다중처리기 시스템에서 중앙 스케줄러를 기반으로한 실시간 스케줄링 기법

이경복, 윤인숙, 이재완
군산대학교 정보통신공학과

The Real-time Scheduling Mechanism Based on Central Scheduler in Multiprocessor System

Kyoung-Bok Lee, In-Suk Yun, Jae-Wan Lee
Dept. of Telecommunication Engineering, Kunsan National Univ.

요 약

다중 프로세서 환경에서 태스크들을 할당해주는 중앙프로세서를 두고 Laxity를 기반으로 긴급타스크를 선택하여 스케줄링을 수행한다 중앙 프로세서는 프로세서들의 슬랙시간과 각 프로세서의 Local 큐에 대기하고 있는 비주기적 타스크의 총 수행시간 등의 상태정보를 수집 분석하여, 타스크의 실행시간에 가장 적합한 프로세서를 선택하여 할당한다. 또한 타스크 특성에 따라 주기적 타스크와 비주기적 타스크로 나누고 주기적 타스크는 마감시간을 지키는 범위 내에서 최대한 수행시간을 연기 시켰다 시뮬레이션 결과 Overload(마감시간을 지키지 못하는 타스크)수의 감소와 빠른 응답시간을 제공함을 알수 있었다.

1. 서 론

실시간 시스템은 일반적인 시스템과는 달리 수행의 정확성 뿐만 아니라 일정한 마감시간 내에 실세계에서 발생하는 사건을 처리할 수 있는지가 중요한 척도기 된다[1,2,3]. 따라서 실시간 타스크들에 대해서는 마감시간을 보장하면서 빠른 응답시간을 제공하는 것이 실시간 타스크 스케줄링 알고리즘의 목적이다.

다중 처리기 환경에서 스케줄링하는 방법은 크게 중앙 스케줄링 방법과 분산 스케줄링 방법이 있다. 중앙 스케줄링 방법은 스케줄링을 전담하는 프로세서를 두고 이 프로세서가 나머지 프로세서들의 상태정보를 이용하여 스케줄링 하는 방법이며, 분산 스케줄링 방법에서는 각기의 프로세서가 스케줄링을 담당한다 일반적으로 멀티프로세서 환경에서는 중앙스케줄러 방법이 우수한 것으로 알려져 있다.

중앙 프로세서 스케줄링 알고리즘은 정적 또는 동적으로 분류될 수 있는데, 정적 알고리즘은 사전에 설정된 어떤 규칙에 따라 부하를 할당하고, 반면에 동적 알고리즘은 온라인 상에서 각 프로세서의 시스템 상태정보를 이용하여 가장 적당한 프로세서에게 프로세스를 할당하는 정책을 사용한다[4,5].

각 프로세서의 상태를 수집하는 방법은 Polling 과 Broadcasting 방법으로 크게 나눌 수 있다[5]. 중앙 스케줄러는 고장이 발생하면 다른 프로세서들중 하나가 자동적으로 중앙 스케줄러의 규칙을 떠맡게 된다. 따라서 시스템의 전역상태정보를 수집하고 조절하는데 이주 쉽다는 잇점을 가지고 있다.

본 연구에서는 중앙스케줄링 방법을 사용하며 타스크의 종류를 주기적 타스크와 비주기적 타스크로 구분한다. 주기적 타스크는 정적으로 계산된 주기정보를 이용하여 마감시간을 지키는 범위 내에서 실행시간을 최대한 미루어 수행하므로서, 비주기적 타스크에 빠른 응답시간을 제공할 수 있다 중앙 스케줄러에서 긴급 타스크의 선정방법은 LLA(least laxity algorithm)을 사용하여 긴급한 타스크를 먼저 처리하는 방법을 사용한다[6].

2. 시스템 모델 및 타스크 모델

2.1. 시스템 모델

시스템의 스케줄링 단위는 타스크이며, 중앙 프로세서는 주기적 타스크와 비주기적 타스크를 저장하는 큐를 가지며 나머지 각각의 프로세서들은 자신의 Local 큐를 갖는다

주기적 타스크는 주기정보에 따라 각 프로세서에 할당하며, 비주기적 타스크는 주기적 타스크가 할당되고 남은 슬랙에 할당된다 중앙 스케줄러는 각 프로세서의 상태정보(각 프로세서의 슬랙및 Local 큐에 대기하고 있는 비주기적 타스크의 수행시간)에 따라 타스크를 할당한다.

2.2. 타스크 모델

시스템의 스케줄링 단위는 타스크이며, 일반적으로 타스크는 정수 파라미터 S(타스크의 시작시간), C(수행시간), D(마감시간)으로 구성된다.

