멀티미디어 응용을 위한 요구비율 기반 프로세서 예약 기법*

박 영 일*, 하 란**
*홍익대학교 전자계산학과
**홍익대학교 컴퓨터공학과
{yipark, rhanha}@cs.hongik.ac.kr

Rate-based Processor Reservation Technique for Multimedia Applications*

Young-il Park*, Rhan Ha**
*Dept. of Computer Science, HongIk University
**Dept. of Computer Engineering, HongIk University

요 약

멀티미디어 태스크는 기존 범용 운영 체제의 시분할 스케줄러에서 만족시킬 수 없는 시간적 요구사항을 가진다. 이런 태스크를 기존의 시분할 태스크와 함께 서비스하기 위해서는 새로운 스케줄링 프레임워크가 필요하다. FQ(Fair Queueing)은 태스크의 공유비율에 비례하여 자원을 할당하는 방법으로 이질적인 태스크(멀티미디어 태스크, 일반 시분할 태스크)가 공존하는 개방적인 환경에서의 스케줄링 정책으로 적합하다는 특징이 있다. 본 논문에서는 FQ의 종류인 WFQ(Weighted Fair Queueing)를 개선하여 하나의 스케줄러에서 다른 두 부류의 태스크를 모두 처리하는 요구비율 기반의 프로세서 예약 기법을 제안한다. 실시간 태스크와 시분할 태스크를 처리하기 위해서 실시간 부류의 태스크를 우선적으로 배치하고, 실시간 부류 태스크의 실행 사이에 시분할 태스크를 스케줄하여 실시간 태스크에 대해서 보장된 서비스를, 시분할 태스크에 대해서는 이 태스크에 할당된 예약만큼의 프로세서 시간을 제공한다. 모의 실험에서는 제안한 프로세서 예약 방식이 실시간 태스크와 시분할 태스크를 효율적으로 처리하며 기존의 WFQ보다 더 안정적임을 보인다.

1. 서 론

디지털 비디오와 오디오 같은 멀티미디어 응용의 필요성과 중요성은 나날이 커지고 있지만 기존의 스케줄 프레임워크는 이런 응용이 가지는 태스크의 시간적 제약을 만족시키지 못한다. 기존의 실시간 시스템에 멀티미디어 응용을 수행하여도 다양하게 변하는 사용자의 요구에 의해 성능의 감소를 초래할 수 있다. 이것은 멀티미디어 태스크와 기존의 태스크가 동시에 수행될 때 시스템이 갖는 특성에 대한 정확한 분석 없이 단지 멀티미디어 태스크의 시간적 요구사항만 만족시켜 주면 시스템이 제대로 동작할 것이라는 가정을 하고 있기 때문이다. 또한 멀티미디어 태스크는 다양한 프로세서 대역폭을 가진다. 예를 들면 MPEG 비디오 데이터의 경우 I, B, P로 구성된 각 프레임에 대해서 각기 다른 압축 기법으로 처리하기 때문이다. 그러나 이러한 프로세서 대역폭은 미리 측정하는 것이 불가능하기 때문에 단순히 기존의 프로세서 예약 알고리즘을 그대로 적용하는 것은 문제가 된다. 따라서 실시간 태스크와 시분할 태스크의 이질적 특성의 태스크를 한 시스템 내에서 처리할 수 있도록 하기 위해서는 기존에 제안된 프로세서 예약 기법을 개선하지 않으면 안된다. 기존에 제안된 예약 기법은 크게 RM(Rate Monotonic), EDF(Earliest Deadline First) 및 그 변형과 Fair Queueing을 이용한 방법이 있다. 그러나 RM이나 EDF의 경우 태스크의 주기, 실행시간, 발생 시간 등을 사전에 요구하기 때문에 프로세서의 요구 대역 측정이 불가능한 멀티미디어 태스크에는 적합하지 않다. 그러나 Fair Queueing의 경우, 자원의 공정한 분배를 통한 대역폭 보장이 가능하기 때문에 이질적인 태스크(멀티미디어 태스크, 일반 시분할 태스크)가 공존하는 개방된 환경의 프로세서 예약을 위해 적합하다는 특징이 있다.

