스케쥴링하기 위해서 RMFF(Rate-monotonic First-Fit) 스케쥴링 알고리즘을 사용하는데, 이 알고리즘에서는 비주기 태스크의 스케쥴링을 위한 알고리즘을 추가로 요구한다.

따라서, 본 논문에서는 멀티프로세서 환경에서 주기 태스크와 비주기 태스크가 혼합되어 있을 때, 기존의 RMFF 방식에 슬랙 체크테이블(Slack Check Table;SCT)을 이용하여 비주기 태스크를 효율적으로 스케쥴링하기 위한 방법을 제안한다.

1. 서 론

실시간 시스템은 계산의 논리적인 적합성뿐만 아니라 시간적인 적합성을 만족해야만 하는 시스템이다. 이러한 실시간 시스템의 시간제약조건은 주로 마감시간(deadline)으로 주어지는데, 이러한 마감시간은 계산결과의 정확성과 태스크들에 부여된 시간제한을 엄격히 요구하는 경성 마감시간과, 시간제한이 존재하기는 하지만 시간제한을 지키지 못하면서 태스크를 종료해도 어느 정도의 가치를 가지는 연성 마감시간으로 나누어진다. 경성 또는 연성 마감시간을 갖는 태스크들은 주기적일 수도 있고 비주기적일 수도 있다. 이러한 태스크들의 성공적인 스케쥴링은 실시간 시스템의 성능을 결정짓는 중대한 문제중 하나이다.

단일 프로세서 환경에서 태스크를 할당하는 문제는 이미 충분한 연구가 이루어지고 있으나, 현재는 단일프로세서 환경보다는 다수개의 프로세서를 병렬처리하여 프로그램의 처리속도 향상을 목적으로 하는 멀티프로세서 환경이 보편화되고 있다.

멀티프로세서 환경에서 주기 태스크는 경성 마감시간 태스크로 간주하고 정적으로 태스크를 할당하는 방식을 채택하며, 인터럽트나 복구처리등에 의해 발생하는 비주기 태스크는 연성 마감시간 태스크로 간주하고 동적으로 태스크를 할당하는 방식을 채택하기도 한다. 주기 태스크와 비주기 태스크가 함께 존재하는 혼합 태스크의 스케쥴링 목적은 주기 태스크의 정적 우선순위 환경에서 비주기적으로 발생하는 비주기 태스크의 응답시간을 최소화하는 것이다.

따라서 멀티프로세서 환경에서의 태스크 스케쥴링 알고리즘에 관한 연구의 필요성이 부각되고 있다. 그러나 멀티프로세서 환경의 다양성 및 멀티프로세서 스케쥴링의 어려움등으로 인해 아직은 많은 연구가 이루어지지 않고 있는 실정이다.

본 논문에서는 멀티프로세서 환경에서 비주기 태스크를 효율적으로 스케쥴링하기 위한 방법을 살펴보고, 그에 대한 알고리즘을 제안하고자 한다.

2. 관련연구

스케쥴링 알고리즘은 태스크의 자원과 시간제한이 충족되도록 각 프로세서에 태스크의 실행순서를 결정한다. 이러한 스케쥴링 알고리즘은 크게 정적 우선순위 알고리즘과 동적 우선순위 알고리즘으로 나뉘는데, 동적 우선순위 알고리즘은 정적 우선순위 알고리즘에 비해 스케쥴링 가능성이 우수한 반면, 스케쥴링의 오버헤드가 크다. 이와는 달리 정적 우선순위 알고리즘은 스케쥴링 가능성은 낮지만, 평균 프로세서 이용률이 높기 때문에 실시간 시스템의 설계에는 구현이 복잡한 동적 우선순위 알고리즘에 비해 상대적으로 많이 사용되고 있다. 그러나, 신뢰할 수 있는 시스템 환경의 설계를 위해서는 비주기적으로 발생하는 태스크를 더 효율적으로 처리하기 위한 방법이 필요하다.

본 장에서는 멀티프로세서 환경에서 주기 태스크 및 비주기 태스크에 대한 알고리즘을 간략히 살펴보기로 한다.