시간 t에서 타스크의 긴급성을 나타내는 Laxity는 다음과 같은 식으로 나타낼 수 있다.

- ○ 타스크의 수행시간 : $C(i)$
- ○ 타스크의 마감시간 : $D(i)$
- ○ 타스크의 $Laxity$: $D(i) - C(i)$

Laxity는 타스크의 긴급성을 나타내는 척도이며, Laxity가 0인 타

스크는 즉시 실행되어야 하며 Interrupt 되어서는 않된다. 음수의 Laxity는 마감시간을 만족시키지 못한다는 것을 나타낸다. 여러 개의 Laxity가 0인 타스크가 존재하는 경우에는 수행시간이 짧은 타스크를 먼저 처리해준다

3. 실시간 타스크 스케쥴링 기법

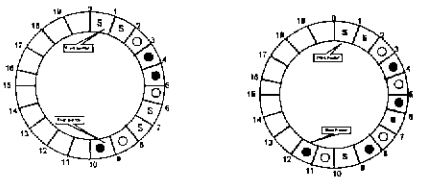
3.1. 중앙프로세서 스케쥴링 기법

중앙 프로세서는 Broadcasting방법을 사용하며 각 프로세서의 진역상태정보를 수집한다 각 프로세서는 요구를 받으면 자신의 슬랙시간 등의 프로세서 상태를 검사하여 중앙 스케쥴리에게 반환하고 중앙 스케쥴러는 이 상태정보를 기반으로 긴급한 비주기적 타스크를 선별하여 타스크의 수행시간과 유사한 슬랙을 가진 프로세서에 타스크를 할당한다 즉, 타스크의 수행시간을 프로세서의 슬랙시간에 최적적합(best-fit)방식에 의해 배치함으로서 스케쥴링 overhead를 줄일 수 있다.

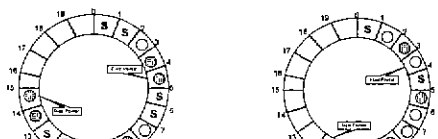
3.1.1. 주기적 타스크 할당

주기적 타스크 모델은 비율 단조형(rate monotonic) 스케쥴링 알고리즘의 모델과 동일하다. 즉, 임의의 주기 타스크 τ_i는 $\tau_i=(C_i, T_i, D_i)$의 속성을 갖는다. ($C(i)$: 타스크의 최악의 수행시간, $T(i)$: 타스크의 주기, $D(i)$: 타스크의 마감시간)

주기적 타스크가 $\tau_1(1,3,3)$, $\tau_2(1,4,4)$, $\tau_3(1,3,3)$, $\tau_4(2,6,6)$, $\tau_5(1,4,4)$, $\tau_6(2,5,5)$, $\tau_7(2,4,4)$, $\tau_8(1,3,3)$ 일 때, 프로세서 ①에는 τ_1와 τ_2, 프로세서 ②에는 τ_3와 τ_4 , 프로세서 ③에는 τ_5와 τ_6, 프로세서 ④에는 τ_7, τ_8이 할당될 경우 각 프로세서의 상태는 그림 1와 같다



① , $\tau_1(1,3,3)$ $\tau_2(1,4,4)$ ② $\tau_3(1,3,3)$, $\tau_4(2,6,6)$



③: $\tau_5(1,4,4)$, $\tau_6(2,5,5)$ ④: $\tau_7(2,4,4)$, $\tau_8(1,3,3)$

그림 1. 프로세서의 상태

3.1.2. 비주기적 타스크 할당

비주기적 타스크(τ_{ap})는 다음과 같은 특징을 갖는다.

$\tau_{ap}=(c_{ap}, d_{ap})$

a_{ap} : 비주기적 타스크의 도착시간

d_{ap} : 비주기적 타스크의 마감시간

c_{ap} 비주기적 타스크의 실행시간

e_{ap} : 비주기적 타스크의 실행을 마치는 시간

타스크의 응답시간 : $r_{ap}=e_{ap}-a_{ap}$

타스크의 긴급성 : $Laxity(t)=d_{ap}(t)-c_{ap}(t)$ (t는 임의의 시점)

중앙 프로세서는 자신의 큐에 도착한 비주기적 타스크의 Laxity를 구하고 각 프로세서에 상태정보를 요청하면 각 프로세서는 상태정보 요구에 따라 자기 상태정보를 중앙프로세서에 반환하게 된다. 반환된 상태정보를 기초로 중앙프로세서는 할당하려고 하는 비주기적 타스크와 각 프로세서의 슬랙값을 비교하여 다음과 같은 원칙에 따라 프로세스를 할당하게되며 프로세서의 적용 순서는 다음과 같다.