본 논문에서는 Fair Queueing 방법을 개선하여 태스크간의 공정함과 시간적인 보장을 함께 제공할 수 있는 요구비율에 기반한 프로세서 예약 기법을 제안한다. 요구비율을 통해서 각 태스크간 자원의 공정한 분배를 보장하며 프로세서 예약을 통해서 시간 제약을 만족하도록 보장한다.

논문의 구성은 다음과 같다. 2장에서는 기존에 제안된 관련 연구에 관하여 살펴보고, 3장에서는 본 논문에서 제안하는 요구 비율에 기반한 프로세서 예약 기법에 대해서 설명한다. 4장에서는 모의 실험 결과를 이용하여 제안한 기법의 성능을 분석하고, 5장에서는 결론과 향후 연구 과제에 대해 논한다.

2. 관련 연구

최근에 그 중요성이 증대되고 있는 멀티미디어 응용과 기존의 시분할 응용을 동시에 지원하기 위해서 많은 자원 관리 알고리즘이 제시되었다. 이런 알고리즘은 기본적으로 WFQ와 계층적인 스케줄링의 두 부류로 나눌 수가 있다.

2.1 WFQ(Weighted Fair Queueing)

최근의 Fair Queueing에 관련된 연구들은 추가적인 오버헤드(태스크에 대한 실행 시간의 모니터링)를 통해서 좀 더 정확한 비례 요구량을 얻는 데 초점을 맞추고 있다[1]. FQ으로 대표적인 것이 WFQ가 있다. WFQ는 프로세서를 할당받으려는 태스크중에서, 다음 식(1)의 계산에서 나온 VFT(Virtual Finish Time)이 가장 작은 것을 선택하고 이렇

* 본 연구는 정보통신연구진흥원(과제번호 : C1-1999-1234-00), 한국과학재단(과제번호 : KOSEF97-0102-05-01 3), 교육부(과제번호 : BK991031001)의 후원으로 연구되었습니다.

게 선택된 태스크에 프로세서를 할당하는 스케줄 방식이다[2].

$$F_n = \max\{real-time,\ F_{n-1}\} + \frac{l}{r} \quad (1)$$

Waldspurger, Weihl은 프로세서 스케줄링에 WFQ기법을 적용한, stride 스케줄링이라 불리는 비례 공유 알고리즘을 제시하였다[3]. stride 스케줄링에서 각 태스크는 stride와 각 태스크의 진행 과정을 측정하기 위한 pass를 사용하여 프로세서를 할당받는다. 이 알고리즘은 WFQ와 비슷하게 가장 작은 pass값을 가진 태스크를 선택하여 프로세서를 할당한다.

그러나, WFQ를 이용한 스케줄은 태스크간의 비례 공유를 이룰 수가 있지만, 여러 상이한 특성을 가지는 태스크가 함께 있는 경우 시스템 성능의 감소를 초래할 수 있다. 다음 표1과 같은 태스크의 집합이 있는 경우, (가정: Job1은 시분할 태스크, Job2는 실시간 태스크, 실행 시간 단위는 1)

	비율	태스크의 수
Job1	0.7	3
Job2	0.3	1

표 1 태스크 집합

위에 나온 태스크 집합에 대한 VFT는 다음과 같다.

	태스크1	태스크2	태스크3
Job1의 VFT	10/7	20/7	30/7
Job2의 VFT	10/3		

표 2 VFT 테이블

그림1은 표2의 VFT에 의한 프로세서 이용 순서를 나타낸다.

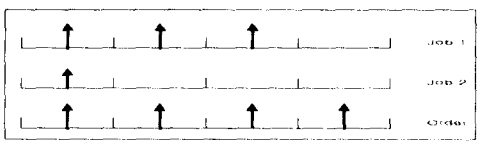

그림 2 프로세서 이용 순서

그림1에서 보듯이 Job2는 Job1의 2개의 태스크가 스케줄 된 뒤에 서비스를 받게 된다. 일반적인 WFQ를 이용한 스케줄 방법에서는 실시간 태스크의 시간적 제약 조건을 만족하지 못하는 경우가 빈번하게 발생할 수 있음을 알 수 있다. 즉, 응용 특성에 따라서 실시간 태스크와 시분할 태스크는 스케줄 되는 방법이 달라야 함에도 불구하고, WFQ에서는 두 부류의 태스크를 분리된 것이 아닌 하나의 부류관점에서 보기 때문에 적절하지 못하게 된다.