2.1 주기 태스크 스케쥴링 알고리즘

주기 태스크를 스케쥴하기 위한 방법으로 RMFF 스케쥴링 알고리즘이 있다. RMFF 스케쥴링 알고리즘은 Liu와 Layland에 의해 단일프로세서 환경에서 주기 태스크 스케쥴링 알고리즘중 최적으로 알려진 RM(Rate-Monotonic) 스케쥴링 알고리즘을 기준으로 스케쥴링 가능성 검사를 한다. 이는 주어진 태스크 집합의 스케쥴링 가능성은 주어진 주기 태스크 집합

의 프로세서 이용률(Utilization: U)이 $n(2^{1/n}-1)$ (n은 태스크의 수)보다 작거나 같으면 스케줄이 가능하며, 일단 주기 태스크 세트가 도착하면 각 태스크는 주기가 작은 태스크부터 순서대로 정렬된 다음, 【알고리즘 1】에 나타난 RMFF 스케줄링 알고리즘에 의해 스케줄링된다.

```
Phase 1. i=N=1
Phase 2. (a) j=1
         (b) if ( t_i가 RM방식으로 P_j에 스케줄가능)
               프로세서 P_j에 t_i할당하고 goto Phase 3
         (c) j=j+1; goto Phase 2(b);
Phase 3. if (j>N) N=j;
Phase 4. if (모든 태스크 할당) 실행 종료;
           else {i=i+1; goto Phase 2;}
```

【알고리즘 1】 RMFF 스케줄링 알고리즘

2.2 비주기 태스크 스케줄링 알고리즘

비주기 태스크 스케줄링 알고리즘은 임의의 시간에 도착한 비주기 태스크를 동적 우선순위 알고리즘으로 스케줄링한다.

주기 태스크의 마감시간을 만족시키면서 비주기 태스크의 평균 서비스 시간을 최소화하기 위한 방법이 다양하게 연구되어 왔다. 이러한 방법들 중에서 가장 간단한 방법은 비주기 태스크를 주기 태스크보다 항상 나중에 처리하는 후위처리(Background) 방법인데, 이는 비주기 태스크의 응답시간 최소화를 기대할 수 없다는 단점이 있다. 또 다른 방법은 비주기 태스크를 실행하기 위한 주기 태스크, 즉 폴링 태스크를 별도로 설정하는 폴링 서버 방법으로 비주기 태스크를 위한 주기 기간동안 비주기 태스크가 발생하지 않는다면 그 주기 기간은 낭비가 된다. 이런 단점들을 보완한 대역폭 보존(Bandwidth preserving) 서버 방법은 서버 주기동안 비주기 태스크가 발생하지 않으면, 그 기간을 주기 태스크에게 넘겨주어 서버의 낭비되는 시간을 줄일 수 있다. 여기에는 PE(Priority Exchange Server), DS(Deferable Server), SS(Sporadic Server) 등이 있으며, 이는 비주기 태스크가 충분히 준비되어 있을 때는 유리하지만, 실제 실시간 시스템에 있어서 비주기 태스크의 발생은 그리 빈번하지 않음으로 그 효율성이 떨어진다. 또, 비주기 태스크의 도착시기가 항상 주기와 일치하지는 않으므로 동적으로 발생하는 비주기 태스크의 스케줄링에 적절히 대응하지 못한다는 한계가 드러난다. 위에 나열된 방법들의 결점을 보완하기 위한 방법으로 비주기 태스크가 발생했을 때 이를 실행시키기 위한 프로세서 시간을 주기 태스크로부터 스틸링하는 방법으로는 슬랙 스틸링 스케줄링 알고리즘이 있다.

이상과 같이 스케줄링 알고리즘에는 주기 및 비주기 알고리즘이 있으며, 본 논문에서는 이들 주기 및 비주기 태스크가 혼합된 태스크를 스케줄링하기 위하여 기존의 RMFF 방법에 슬랙 체크테이블을 이용한 스케줄링 알고리즘을 제안하고자 한다.

