멀티미디어 프로세스를 위한 개선된 EDF 스케줄링 방법

이정훈 Felix M Villarreal
서강대학교 전자계산학과

An Enhanced EDF Scheduling Method For the Multimedia Process

Junghoon Lee Felix M. Villarreal
Dept of Computer Science Sogang University

요 약

본 논문에서는 BSD 운영체제에서 마감시간 (dead line)의 개념을 도입하여 멀티미디어 프로세스의 우선순위를 관리하고 실시간 준비 큐를 관리하는 방법에 대하여 알아보았다 마감시간이 가장 먼저 임박한 프로세스에 대하여 높은 우선순위를 할당하는 EDF(Earlist Deadline First) 알고리즘을 바탕으로 구현을 하였다. BSD에서 실시간 프로세스의 준비큐와 시분할 프로세스의 준비 큐 사이에는 프로세스의 이동이 없고 실시간 프로세스는 시분할 프로세스보다 우선적으로 선택되어지게 되어있는데 이로 인해 시분할 프로 세스의 기아(starvation) 현상이 발생할 수가 있다 이를 방지하기 위해 준비 큐 사이에서 프로세스를 이동하는 방법에 대해서도 알아보았다

1. 서 론

Multimedia란 오디오나 동화상, 정지화상, 그래픽, 음성, 애니메이션 등의 여러 미디어가 하나의 응용 프로그램으로 결합된 것을 말한다[5] 이러한 멀티미디어 정보는 기존의 컴퓨터가 다루던 텍스트 기반의 정보와는 다른 특성을 지니고 있다 멀티미디어 정보는 정해진 시간내에 처리되어야만 의미를 갖는 실시간 처리를 요구하며, 한순간에 처리되어야 할 정보량이 기존에 다루던 정보에 비하여 월등히 많으며, 정보저장이 연속적으로 이루어져야 한다. 이러한 멀티미디어 정보를 가지고 있는 데이터중 비디오나 오디오같이 실시간 처리를 요구하는 데이터를 처리하는 데 있어서 기존의 OS가 가지고 있는 문제점이 부각이 되었다. 본 논문에서는 우선적으로 기존의 OS가 가지고 있는 문제점 중 범용 운영체제인 BSD 운영체제에서 CPU 스케줄링의 문제점에 대하여 논하였다. 기존의 CPU 스케줄링에 있어서 문제점은 프로세스의 우선순위를 CPU 사용량에 따라 계산을 하여 CPU를 오래 점유한 프로세스의 우선순위를 낮추기 때문에 CPU 사용량이 많은 멀티미디어 프로세스의 우선순위를 낮추어 실시간 처리를 어렵게 하는 것이다. 실시간 프로세스를 위한 CPU 스케줄링 알고리즘으로 EDF(Earlist Deadline First), RM(Rate Monotonic), LLF(Least Laxity First) 등이 있는데 [5] 그 중에서 최적이며 동적인 EDF 알고리즘을 사용하여 구현하여 보았다 이를 위해 멀티미디어 프로세스의 감시간을 구하고 이를 관리하여 우선순위를 관리하는 새로운 방법을 제시하였다 그리고 실시간 프로세스를 위한 CPU 스케줄링은 시분할 프로세스에 대한 고려가 없기 때문에 시분할 프로세스의 기아(starvation)현상을 방지할 수가 없는데 이를 위해 멀티미디어 프로세스의 여유 시간을 고려하여 멀티미디어 프로세스를 일시적으로 시분할 준비큐에서 관리하여 다른 시분할 프로세스의 수행을 가능하게 하는 방법을 제시하였다.

본 논문에서 구현은 PC에서의 FreeBSD 운영체제의 소스 코드를 수정하여 구현하였고, 모든 실험 또한 소스 코드를 수정하여 실험하였다

2. BSD CPU 스케줄링

CPU 스케줄링은 시스템 내에 존재하는 복수개의 프로세스 중 어느 프로세스를 실행할 것인가를 결정하는 작업으로, 구체적으로 말하면 준비 큐(ready queue)에 있는 프로세스들 중 어떤 프로세스에게 CPU를 할당할 것인 가를 결정하는 작업이다[4] 이 작업은 운영체제의 스케줄러(scheduler)가 담당을 하며 BSD에서는 다단계(multilevel)

피드백 큐(feedback)를 갖는 roundrobin 스케줄러가 담당을 한다. [2]

2.1. BSD CPU 스케줄링을 위한 작업

BSD에서 CPU스케줄링을 위한 작업은 다음과 같이 네가지로 나눌 수 있다.