2.2 계층적 스케줄링 방식

실시간 응용과 시분할 응용을 하나의 스케줄러에서 모두 처리하는 것은 어렵기 때문에, 각 응용별로 스케줄러를 유지하도록 하고, 또한 각 응용별 스케줄러를 기본 스케줄러에서 관리하도록 하는 계층적 스케줄 기법이 제안되었다[4, 5].

Goyal, Guo, Vin은 다른 응용 부류사이에서 프로세서의 계층적 분할을 지원하기 위한 Start-Time Fair Queueing(SFQ)를 제안하였다[4]. 여기에서는 각기 다른 응용 부류에 다른 스케줄러를 할당하고, 각 부류가 독립적으로 수행되도록 하는 방법을 취하였다.

Deng, Liu, Sun은 2단계 스케줄러를 제안하였다[5]. 즉, 각 응용 부류마다 고유의 스케줄러를 가지고 있고, 이 부류에는 일정한 효율이 할당이 된다. 그리고, 그 하부에 EDF를 사용하는 OS 스케줄러를 사용하여, 상위 응용 부류 스케줄러에서 선택된 태스크를 서비스하게 된다.

이 접근 방법은 실시간 태스크와 시분할 태스크, 각각에 대해서 독립된 스케줄러를 가진다. 하부에는 각각의 부류를 어떤 정책을 통해서 조절해야 하는데, 이때 사용되는 정책으로 우선 순위, 혹은 비례 공유를 들 수 있다.

대부분의 계층적 스케줄러에서는 실시간 부류에 속하는 응용의 우선 순위를 시분할 부류에 있는 응용의 우선 순위보다 높게 정하기 때문에, 실시간 태스크의 마감 시간이 많이 남아 있더라도 시분할 태스크는 기아 상태에 이를 수 있는 단점이 있다[1].

비례 공유를 사용하였을 경우, 각 부류 별로 독립적인 스케줄러를 사용하게 되나, 기본 스케줄러는 WFQ에 의해서 동작하게 된다. 즉, 태스크에 대한 스케줄에서 태스크 부류에 대한 스케줄로 바뀐 것이라고 볼 수 있다. 그렇기 때문에 앞에서 제기한 WFQ가 가질 수 있는 문제를 계층적 스케줄링 방식에서도 갖는다.

3. 요구비율에 기반한 프로세서 예약 기법

이 장에서는 WFQ와 계층적 스케줄 방식을 실시간 태스크와 시분할 태스크가 공존하는 범용 시스템에 적용하였을 때, WFQ나 계층적 스케줄을 통해서 두 부류의 태스크를 서비스하는 것에는 어느 정도 한계가 있는 것을 2장에서 보았다. 이에 비해서, 본 논문에서 제안하는 프로세서 예약 기법은 실시간 태스크에 대해서는 보장된 서비스를, 시분할 태스크에 대해서는 이 태스크에 할당된 예약 비율만큼의 처리 시간을 제공한다.

제안하는 방법은 프로세서를 예약하기 위해서 WFQ기법을 이용한 것으로 다음과 같은 수식에 의해서 스케줄을 하게 된다.

$$F_n = F_{n-1} + RT_{Quantum} \times Ratio + TS_{Quantum} / Nmber\ of\ RT\ tasks \quad (2)$$

$$Execution\ Time = Quantum \times Ratio \quad (3)$$

식(2)에서 우변의 처음 항은 이전 태스크가 끝나는 시간을 얻어내기 위해서, 두 번째 항은 각 태스크에 할당된 비율에 맞는 처리 시간을 계산하기 위해서, 세 번째 항은 시분할 태스크의 프로세서 할당량을 계산하기 위해서 필요하다. 식(3)은 실제 태스크가 수행을 하기 위해서 프로세서를 예약하는 시간을 나타낸다. 여기서 할당량(Quantum)은 미리 주어지게 되고, 할당량이 끝날 때마다 다시 예약 시간이 계산된다.