3. 제안한 스케줄링 알고리즘

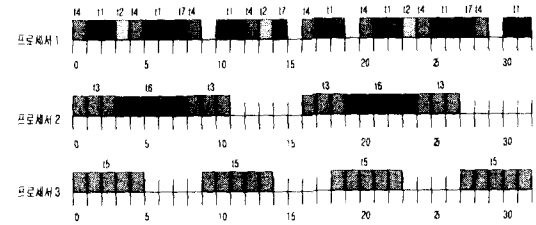
본 논문에서 제안한 스케줄링 알고리즘은 주기 태스크를 스케줄하기 위해서 기존의 RMFF 스케줄링 알고리즘을 사용하였고, 각 프로세서에서 발생하는 슬랙을 검색하여 슬랙 체크테이블을 생성한다. 또한, 주기 태스크를 서비스하는 동안 비주기 태스크가 발생하면 그 시점에서 각 프로세서의 슬랙 체크테이블을 참조하여 할당 가능한 프로세서에 비주기 태스크를 할당한다. 【표 1】은 주기 태스크 세트의 예로 주기 태스크의 실행시간과 주기에 대한 정보를 나타낸다.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
실행시간	2	1	3	1	5	6	1
주기	5	10	8	4	9	16	13

【표 1】 주기 태스크 세트

주어진 태스크들을 주기가 작은 것부터 재정렬하여 RM 스케줄링 알고리즘으로 각각의 프로세서를 【식 1】에 의해 태스크 스케줄링 가능성을 검사하여 스케줄이 가능한 경우 그 프로세서에 각 태스크를 할당한다. 그에 대한 결과는 【그림 1】과 같다.

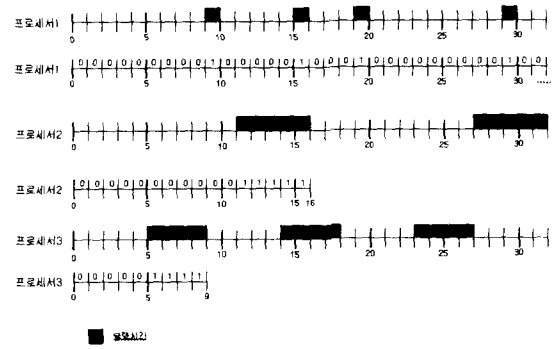
$$\sum_{i=1}^{m} \frac{C_i}{T_i} \le n(2^{1/n}-1) \qquad \text{【식 1】}$$



【그림 1】 주기 태스크의 RMFF 방식의 스케줄링 결과

주기 태스크를 각 프로세서에 할당한 후, 비주기 태스크를 처리하기 위한 슬랙을 계산한다. 프로세서의 슬랙 정보를 테이블로 구축하는 경우 테이블의 크기는 태스크들의 초월주기(hyperperiod) 크기만큼 구성하게 된다. 여기서 초월주기는 프로세서에 할당된 주기 태스크들의 주기의 최소공배수(LCM)가 되는 시간단위를 말한다. 슬랙 체크테이블은 【그림 2】에서와 같이 각 시간단위에 있어서 그 시간단위가 슬랙인지 아닌지에 대한 정보를 가지게 된다.

비주기 태스크가 발생한 시점에서부터 비주기 태스크가 필요로 하는 실행시간만큼의 슬랙이 어느 정도의 시간이 경과한 후인지에 대한 정보를 슬랙 체크테이블을 이용해서 검색한다. 이 때 슬랙 체크테이블은 환형리스트를 이룬다.



【그림 2】 프로세서의 슬랙 체크테이블

【표 2】는 주어진 태스크를 스케줄한 결과, 프로세서 이용률과 초월주기, 초월주기당 슬랙수를 나타낸다.

	프로세서이용률	초월주기	슬랙수 (초월주기당)
프로세서1	0.827	260	45
프로세서2	0.688	16	5
프로세서3	0.556	9	4

【표 2】 프로세서 이용률, 슬랙타임 그리고 슬랙의 수

4. 알고리즘

주기 태스크의 마감시간을 만족하는 범위내에서 비주기 태스크를 서비스하기 위해, 본 논문에서는 다음과 같은 가정을 두고 알고리즘을 설계한다.

- 주기 태스크의 실행시간(Computation time)과 주기(Period)등에 대한 정보는 실행 초기에 이미 알려져 있다.
- 주기 태스크는 경성 마감시간을 갖는다.
- 비주기 태스크는 임의의 시간에 동적으로 도착한다.
- 각 태스크는 서로 독립적이며, 선점적(preemptable)이다.
- 문맥교환(text switching)으로 인한 오버헤드(overhead)는 무시할 수 있을 정도로 작다.
- 각 프로세서는 동등하며, 서로 독립적이다.