1 큐(queue)의 관리

2 우선순위(priority)의 계산

3 시간(clock)관리

4 환경교환(context switch)

우선순위를 재계산하는 함수들을 일정한 시간마다 callout 큐에 등록을 하며 이렇게 계산된 우선순위에 입각하여 준비 큐를 재정렬하여 큐의 head에 위치한 프로세스를 떼어내어 자원인 CPU를 할당을 한다. [2]

3. BSD CPU 스케줄링의 문제점

3.1. 실험 환경

1 실험 환경

- CPU : Pentium 200 MHz
- Memory : 64 M
- OS : BSD (FreeBSD 3.0)

2 평가에 사용된 프로그램

- CPU의 사용이 많은 프로그램 (1을 증가하고 감소하는 작업을 130^4번 수행을 하는 작업 작업 1이라 표기)
- I/O 작업이 많은 프로그램 (화일의 읽고, 쓰기를 4000byte의 크기로 1600번을 반복하는 작업. 작업 2라 표기)
- Mpeg video player

3.2. 우선순위의 변화

위의 세 개의 프로그램을 동시에 수행을 시작하였을 때, 우선순위의 변화는 <그림 2>와 같다. <그림 2>에서 보는 바와 같이 CPU의 사용이 많은 Mpegplayer와 작업 1은 우선순위가 계속해서 낮아짐을 알 수 가 있다.

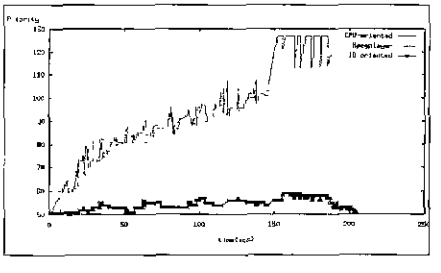

그림 1 우선순위의 변화

3.3. 대기 시간(wating time)

프로세스가 수행되기 전에 준비큐에서 선택되어지기를 기다리는 시간을 대기 시간이라 하는데 실험에서 세 개의 프로세스를 동시에 수행을 시작하였을 때의 대기시간은 다음과 같다.

	momentum	waiting time
mpegplayer	809	21814ms
CPU-oriented job	707	5458ms
IO-oriented job	1429	300ms

표 1. 각 프로세스의 대기시간

결과에서 momentum은 프로세스가 준비큐에 들어간 횟수이다. 3.2와 3.3에서 mpegplay는 10fps 를 요청하였는 데 그 결과로 5.48fps 가 나왔다. 이는 3.2와 3.3에서 알 수 가 있는데, mpegplayer는 계속 낮아지는 우선순위로 인해 CPU점유 횟수기 상대적으로 떨어지며, 이로 인해 준비 큐에서 기다리는 시간이 증가하게 된 것이다.

3.4. BSD 스케줄링의 문제점

실험 결과에 따라서 BSD CPU스케줄링의 문제점은 다음과 같다.

BSD에서 우선순위를 부여하는 방법은 CPU의 사용량이 클수록 우선순위를 감소시키는 것인데 그림 1에서 알 수 있듯이 CPU의 사용할 때간이 큰 mpegplayer의 우선순위가 급격히 감소함을 알 수가 있다. 이는 멀티미디어 프로세스의 요청을 어기는(10fps의 요청에 5.48fps) 결과를 초래한다. CPU 점유횟수가 줄어들게 되면서 대기시간이 길어지는 것이다.

뿐만 아니라 시스템이 과부하 된 경우에 새로운 프로세스의 유입을 차단하는 제어의 기법을 가지고 있지 않는 것도 문제점이다.

4. 제안하는 CPU스케줄링

제안하는 CPU스케줄링의 방법은 다음과 같이 세부분으로 나눌 수 있다.

1. 실시간 큐의 관리

2 멀티미디어 프로세스간에 EDF알고리즘으로 우선순위 관리

3. 멀티미디어 프로세스의 여유시간에 따라 실시간 큐와 시분할 큐사이의 이동

4.1. 실시간 큐의 관리

BSD에서는 실시간 큐를 제공을 하는데 실시간 큐는 시분할 큐보다 context switch하는데 있어서 우선적으로 처리를 한다.[4] 이는 BSD에서 제공하는 rtprio() 시스템 호출에 의해서 가능하며 응용 프로그램 수준에서 이 시스템 호출을 해주어야 한다. 본 논문에서는 mpegplayer가 수행을 시작할 때 이 시스템 호출을 하여 실시간 프로세스로서 관리를 하게된다.

4.2. EDF 알고리즘으로 우선순위 관리

EDF알고리즘은 마감시간이 임박한 프로세스를 우선적으로

선택하여 CPU를 할당하게 끔하는 알고리즘이다. [5]

4.2.1. 마감시간(deadline)

멀티미디어 프로세스에 있어서 마감시간은 soft deadline이다 따라서 이 마감시간을 어긴다고 시스템에 치명적인 손상을 가하지는 않는다.[5] 그러나 이 마감시간을 되도록 준수하여 서비스의 질을 높이는 것이 스케줄러의 의무이다

· 멀티미디어 프로세스의 마감시간 구하기

BSD에서 프로세스의 마감시간을 구하는 데 있어서 가장 중점을 두는 부분은 자발적인 환경교환(voluntary context switch)이다 mpegplayer는 2048byte씩 읽기를 수행하여 읽은 내용을 decode하는데 이러한 읽기를 수행할 때 disk I/O가 수행이 되어 프로세스는 자발적인 환경교환을 하여(tsleep의 호출) 수면 큐(sleep queue)에 들어가게 된다. 멀티미디어 프로세스의 한 번의 수행을 준비큐에서 나와 수면 큐로 들어간 뒤 다시 준비큐에서 나와 CPU를 점유할 때까지의 시간으로 하는 것이다 그림 2는 이를 설명하고 있다

