제19회 한국정보처리학회 춘계학술발표대회 논문집 제10권 제1호 (2003.5)

경성 비주기 태스크에 대한 확장된 EDL 알고리즘

김현수 김인국
단국대학교 전자계산학과
jordan@dankook.ac.kr

Extended EDL Algorithm for Hard Aperiodic Task

Hyun-Soo Kim In-Guk Kim
Dept. of Computer Science, Dankook Unive

요 약

본 논문은 동적 우선순위 실시간 시스템에서 경성 종료시한을 갖는 비주기 태스크를 스케줄링하는 EDL 알고리즘을 확장하여 구현하였다. 동적 우선순위 방식의 비주기 태스크를 스케줄링 하는데 있어서 최적이라고 증명된 EDL 알고리즘이 갖고 있는 문제점인 실행되고 있는 비주기 태스크가 있으면 다음 비주기 태스크의 요청이 들어왔을때 선행된 비주기 태스크의 종료시점에서 받아들이는 제약을 개선하고 경성 비주기 태스크들이 동시에 들어왔을 때의 응답시간을 시분할 방식을 이용하여 최소화 하였다.

1. 서론

실시간 시스템들은 범용 컴퓨팅 시스템과는 다르게, 실시간 시스템의 정확성과 안전성을 보장하기 위해 반드시 충족되어져야 하는 각 태스크들에 대한 시간적 요구사항들에 중요성을 둔다. 실시간 시스템들은 가능하면 예측이 불가능한 환경에서도 예측 가능하도록 동작해야 하며, 이러한 예측성은 일반적으로 실시간 운영체제에 의해 보장된다. 실시간 시스템은 시간적 제약 특성에 따라 크게 경성 실시간 시스템과 연성 실시간 시스템으로 구분되어질 수 있다. 경성 실시간 시스템은 각 실시간 태스크들이 주어진 종료 시한 내에 반드시 완료해야 되는 시간적 제약의 특징을 갖으며 만약 해당 태스크가 종료 시한 내에 완료되지 못한다면 치명적인 결과를 초래하게 된다. 이에 반해 연성 실시간 시스템에서는 각 태스크들이 종료 시한 내에 완료되지 못하더라도 작업은 지속되지만 수행되지 못한 시간만큼의 비효율적 결과를 초래하게 된다. 실시간 시스템에 존재하는 태스크들은 일정시간마다 도착하여 반복적으로 수행되는 주기 태스크와 도착시간이 정해져 있지 않은 비주기 태스크로 구분할 수 있다. 일반적으로 주기 태스크들은 경성 종료시한을 가지고 비주기 태스크들은 마감시간에 따라서, 연성 비주기적 태스크 스케줄링과 경성 비주기적 태스크 스케줄링으로 나눌 수 있다. 본 논문은 실시간 시스템에서 태스크를 스케줄링 함에 있어 비주기적으로 발생하는 태스크들의 슬랙 이용률을 최대한 높이고 두 개 이상의 비주기 태스크를 스케줄링 함에 있어 응답시간을 최소화 하는데 그 목적이 있다. 비주기 태스크 스케줄링의 목적은 주기적 태스크의 마감시간을 보장하고 연성 비주기적 태스크의 응답시간을 최소화하는데 있고[1][6] 경성 비주기 태스크 스케줄링의 목적은 주기적 태스크의 마감시간을 보장하고 스케줄링 가능성을 판단하여 경성 비주기적 태스크에 대한 마감시간을 보장할 수 있는데 있다[7]. 여기서 제안된 알고리즘으로 Chetto and Chetto가 제안한 EDL 스케줄

링 방식으로 주기 태스크의 실행 시 생기는 최대한의 슬랙 타임을 계산하여 비주기 태스크에 제공하는 방법이 있다[5]. 이 논문은 이 방식을 따른다.