【알고리즘 2】에서와 같이 n번째 프로세서부터 첫번째 프로세서까지 비주기 태스크의 마감시간을 만족하면서 태스크의 실행을 종료할 수 있는 프로세서를 슬랙 체크테이블의 정보를 이용해서 검색한 후, 검색된 프로세서에 비주기 태스크를 할당한다. 만약, 마감시간을 만족할 수 없는 경우라 할지라도 각 프로세서 중에서 가장 가까운 시점에 작업을 종료할 수 있는 프로세서에 비주기 태스크를 할당한다.

```
if(비주기 타스크 발생) {
  for(i=n; i>=1; i--) { // 할당 가능한 프로세서 검색
     while(i) {
        if(Pi_SCT_slack == true) {
           전체 실행 단위시간++;
           Pi_슬랙의 수++;
        }
        else
           전체 실행 단위시간++;
        if(Pi_슬랙의 수 == 비주기 태스크의 실행시간)
           break;
        }
     }
  }
}
P_number = Select_minimum_전체 실행 단위시간();
// 가장 적은 실행 단위시간을 갖는 프로세서를 선택

Allocate Aperiodic_task to Process[P_number];
```

【알고리즘 2】 SCT를 이용한 비주기 태스크 스케쥴링 알고리즘

위와 같은 알고리즘을 사용하여 슬랙 체크테이블에서 비주기 태스크를 할당할 수 있다. 따라서, 주기 태스크만 스케쥴링하던 RMFF 스케쥴링 알고리즘을 사용하면서 비주기 태스크도 함께 처리할 수 있도록 슬랙 체크테이블을 이용하여 개선한 비주기 태스크 스케쥴링 알고리즘의 효율성이 기존의 방식보다 더 향상됨을 알 수 있다.

5. 결론 및 향후 연구과제

본 논문에서는 경성 마감시간을 갖는 주기 태스크는 RMFF 스케쥴링 알고리즘을 사용하고, 연성 마감시간을 갖는 비주기 태스크의 효과적인 스케쥴링 알고리즘을 구현하기 위해 슬랙 체크테이블을 이용한 스케쥴링 알고리즘을 제안하였다. 슬랙 스틸링 스케쥴링 알고리즘이 비주기 태스크가 발생한 시점에서 슬랙을 계산하던 것을 개선하여 주기 태스크의 할당 직후에 초월주기만큼의 범위내에서 슬랙의 유무에 대한 정보를 갖는 테이블을 작성하고, 비주기 태스크가 발생하는 시점에서 이 슬랙 체크테이블을 참조하여 비주기 태스크를 스케쥴링한다. 이로써 비주기 태스크가 발생할 때마다 슬랙의 재계산에 소요되는 계산시간의 부담을 감소시킬 수 있고, 보다 짧은 시간 안에 비주기 태스크를 종료할 수 있었다.

향후 연구과제로는 문맥교환에 대한 오버헤드를 고려한 스케쥴링이 가능하도록 해야하며, 멀티프로세서 환경에서 각각의 프로세서가 각 태스크의 연관성을 처리할 수 있도록 상호작용하는 프로세스간의 연결에 대한 연구가 필요하다. 또한 멀티프로세서 환경의 특성을 충분히 살린 스케쥴링 알고리즘에 대한 연구가 더 이루어져야겠다.

참고 문헌

【1】 Clifford W. Mercer, "An Introduction to Real-Time Operating Systems: Scheduling Theory", November 13, 1992.

【2】 N.Audsley and A. Burns, "Real-Time System Scheduling", Predicatably Dependable Computer Systems, Volume2, Chapter2, Part II.

【3】 C.L.Liu and W.Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real Time Environment", Journal of the Association for Computing Machinery, Vol.20, No.1, Jamuary, 1973, pp.46-61.

【4】 Almut Burchard, Jorg liebeherr, Yingfeng Oh, and Sang H. Son, "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems",1994.

【5】 김지용, "멀티프로세서 환경에서 비주기 태스크를 위한 실시간 스케쥴링 기법 연구", 서울대학교 대학원 컴퓨터공학과, 1997.

【6】 한대만, "비주기적 태스크의 효율적 처리를 위해 개선된 슬랙 스틸링 알고리즘", 수원대학교 대학원 전자계산학과, 1999.

【7】 신현식, 김태웅, "실시간 시스템의 개관", 한국정보처리학회, 1998. 7.