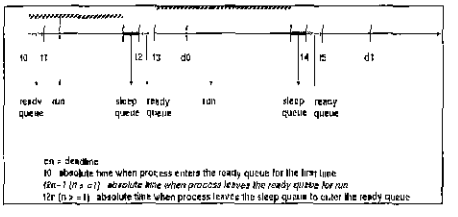

그림 2 프로세스가 준비큐를 나올 때의 고려

위의 그림에서 멀티미디어 프로세스의 주기와 마감시간은 다음과 같은 식으로 구할 수 있다.

$$C_1 = t_2 - t_0 \quad (C_n\text{은 현재 프로세스가 수행된 시간})$$
$$P_1 = P_0 \times \alpha + C_1 \times (1-\alpha)$$
$$d_1 = t_2 + P_1$$

$n \geq 2$ 일 때,

$$C_n = t_{2n-1} - t_{2n-3}$$
$$P_n = P_{n-1} \times \alpha + C_n \times (1-\alpha)$$
$$d_n = t_{2n} + P_n$$

P_n은 등적으로 변하는 주기를 나타내며, d_n은 마감시간을 나타낸다 위의 식에서 α값은 이전 주기에 비중을 두어 0.9의 값으로 하였다. 초기 주기 P_0는 mpegplayer의 단독적인 실험으로 30ms의 값을 구할 수 있었다

4.2.2. 우선순위의 할당

실시간 프로세스의 우선순위는 0부터 31의 값을 갖는데 우선순위를 부여할 때의 경우는 다음과 같이 세부분으로 나뉜다

1 멀티미디어 프로세스가 실시간 준비큐에 없는 경우
2. 모든 우선순위가 다 할당된 경우
3 우선순위가 남아 있는 경우

이 세가지 경우에 대하여 현재 가장 높은 우선순위와 가장 낮은 우선순위를 두고 멀티미디어 프로세스의 마감시간의 값을 저장하여 이 값과의 비교를 통해 우선순위를 할당받게 된다

4.1과 4.2의 관리를 통해 실험한 결과는 다음 표 2와 같다

	기존의 BSD	제안한 BSD
mpegplayer 단독 수행	10126ms 9.99fps	10560ms 9.99fps
CPU job과 같이 수행	25055ms 7.84fps	11125ms 9.99fps

표 2 제안한 스케줄러의 실험 결과

위의 실험에서 451frame을 갖는 mpeg file을 10fps로 요구했을 때의 대기시간과 fps이다.

4.3. 시분할 프로세스에 대한 고려

앞에서 언급했듯이 시분할 프로세스의 기아(starvation)의 발생에 대한 고려로 다음과 같은 방법으로 한다.

4.2.1에서 구한 마감시간을 구하는 식은 프로세스가 한 번의 수행을 마치고 준비큐로 들어갈 때, 구한 주기의 값을 더한 것이다. 프로세스가 마감시간에 이전에 한 번의 수행이 끝났다면 남는 여유시간(laxity)을 축적하여 그 값이 100ms(time quantum)을 넘는 경우, 멀티미디어 프로세스를 시분할 큐로 일시적으로 옮겨 수행을 늦춘다. 이렇게 하면 대기중인 시분할 프로세스는 roundrobin에 의해 적어도 한 번 수행이 되게 된다. 그리고 한 번의 quantum이 지난 뒤 멀티미디어 프로세는 다시 실시간 큐로 이동을 한다

5. 결과

멀티미디어 프로세스는 기존의 운영체제의 CPU 스케줄링 방법으로는 서비스의 질을 만족시킬 수가 없었다. BSD운영체제에서 제공하는 실시간 큐를 이용하여 멀티미디어 프로세를 그 마감시간으로 관리하는 방법은 이러한 서비스의 질을 만족시킨다. 그리고 우선적으로 선택되는 멀티미디어 프로세스가 늘 실시간 큐에 있다면 발생될 수 있는 시분할 프로세스의 기아현상도 멀티미디어 프로세스의 일시적인 큐 옮김으로 방지할 수가 있었다. 단 EDF알고리즘의 구현은 많은 계산량으로 overhead를 초래한다

6. 참고 문헌

[1]Henrick Vestergaard Draboe, David Greenman, FreeBSD 3.0 source code,1998

[2]Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, John S.Quarterman, The Design and Implementation of the 4.4BSD Operating System, Addison Wesley, 1996

[3] Lynne Greer Jolitz, Willeam Frederick Jolitz, The Basic Kernel Source code secret, Peer-to-peer communicatoins San jose, California

[4] William Stalling, Operating System, Prentice-Hall, 1995

[5] Ralf Steinmetz , Klara Nahrstedt, Multimedia : computing communication and applications, Prentice Hall, 1995