2. 관련연구

2.1 EDL 알고리즘

Chetto 와 Chetto가 제안한 EDL 알고리즘은 동적 우선순위 기반의 경성 비주기 태스크 스케줄링하는 알고리즘이다. 온라인 상에서 마감시간 사전 할당 기법과 일반 동적 우선 순위 알고리즘을 혼합한 스케줄링 기법으로 모든 주기적 태스크의 마감시간을 엄격히 지키면서도 가능한 최대의 비주기적 태스크 처리를 위한 슬랙이 전체 스케줄링 구간을 통하여 유지된다. 기본 개념은 EDF의 반대되는 개념이고 오프라인에서 주기적인 태스크의 마감시간을 보장하는 범위에서 주기적 태스크의 실행을 가장 뒤로 미룬 결과로 슬랙 타임 테이블(EDL 테이블)을 만든다. 온라인에서 경성 비주기 태스크가 도착하면 주기적 태스크의 실행을 가장 뒤로 미루고 이때 중요한 것은 주기적 태스크의 마감시간을 보장하여야 한다. 주기적 태스크의 주기는 마감시간과 동일하고 비주기적인 태스크에 대한 요청은 선행된 비주기 태스크가 종료된 시점에서 이루어진다고 가정하고 있다. 이 알고리즘은 Rmos-Theul과 Lehoczky가 제안한 슬랙 스틸링 알고리즘보다 슬랙을 계산하는 시간 복잡도가 낮다.

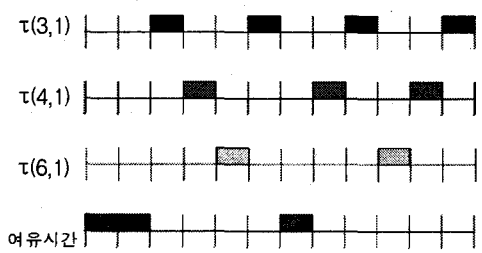

그림 1. EDL 스케줄링된 예

2.2 슬랙 스틸링알고리즘

슬랙은 주기태스크가 실행을 끝마쳤을 때 종료시한까지의 사용되지 않은 부분을 말한다. 슬랙 스틸링 알고리즘에는 정적 슬랙 스틸링 알고리즘과 동적 슬랙 스틸링 알고리즘이 있다. 두 알고리즘은 모든 종류의 비주기 태스크에 대해 최소 응답시간을 제공한다는 관점에서 최적이라고 증명되어 있다[2][4]. 정적 슬랙 스틸링 알고리즘은 연성 종료시한을 갖는 비주기 태스크와 경성 종료시한을 갖는 비주기 태스크에 대해서 적용할 수 있는 알고리즘이다. 이 알고리즘은 주기 태스크에 종료시한 단조형으로 우선순위를 할당하고 비주기 태스크를 위한 주기적인 서버를 두지 않고 비주기 태스크가 도착하면 도착시간에 해당하는 오프라인 시 구해놓은 주기 태스크의 각 우선순위 레벨에서의 가능한 여유시간들의 최소값을 비주기 태스크의 실행에 사용된다. 슬랙 스틸링 알고리즘은 기본적으로 다음과 같은 가정을 하고 있다.

- 문맥교환이나 태스크 스케줄링을 위한 오버헤드는 스케줄링 시 적용되지 않는다.
- 태스크의 준비시간은 주기의 시작과 일치하며, 다른 태스크와의 동기나 자체적인 suspend는 발생하지 않는다.
- 모든 태스크는 선점 가능하다.
- 태스크들간의 충돌은 발생하지 않는다.

슬랙 스틸링 알고리즘에서 주기들의 최소 공배수에 대해서 유휴 시간을 구하는 이유는 최소 공배수만큼의 시간마다 주기 태스크들은 동일한 실행 패턴을 반복하기 때문이다. 주기 태스크 T_n개 가 있을때 주기 태스크들의 최소 공배수 H로 정의되고 오프라인 시에 0부터 H동안의 시간에 대해서 T_i의 여유시간 Ai(t)를 구해 놓는고 이 여유 시간은 A(i,j)라는 이차원 배열 형태로 저장된다. i는 태스크의 우선순위이고 j는 Ti의 j번째 실행을 나타낸다. 비주기 태스크가 s라는 시간에 도착하였다면 비주기 태스크에 할당할 수 있는 서버의 최대 크기인 A^*를 다음의 계산식을 이용하여 구한다.

$$A^* = \min_{1 \le i \le n}(A(i,j) - I_i(s) - A(s))$$

A^*가 비주기 태스크의 계산 요구량보다 작을 때는 A^*만큼 비주기 태스크를 실행시키고 주기 태스크가 끝날 때마다 A^*를 다시 계산한다. 슬랙 스틸링 알고리즘에서 A^*를 동적으로 계산하는 알고리즘의 복잡도는 O(n)이고 구현할 때의 문제점은 A(i,j)테이블의 크기이다. n개의 주기태스크가 있을 때 이 태스크들의 H는 n이 증가할수록 커지는데 A(i,j)테이블의 크기는 m이 $\max_{1 \le i \le n} H/T_i$ 일 때 n*m으로

구할 수 있다.

