멀티미디어 태스크 지원을 위한 다단계 스케줄러

고 영 웅†

요 약

연성 실시간 특성을 가진 멀티미디어 응용 프로그램이 일정한 서비스 품질을 유지하고 수행되기 위해서는 커널 수준과 사용자 수준에서 실시간 처리를 요구한다. 본 연구에서는 널리 사용되는 범용 운영체제 환경에서 멀티미디어를 지원하는데 있어서 문제가 되는 부분을 살펴보고 이를 개선할 수 있는 다단계 스케줄러를 제안한다. 다단계 스케줄러는 사용자 피드백을 기반으로 스케줄링 정보를 생성하는 상위 스케줄러와 커널에 구현된 하위 스케줄러로 구성된다. 제안한 다단계 스케줄러를 리눅스에서 구현하고 성능 분석을 수행하였으며, 실험 결과, 제안한 방식이 시스템에 성능 저하를 발생시키지 않으면서 멀티미디어의 서비스 품질을 일정하게 보장하는 것을 확인하였다.

키워드 : 연성 실시간, 다단계 스케줄링, 멀티미디어

Multi-level Scheduler for Supporting Multimedia Task

Young-Woong Ko†

ABSTRACT

General purpose operating systems are increasingly being used for serving time-sensitive applications. These applications require soft real-time characteristics from the kernel and from other system-level services. In this paper, we explore various operating systems techniques needed to support time-sensitive applications and describe the design of MUSMA(Multi-level Scheduler for Multimedia Application). MUSMA is a framework that combination of user-level top scheduler and kernel-level bottom scheduler. We develope MUSMA in linux environment and it's performance is evaluated. Experiment result shows that it is possible to satisfy the constraints of multimedia in a general purpose operating system without significantly compromising the performance of non-realtime applications.

Key Words : Soft Real-time, Multi-level Scheduler, Multimedia

1. 서 론

멀티미디어 응용 프로그램은 다양한 미디어로 구성되어져 있으며 사용자와 상호 작용을 하는 특성, 대용량 그리고 시간 제약적인 요소를 가지고 있다. 이러한 제약 사항을 만족시키기 위해서 멀티미디어 데이터의 처리시간이나 자원 할당에 관한 문제들은 중요하다. 현재 대부분의 멀티미디어 응용이 범용 운영체제 환경을 전제로 수행이 되므로 범용 운영체제에서 멀티미디어가 수행될 수 있도록 하는 것은 중요하다. 하지만 범용 운영체제는 멀티미디어 태스크를 제한시간 내에 지원함에 있어서 어려운 요소들이 존재한다. 전통적인 범용 운영체제 환경은 시분할 방식의 스케줄러와 요구 페이징(Demand Paging)을 사용하는 가상 메모리 시스템에 가반을 하고 있으므로 멀티미디어가 가지는 연성 실시간 특성을 제공하기에 적합하지 않다. 리눅스, 솔라리스와 같은 범용 운영체제의 스케줄러는 공평성의 원칙에 따라서 프로세서를 많이 사용하고 있는 태스크의 우선순위가 낮아지는 정책을 취하고 있으며, 따라서 일정한 제한 시간 내에 수행되어야 하는 멀티미디어 태스크의 요구 조건을 충족하지 못하고 있다.

최근 들어 다양한 멀티미디어 스케줄러가 개발되었으나, 대부분 실시간 스케줄러의 개념에서 출발하였으며 멀티미디어 응용 프로그램의 특성에 대해서 간과하고 있는 부분이 많다. 예를 들어서 리눅스 시스템에서 동영상이나 그래픽 뷰어와 같은 멀티미디어 응용이 수행되는 경우에, 화면에 영상 및 이미지를 출력시키는 대부분의 작업은 데몬(Daemon)으로 수행되는 X 서버가 전담하게 된다. 멀티미디어 응용 프로그램은 X 서버로 프로세스간 통신(Inter Process Communication) 메커니즘을 통해서 이벤트를 전달하며, X 서버는 전달된 이벤트를 해석한 후에 화면에 그래픽 프리미티브(Primitive)를 출력해준다. 따라서 멀티미디어 응용 프로그램이 제한된 시간 내에 작업이 완료되기 위해서는 X 서버 관련 프로세스의 우선순위도 스케줄링에서 고려되어야