일정 시간 할당량이 100ms이고, 실시간 부류에 0.7, 시분할 부류에 0.3이 할당된다면, 실시간 부류에서 사용하는 시간은 70ms이고, 시분할 부류에서 사용하는 시간은 30ms이다. 실시간 부류에는 A, B, C가 있고, 각각은 3:3:1의 ratio를 가지고 있다.

위의 예에 따른 VFT에 대한 표와 서비스 순서를 표3과 그림2에 나타내었다.

real-time	A	B	C
0	40	0	0
30	TS	TS	TS
40	40	80	0
70	TS	TS	TS
80	40	80	100
90	TS	TS	TS
100	140	80	100

표 3 VFT 테이블

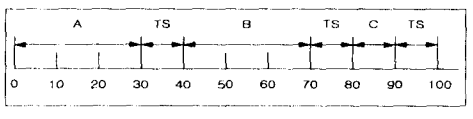

그림 3 서비스 순서

제안한 방법은, 시분할 태스크에 대해서는 엄격한 시간적인 제약조건을 지키지 않아도 되기 때문에 스케줄에 포함시키지 않고 있다. 다만,

이 시분할 태스크에 주어진 예약율은 지켜져야 하기 때문에, 각 실시간 태스크의 VFT에 이를 반영하도록 하였다. 예를 들어, 위의 그림에서 태스크 A의 VFT를 기존의 방법으로 계산을 하게 되면 30이 되지만, 여기에 시분할 부류가 취해야할 처리 시간을 더하고(VFT는 40이 됨), 실제 태스크 A의 수행시간은 40이 아닌 30까지로 하고, 30에서 40사이는 시분할 태스크가 수행을 하도록 한다. 이런 식으로 스케줄을 하게 되면, 실시간 부류와 시분할 부류가 하나의 스케줄러에서 서비스 될 수 있다.

즉, 실시간 태스크와 시분할 태스크를 스케줄하기 위해서 시분할 부류에서는 따로 다른 스케줄러를 사용하지 않고, 실시간 응용의 실행 시간이 끝날 때 시분할 태스크를 수행하도록 하는 방법이다.

위의 예에서, 만일 [30, 40]사이에 시분할 태스크가 없을 경우, 프로세서는 아무런 작업을 하지 않게 된다. 이는 실시간 태스크의 시간적 요구사항을 만족시키려는 목적이지만, 시스템 효율 측면에서 보면 큰 낭비이다. 이런 낭비를 피하기 위해서 빈 프로세서 시간을 실시간 태스크가 이용하도록 하는데, 다음과 같은 방법을 따른다.

*** Lottery 스케줄[3]과 같이 빈 시간에 대해서, 각 태스크가 주어진 비례에 맞게 티켓을 받은 후 임의대로 선택한다.**

또한, 실시간 태스크의 수행시간이 가변일 경우에도 시간이 생길 수 있는데, 이는 프로세서 예약 시간을 계산하거나, 빈 시간을 할당받는 태스크를 선택하는 데 이용할 수 있다.

4. 실험 결과

실험은 Solaris2.5상에서 하였고, 커널과의 통신을 줄이기 위해서, 사용자 수준에서 시스템 호출을 거의 배제한 상태에서 프로그램을 작성하였다.