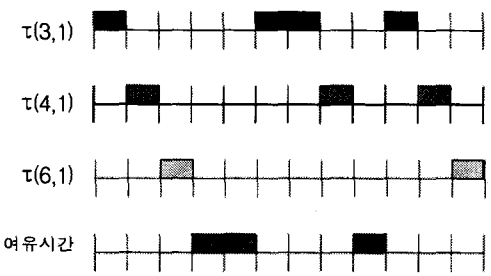

그림 2. 태스크의 여유시간

2.2 확장한 슬랙 알고리즘

경성 비주기 태스크 스케줄링의 목적은 주기적 태스크의 마감시간을 보장하고 경성 비주기 태스크에 대한 마감시간을 보장할 수 있는지의 여부를 태스크 도착 시 스케줄링 가능성을 판단하여 승인하거나 거부하는데 있다. 그러나 경성 비주기 태스크를 스케줄링할 때 선행된 비주기 태스크가 종료되야만 다음 비주기 태스크를 받는 제약이 따른다. 이는 두 번째부터의 비주기 태스크들의 응답시간이 느려지는 결과를 초래하기 때문에 이에 대한 제약을 다음 비주기 태스크도 현재 실행중인 비주기 태스크의 실행시간을 고려하여 나머지 시간안에 스케줄링이 가능하다면 받아들여서 두 태스크의 응답시간을 개선하기 위해 시분할로 처리하는 기법을 제안한다.

```
while(( Tap의 수행시간) !=0)
{
    if(slack_table[current_time] !=0)
    //여유시간이 있다면
    {
       if( Tap가 존재)
          {
          수용검사( );
          시분할스케줄링( );
          }
       else
          {
          수용검사( );
           Tap 수행;
          }
```

```
else
{
    //여유시간이 없다면
       while(slack_table[current_time] == 0)
       delay;
}
```

그림 3. 알고리즘 의사코드

그림2 의 기본적인 의사 코드를 보면 비주기 태스크(T_{ap})의 요청이 왔을 때 EDL 테이블에 기초하여 슬랙 테이블에 최대여유시간이 있으면 먼저 그 슬랙의 크기와 비주기 태스크의 실행시간을 비교하여 받아들인다. 만약 비주기 태스크가 존재를 한다면 먼저 들어온 비주기 태스크의 시작시점에서부터의 실행시간을 뺀 나머지 슬랙과 비교하여 만약 슬랙이 크다면 받아들이고 아니면 제거한다. 받아들여졌을 때는 비주기 태스크를 그 시점에서부터 시분할 스케줄링으로 스케줄링된다. 반대로 여유시간이 없을 경우에는 비주기 태스크를 지연시킨다. 이 경우는 연성 비주기 태스크 스케줄링이 요구하는 응답시간의 최소화는 기대하기 어려우나 경성 비주기 태스크의 목적인 마감시간을 보장하면서 응답시간을 최소화시키는데 그 의의가 있다.

```
수용검사( )
{
   if(slack[MAX] <  Tap_excution_time)
      reject  Tap;
   else
      return  Tap;
}
```

그림 4. 수용검사 알고리즘

그림 4의 수용검사 알고리즘에서는 슬랙의 크기와 경성 비주기 태스크의 실행시간을 비교하게 된다. 만약 슬랙의 크기가 비주기 태스크의 실행시간보다 작게 되면 거절하게 되고 비주기 태스크의 실행시간이 슬랙의 크기보다 작다면 받아들인다. 이때 슬랙의 크기에서 비주기 태스크의 실행시간을 뺀 나머지를 slack[MAX]에 저장하고 비주기 태스크를 스케줄링 하게 된다. 다시 비주기 태스크가 요청이 되었을 때 남아있는 슬랙을 저장하고 있는 slack[MAX]가

다시 비주기 태스크의 실행시간과 비교해서 거절할 건지 받아들일 건지를 반복 수행하게 되는 것이다. 받아들인 비주기 태스크들은 실행시간이 다하는 동안 시분할로 스케줄링 하게 된다. 그림 5의 $T_{ap}1$ 이 실행되고 있을 때 $T_{ap}2$가 들어오면 $T_{ap}2$가 들어온 시점부터 시분할 스케줄링 하게 된다. 시분할 스케줄링 방식으로는 시분할 방식에 효과적인 라운드 로빈 스케줄링을 하게 되고 비주기 태스크 T_{ap} n은 같은 크기의 cpu시간을 할당 받게 된다.