※ 본 연구는 중소기업청의 산학연 공동기술개발 컨소시엄사업에 의해 수행되었음.
† 정 회 원 : 한림대학교 정보통신공학부 조교수
논문접수 : 2005년 3월 8일, 심사완료 : 2005년 7월 29일

한다. 또한 멀티미디어 응용 프로그램은 과도한 시스템 자원을 사용하여 작업을 수행하는 경우가 많다. 프로세스가 수행되는 중간에 과부하가 발생하는 경우, 다양한 프로세스 중에서 일부 프로세스를 희생시킴으로써 중요한 프로세스들이 정상적으로 수행될 수 있도록 해야 한다. 대부분의 멀티미디어 응용이 수행되는 환경에서는 사용자의 판단에 따른 우선순위 조절 및 응용 프로그램의 중요도에 의해서 정책이 결정되는 것이 일반적이다. 즉 멀티미디어 시스템에서는 사용자의 피드백에 의해서 스케줄링 정책에 영향을 줄 수 있는 요소가 다수 존재한다는 것이다.

본 연구에서는 이러한 문제를 해결하기 위한 방법으로 일정한 주기 및 이벤트에 의해서 제어권을 가지는 스케줄러 서버(Scheduler Server)와 커널 스케줄러(Kernel Scheduler)의 다단계 스케줄러(Multi-level Scheduler) 구조를 제안하였으며, 다단계 스케줄링 기법을 통하여 멀티미디어 응용이 다양한 시스템 환경에서 효과적으로 동작할 수 있음을 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존에 제안되었던 멀티미디어 스케줄러에 대해서 기술하였고, 3장에서는 멀티미디어를 지원함에 있어서 문제가 되고 있는 내용에 대해서 설명하였다. 그리고 4장에서는 본 연구에서 제안하는 다단계 스케줄링 알고리즘과 동작원리에 대해서 설명하였다. 5장에서는 제안한 기법에 대한 성능 분석 결과를 보이고, 6장에서 결론을 맺고 향후 연구 과제를 제시하였다.

2. 관련연구

멀티미디어 응용과 같은 연성 실시간 태스크를 위해서 스케줄링 분야에서 연구가 많이 진행되었다. Yavatkar[1]는 멀티미디어 태스크의 수행시간을 예측하기 어렵다는 전제 하에 비율에 기반한 적응적 우선순위 스케줄링(Rate-based Adjustable Priority Scheduling)을 제안하였다. Goyal[2]은 계층적 프로세서 스케줄러 (Hierarchical CPU Scheduler) 방식을 제안하였으며, 이것은 우선순위(Priority)와 비례 할당(Proportional Sharing)을 기본 메커니즘으로 사용하며, 각 스케줄링 정책은 비례 할당에 따라서 일정량의 프로세서 용량을 할당 받고 모든 태스크는 우선순위에 따라서 수행되는 메커니즘이다. Nieh[3]에 의해서 제안된 SMART(Scheduler for Multimedia And Real-Time application)는 솔라리스 운영체제에 구현되었으며, 시간 제약적인 응용 프로그램을 지원하며, 시스템의 현재 부하를 살펴보고 동적인 조정을 통해서 자원을 할당한다. 스케줄링을 결정하는 요소를 중요도(Importance)와 긴급도(Urgency)로 분류하고, 중요도를 제공하기 위해서 우선순위와 WFQ(Weighted Fair Queuing)를 사용하며, 긴급도를 위해서 마감 시간 우선 스케줄링(Earliest Deadline First)을 사용한다.