① 할당하려고 하는 비주기적 타스크의 수행시간과 크기가 같은 슬랙을 가진 프로세서에 할당한다.

② 할당하려고 하는 비주기적 타스크의 수행시간과 근사하게 큰 슬랙을 가진 프로세서에 할당한다.

③ 할당하려고 하는 비주기적 타스크의 수행시간과 근사하게 작은 슬랙을 가진 프로세서에 할당한다

비주기적 타스크가 다음과 같이 큐에 존재한다고 가정하자.

$a_1(3,6)$, $a_2(4,8)$, $a_3(1,6)$, $a_4(6,9)$, $a_5(4,10)$, $a_6(2,4)$.

위와 같은 비주기적 타스크는 Laxity 값을 구하여 타스크의 긴급성을 알 수 있다. 프로세서 ①은 t가 0일때, 프로세서 ②는 t가 0일 때, 프로세서 ③은 t가 4일때, 프로세서 ④는 t가 3일때 비주기적 타스크가 도착한다고 가정하자

위 순서를 기준으로 임의의 시점 즉, 비주기적 타스크가 도착하는 시점에서 중앙 프로세서의 상태정보 요구시 프로세서 ①의 슬랙시간은 4이고, 프로세서 ②의 슬랙시간은 3이고, 프로세서 ③의 슬랙시간은 4이고, 프로세서 ④의 슬랙시간이 2인 상태정보를 중앙프로세서에 반환하게된다.

3.2. 각 프로세서의 스케쥴링 기법

주기적 타스크는 미리 정적으로 계산된 주기정보를 이용하여 마감시간지향 사전할당 정책에 따라 오프라인상태에서 고정 우선순위를 바탕으로 마감시간지향 사전할당 테이블(Deadlinewise Preassignment table)을 만든다[7].

주기적 타스크의 마감시간을 보장하기 위한 최소한의 주기적 타스크 할당시간을 제외한 나머지 시간을 슬랙으로 간주하는 낙관적 슬랙 계산(optimistic slack calculation) 방법을 사용하여 $S(t_a, t_b)=t_b-t_a-\sum_{i=a}^{b}\delta_i$ 식으로부터 가용 슬랙을 구할 수 있다

$\delta_i(t)$ = ({ $\delta_i(t)=\{0$ if$(Cr_i(t)-Cp_i(t)<0)$, $Else$ $Cr_i(t)-Cp_i(t)\}$)는 주기적 타스크의 임계타스크 실행시간을 의미한다($Cp_i(t)$: τ_i가 t시간까지 완료된 실행시간의 총합, $Cr_i(t)$:. τ_i가 수행되어야 할 t시간까지 수행되어야할 실행시간)[8]. 즉, $s(t_1, t_2)$는 t_1과 t_2사이에 실제 가용한 슬랙값을 각 타스크의 마감시간을 보장하기 위한 최소한의 주기적 타스크의 할당시간을 제외함으로서 t_1 과 t_2사이에 슬랙을 계산할 수 있다.

범위를 비주기적 타스크가 발생한 지점부터 t_1부터 시간 t_2까지 고려한 이유는 오프라인에서 슬랙리스트를 계산하는데 걸리는 시간이 시스템에 overhead를 요구하기 때문에 한번 계산한 슬랙리스트

를 되도록 긴급한 비주기타스크중 비교적 같은 수행시간을 가지는 비주기적 타스크에 할당하므로서 중앙 스케쥴러의 Laxity 검사 횟수를 줄여서 시스템의 overhead를 줄일 수 있기 때문이다.

중앙 스케쥴러는 각 프로세서의 슬랙시간과 할당하려는 비주기적 타스크의 수행시간을 비교하여 되도록 같은 크기의 슬랙을 가지고 있는 프로세서에 타스크를 할당하며 그림 3과 같다.

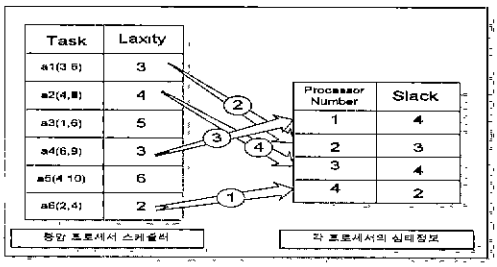


그림 3. 비주기 타스크 할당 테이블

슬랙시간과 근사한 크기의 수행시간을 가진 타스크를 할당 함으로써 비주기적 타스크의 빠른 수행시간을 얻을 수 있다.