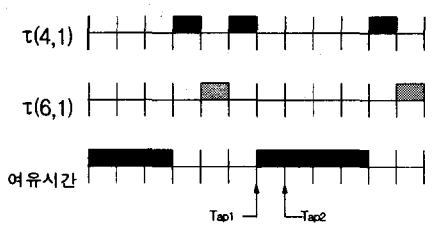

그림 5. 스케줄링 예

3. 결론

실시간 시스템에서 비주기 태스크에 대한 스케줄링은 매우 중요한 문제이다. 비주기 태스크들은 요청하는 시점과 실행 요구량을 미리 알 수 없다. 그래서 경성 비주기 태스크 스케줄링의 목적은 주기적 태스크의 마감시간을 보장하는 범위 내에서 현재까지 요청된 경성 비주기 태스크들을 수용 검사하여 스케줄 가능한 태스크 집합을 생성하는데 그 목적이 있다. 이에 본 논문은 경성 비주기 태스크의 스케줄링 목적을 준수하면서 비주기 태스크가 종료되고 나서 다음 비주기 태스크를 받아들이는 제약을 없앰과 동시에 시분할 기법을 이용하여 비주기 태스크들의 응답시간을 개선하였다. 앞으로는 비주기 태스크들을 EDF방식을 이용하여 스케줄링 한 다음 시분할 방식을 혼합하여 스케줄링 하는 방법도 고려해 볼만하다.

참고문헌

[1] C. L. Liu and J. W. Layland. "Scheduling algorithms for multiprogramming in a hard real-time enviroment." journal of the ACM. Vol.20, pp. 46-61, Jan. 1973

[2] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemtive systems," in Proceedings of the Real-Time Systems Symposium, pp. 110-123, Dec. 1992.

[3] S. Ramos-Thuel and J. P. Lehoczky, "On-line scheduling of hard deadline aperiodic task in fixed-priority systems," in Proceedings of the Real-Time Systems Symposium, pp. 160-171,Dec. 1993.

[4] R. I. Davids, K. W. Tindell, and A. Burns,"Scheduling slack time in fixed-priority preemptive systems," in Proceeding of the Real-Time Systems Symposium, pp. 222-231,Dec. 1993.

[5] H. Chetto and M. Chetto, "Some Reault of the Earliest Deadline Scheduling Algorithm," IEEE Trans. Software Engineering 15 (10) pp. 1261-1269, 1989.

[6] B. Sprunt, L. Sha, and J. Lehoczky,"Aperiodic task scheduling for hard-real-time systems," Journal of Real-Time Systems, 1, July, 1989.

[7] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier, "Preemtively Scheduling Hard-Real-Time Sporadic Tasks on One Processor," Proceedings of the IEEE Real-Time System Symposium, pp. 182-190, Dec 1990.

[8]임 덕주, 박 성한,"최대여유시간 제공 연성 비주기 태스크 스케줄링 알고리즘", 한국정보처리학회 제6권 제2호, 1999.

[9]김 진석, "경성 비주기태스크들을 위한 온라인 스케줄링알고리즘", 석사학위 논문, 강원대학교, 2000.

[10]김 태웅, "고정 우선순위 실시간 시스템에서 비주기 태스크 스케줄링 알고리즘", 석사학위 논문, 서울대학교, 1995.

[11]최 만억, 한 대만, 구 용완, "비주기 태스크의 응답시간을 개선하기 위해 확장한 슬랙 스틸링 알고리즘", 한국정보처리학회 논문지 제7권 제7호, 2000.

[12]이 귀영, "비주기 태스크의 효율적 처리를 위해 이중 우선순위를 갖는 스케줄링 알고리즘", 석사학위 논문, 한국과학기술원(KAIST), 1994.