또한 리눅스 운영체제를 기반으로 실시간 스케줄러에 대한 연구가 활발히 진행되고 있으며, RT-Linux[4]는 리눅스에서 실시간 태스크(특히 경성 실시간 태스크)를 스케줄링 할 수 있는 메커니즘을 제공해 주고 있다. RT-Linux에서 Linux 커널은 실시간 태스크가 동작하지 않을 때 수행되는 낮은 우선순위를 가진 태스크로 여겨지며, 실시간 태스크들은 커널 내부에 모듈로 추가되어진다. 또한 인터럽트가 발생했을 때 나타나는 지연을 소프트웨어 인터럽트 메커니즘을 사용함으로써 해결을 하고 있으며, 실시간 태스크와 사용자 태스크간의 데이터 이동은 실시간 우선순위 큐를 이용해서 한다. 이러한 모델은 기존 시스템에 대한 변경을 최소화시키면서 수백 마이크로세컨드까지의 정밀도로 실시간 태스크를 수행시키는 장점을 가지게 된다. 하지만 커널 내부에 실시간 태스크 모듈을 추가시킴으로써 일관성 유지가 되지 않을 수 있으며, 특히 실시간 태스크가 커널 내부에 들어가야 함으로 사용될 수 있는 범위에 제한을 가져온다. 연속 미디어 태스크의 경우 한 주기에 처리해야 할 작업이 방대하므로 이러한 태스크를 커널 내부의 모듈로 수행하기 어렵다. 특히 사용자 태스크가 이용할 수 있는 라이브러리 등을 전혀 사용하지 못하고 커널 내부의 함수들을 이용해서 작업을 해야 함으로 발생하는 어려움도 있게 된다.

KURT[5]는 RT-Linux와 비슷한 방식을 취하고 있으나 명시적 계획 스케줄러(Explicit Plan Scheduler)를 사용한다는 부분에서 차이가 난다. 즉 응용 프로그램은 스케줄이 발생될 시간을 스케줄 파일(Schedule File)에 명시적으로 지정해주며, 이를 기반으로 각 태스크들이 수행된다. 또한 일반 모드(Normal Mode), 실시간 모드(Real-time Mode)의 두 가지 방식으로 수행 형태를 지정할 수 있으며, 실시간 모드에서는 실시간 프로세스로 지정된 프로세스만이 수행되고 나머지 프로세스는 수행되지 않는다. KURT 스케줄러는 고해상도 타이머(High Resolution Timer)를 수행하기 위해서 독자적으로 UTIME이라는 메커니즘을 사용하고 있다. 이것은 시스템 타이머 모드를 one-shot 모드로 전환시켜서 타이머 인터럽트의 발생시간을 지정함으로써 전체 시스템 클럭의 주파수를 높이지 않고도 높은 해상도의 시간 설정을 할 수 있다. KURT 스케줄러 역시 경성 실시간 작업에 적합하지만, 수행되어야 할 모든 태스크의 스케줄 시간을 지정해야 한다는 단점을 가진다. 특히 멀티미디어 응용 프로그램의 경우 동적으로 스케줄링 되어야 하므로 KURT의 계획에 따른 스케줄링 기법을 적용하기 어렵다는 문제가 있다. SMART-Linux[6]는 SMART 알고리즘을 리눅스로 포팅한 것이며, Linux-SRT[7]는 비율 기반 연성 실시간 스케줄링 정책을 리눅스 커널에 구현하였으며, 대부분의 멀티미디어가 그래픽 기능을 사용하기 위해서 X 서버를 이용하는 문제를 해결하기 위해서 X 서버를 멀티미디어와 우선순위를 같이 높여주는 방식으로 문제를 해결하고 있다. 이외에도 범용적인 운영체제 환경에서 실시간 태스크를 지원하기 위한 연구 성과 등이 다수 있다[8, 9, 10, 11, 12, 13, 14, 15, 16].