다음은 제안한 방법에 대한 스케줄 결과를 보여주는 것으로 여기에 나오는 태스크 집합은 앞의 예와 동일하다. 그림3은 100ms동안 각 태스크가 받는 프로세서 시간을 보여주고 있는데, 처음에 주어진 비율에 맞게 프로세서를 받음을 알 수 있다. 그림4는 시분할 태스크가 없는 경우, 이 시분할 태스크에 할당된 프로세서 시간을 실시간 태스크가 비율에 맞게 나누어 가지는 것을 보여주는 그림이다. 태스크 A, B와 C의 비율이 처음에는 불안정하다가 시간이 갈수록 3:3:1로 접근함을 볼 수

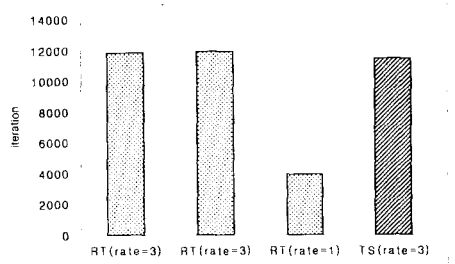

그림 3 스케줄링 결과

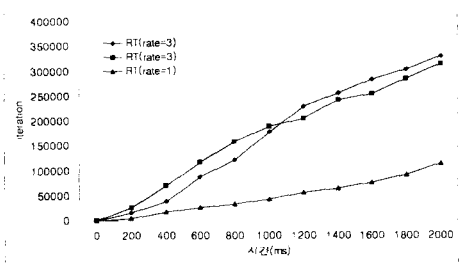

그림 4 시분할 태스크가 없는 경우

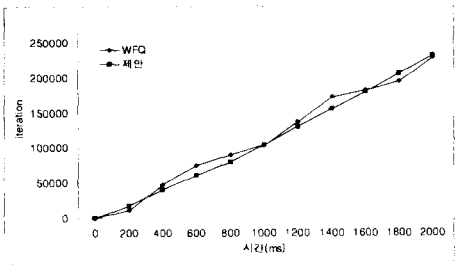

그림 5 WFQ와의 비교

있다. 그림5는 WFQ에 대한 스케줄러를 작성한 후, 임의의 한 태스크에 대해서 제안한 방법과 비교한 그림이다. 그림5에서 두 스케줄러가 2000ms까지 받는 프로세서 시간은 같은 것을 볼 수가 있다. WFQ는 스케줄의 특성상 할당받는 프로세서 시간이 불규칙적인 것을 볼 수가 있다. 실시간 태스크는 자신이 받는 프로세서 시간이 예측 가능해야 하는데, WFQ를 이용해서 실시간 태스크를 스케줄하게 되면 시간당 받는 프로세서 대역폭이 일정하지 않게 된다. 즉, 프로세서 사용시간은 일정하지만, 어느 일부 시간사이에 적거나 혹은 많은 프로세서 시간을 할당받게 된다. 제안한 방법은 시간에 따라서 선형적으로 프로세서를 할당받는데, 이는 실시간 태스크의 예측가능한 수행을 보장할 수 있다.

5. 결론

본 논문에서는 하나의 스케줄러에서 실시간 태스크와 시분할 태스크를 함께 처리하기 위해서 요구비율에 기반한 프로세서 예약기법을 제안하였다. 제안한 방법은 실시간 태스크에 대해서는 보장된 서비스를, 시분할 태스크에 대해서는 이 태스크에 할당된 예약만큼의 프로세서 시간을 제공한다. 모의실험을 통해서 제안한 방법이 WFQ보다 실시간 태스크에 대해 더 높은 안정성을 제공함을 볼 수 있었다

참고 문헌

[1] J. Nieh, M. S. Lam, "The Design, Implementation and Evaluatio of SMART: A Scheduler for multimedia Applications", Proc. c the 16th ACM symp. on Operating System Principles, Octobe 1997.

[2] J.C.R. Bennett, H. Zhang, WF2Q: Worst-case Fair Weighted Fa Queueing", In Proc. of IEEE INFOCOM'96, San Francisco, CA March 1996.

[3] C. A. Waldspurger , "Lottery and Stride Scheduling: Flexib Proportional-Share Resource Management", Ph.D. thesis, MI 1995.

[4] P. Goyal, X. Guo and H. M. Vin, "A Hierarchical CPU Schedul for multimedia Operating Systems", Proc. of the 2nd OSI symp., October 1996.

[5] Z. Deng, J. W.S. Liu, and J. Sun, "A Scheme for Schedulir Hard Real-Time Applications in Open System Environmen Proc. of the 9th Euromicro Workshop on Real-Time Systen June 1997.