4. 성능평가

비주기적 타스크의 도착시간 간격을 지수분표 5, 10, 15, 20으로 변화시켜 시스템 부하에 따라 시뮬레이션 하였다. 타스크를 임의의 프로세서에게 할당하는 방법과 각 프로세서의 슬랙 시간과 현재 할당하려고하는 비주기적 타스크의 수행시간 등의 정보를 이용하여 중앙프로세서에서 스케쥴링하는 제안한 기법을 시뮬레이션을 통해 성능을 비교 평가하였다. 시뮬레이션 결과는 그림 4에서 나타난 바와 같이 평균 응답시간이 약 20%정도 감소함을 알 수 있었고, 또한 부하가 증가하여도 성능이 안정됨을 알 수 있었다. 또한 그림 5에서 나타난 보와같이 Overload를 13%정도 줄일 수 있음을 알 수 있었다.

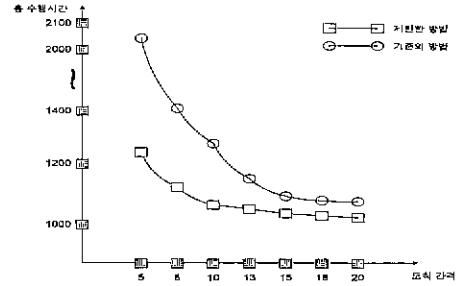


그림 4. 평균응답시간비교

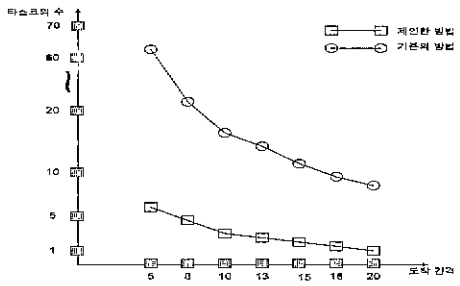


그림 5. Overload 비교

5. 결 론

본 연구에서는 다중처리기 환경에서 중앙 스케쥴링 방법을 사용하여 실시간 타스크들의 스케쥴링 기법을 제시하였다. 타스크의 종류를 주기적 타스크와 비주기적 타스크로 구분하였고 주기적 타스크는 마감시간을 지키는 것에 중점을 두었으며, 비주기적 타스크는 빠른 응답시간을 제공하도록 하였다. 주기적 타스크는 정적으로 계산된 주기 정보를 이용하여 마감시간을 지키는 범위 내에서 실행시간을 최대한 미루어 수행하므로서 비주기적 타스크에 빠른 응답시간을 제공하도록 하였다. 중앙 스케쥴러에서 긴급 타스크의 선정방법은 LLA(Least Laxity Algorithm)을 사용하여 긴급한 타스크를 먼저 처리하는 방법을 사용하였다.

시뮬레이션 결과 평균응답시간이 약 20%, Overload는 13%정도 향상됨을 알 수 있었고, 또한 부하가 증가하여도 성능이 안정됨을 알 수 있었다

참고문헌

[1] 임순영, 이재완, "분산 시스템에서 멀티미디어 지원을 위한 실시간 처리 기법" '97 한국정보학회 호남·제주지부 학술발표 논문집, vol 9. No. 1, pp. 154-159, Jul. 1997.

[2] J.P.Lehoczky and S. Ramos-Thuel, " An Optimal Algorithm for Scheduling Soft-Aperiodic Task in Fixed-Priority Preemptive System", Proceeding of the IEEE Real-Time System Symposium, pp.110-123, Dec. 1992.

[3] John A. Stankovic, "The Integraton of Scheduling and Fault Tolerance in Real-Time System", Proceeding of COINS Technical Report 92-49, Mar. 1992

[4] 유승화 "동일 종류의 다중 프로세서 시스템에대한 동적작업할당 알고리즘", 한국정보과학회 논문집 Vol. 16, No.1, pp. 93-101, Jan. 1989

[5] Ousterhout, J. K., "Scheduling techniques for Concurrent Systems", proceedings of the 3rd International Conference on Distributed Computing System", IEEE, New York, pp. 22-30.

[6] Michael L. Dertouzos and Aloysius Ka-lau Mok, "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks", IEEE Trans. on Software Engineering, Vol. 15, No. 12, pp. 1497-1506, Dec. 1989.

[7] H.I. Kim, S.Y Lee, J.W. Lee, and J.S Kim. "An Aperiodic Task Scheduling Algorithm by Hybrid Priority Assignment in Real-Time Environments", Journal of the Korea Information Science Society, Vol. 22. No.5, pp. 748-758, May, 1995

[8] 이승룡, 이종원, "선택적 우선순위 알고리즘: 가변 우선순위 시스템에서 비주기적 태스크 스케쥴링", 한국정보과학회, Vol. 23, No. 7, pp. 709-717, Jul. 1996.