3. 기존 시스템의 문제점

범용 시스템에서 수행되는 대부분의 멀티미디어 응용은 일반적으로 X 윈도우 라이브러리를 이용하여 구현되고 있

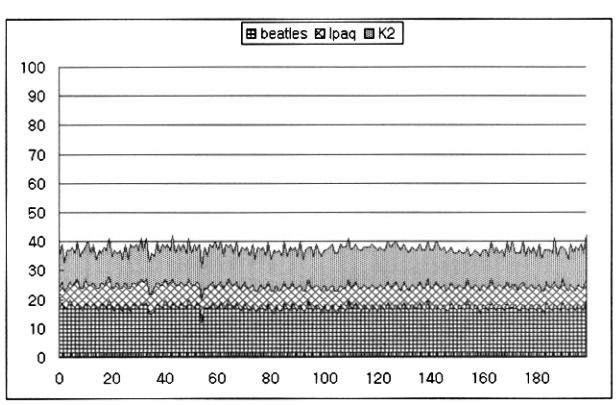
다[17]. 따라서 화면에 출력되는 대부분의 작업은 X 윈도우 시스템을 거쳐서 수행된다. X 윈도우는 클라이언트 서버 구조로 구현되고 있으며, 응용 프로그램은 클라이언트 역할을 하며 X 서버와 서로 통신하며 필요한 요구사항을 전달한다. X 서버는 클라이언트의 화면표시와 서비스 요구에 대한 제어권을 가지며, 그래픽 표시를 담당하는 장치 역할을 한다. 응용 프로그램은 단지 어떻게 서버와 통신할 것인가를 아는 것으로 충분하고 화면 표시장치에 어떻게 정보를 전달할 것인가는 알 필요가 없다. X는 Xlib이라는 저 수준의 클라이언트-서버 통신을 제어하는 라이브러리를 제공하며, 클라이언트는 어떠한 작업을 하기 위해서 Xlib에 포함된 함수를 호출해야 한다. MPEG-1 디코더의 경우도 Xlib을 호출하여 디스플레이를 담당하고 있으므로, X 서버에 계속적으로 필요한 작업을 요청하게 된다. 이러한 요청이 점점 많아질수록 X 서버는 더 많은 프로세서 용량을 차지하게 되는 것이다.

다음은 태스크 관리의 관점에서 범용 운영체제의 문제점을 보이기 위하여, 널리 사용되고 있는 범용 운영체제인 리눅스 상에서 연성 실시간 태스크를 수행시키며 개선되어야 할 사항을 지적하고자 한다. 실험에 사용된 Mpeg-1 데이터 클립은 <표 1>과 같이 세 가지이다.

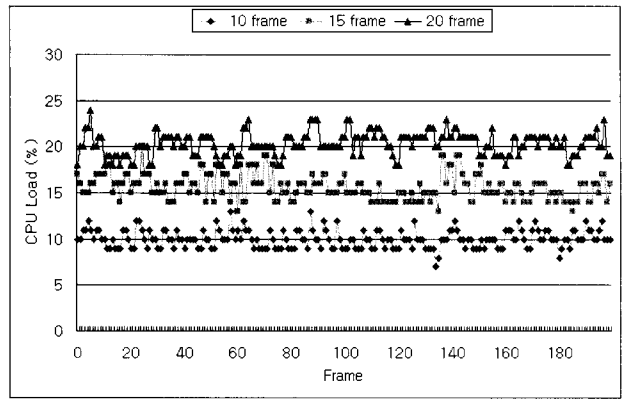
(그림 1)은 주어진 멀티미디어 데이터를 초당 10 프레임 즉 100밀리 세컨드 주기로 수행 시켰을 때의 사용하고 있는 프로세서 용량을 보여 준다. X축은 비디오 데이터의 프레임 수를 의미하며, Y축은 전체 프로세서 용량에 대한 퍼센트를 나타낸다. 즉 각각의 비디오 데이터를 100밀리 세컨드마다 처리해서 화면에 출력할 때 동영상 플레이어가 프로세서를 몇 퍼센트 사용하고 있는지를 나타내고 있다. <표 1>의 데이터에 따르면 전체 프로세스 사용량 중에서 순수하게 멀티미디어 태스크가 차지하는 비율은 이론적으로 (23.2 + 17.8 + 10.3)/100.0 이 되며, 대략 51 퍼센트가 된다. 단, 백그라운드로 수행되는 X 윈도우 데몬이 사용하는 프로세서 용량을 제외한 수치이다.

〈표 1〉 실험 데이터 특성

	width/ height	number of frame	average execution time	worst case execution time
Beatles	352/240	5340	23.2	42.7
K2	240/176	1579	17.8	37.8
Ipaq	304/228	8874	10.3	18.9



(그림 1) 실험 데이터의 프로세서 사용량 측정



(그림 2) X 윈도우 시스템의 프로세스 사용량

(그림 2)는 앞에서 실험을 위해 사용된 세 가지 동영상 클립을 각각 10, 15, 20 frame/sec로 동시에 수행시키면서 X 윈도우의 프로세서 사용량이 어떻게 변하는지 보이고 있다.

10 frame/sec로 수행되는 MPEG-1 디코더의 경우 Beatles, K2 그리고 Ipaq을 동시에 수행시킬 때 대략 10 퍼센트 정도의 프로세서를 사용한다. 하지만 20 frame/sec로 프레임 비율을 증가시킬 때 X 윈도우가 차지하는 프로세서 사용량은 20 퍼센트까지 급격히 증가하고 있다. 즉 멀티미디어 응용이 수행되는 주기가 짧을수록 더 많은 요청을 X 서버로 보내므로, 프로세서 사용량이 급격히 증가하고 있음을 알 수 있다. 이와 같이 범용 운영체제 환경에서는 멀티미디어 응용과 X 윈도우 시스템 간에는 프로세서 사용량 측면에서 비례 관계에 있는 것을 알 수 있다. 따라서 새로운 멀티미디어 태스크를 유입시킬 때 X 윈도우에 미치는 영향을 고려해야 함을 알 수 있다.

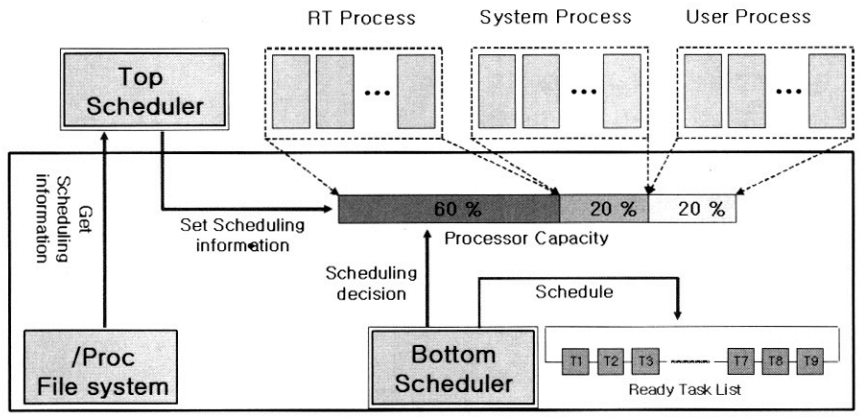
4. 다단계 스케줄링 기법

다단계 스케줄러는 사용자의 피드백을 받아서 스케줄링 정보를 생성하는 상위 스케줄러(Top Scheduler)와 커널에서 연성 실시간 스케줄링과 디스패칭(Dispatching) 기능을 담당하는 하위 스케줄러(Bottom Scheduler)로 구분된다. (그림 3)은 다단계 스케줄러의 개념도를 보이고 있다.

본 논문에서는 사용자 영역에서 수행되는 태스크를 크게 세 가지로 분류하고 있다. 첫 번째는 연성 실시간으로 수행시키고자할 때 사용하는 실시간 프로세스이며, 두 번째는 X 서버와 같은 시스템 프로세스이며, 마지막이 일반 사용자 프로세스이다.

4.1 시스템 프로세스를 고려한 스케줄링 기법

시스템 프로세스는 X 서버와 관련된 데몬 프로세서 및 시스템을 유지하는데 필요한 핵심적인 데몬 프로세서들을 포함한다. 멀티미디어와 같은 연성실시간 프로세스가 화면



(그림 3) 다단계 스케줄러 개념도

예 영상 및 그래픽을 출력하고자 할 때 X 서버와 같은 데몬의 도움을 필요하므로 일정한 프로세서 사용량을 시스템 프로세스들을 위해서 할당해야 한다. 일반 프로세스는 사용자 프로세스로 실시간 프로세스와 시스템 프로세스가 수행하고 남는 프로세서 사용량 한도 내에서 실행된다. 다단계 스케줄링 알고리즘은 실시간 태스크와 비실시간 태스크가 존재하는 범용 시스템을 대상으로 하고 있으며, 실시간 태스크는 주기적 태스크 모델(Periodic Task Model)[18]을 가정하고 있다. 여기서 주기적 태스크 모델은 태스크의 계산 및 데이터 전송이 일정한 주기(Period)로 일정한 수행 시간(Execution Time)을 가지고 반복되는 것을 의미한다. 주기적 태스크 모델에서는 태스크 집합이 스케줄링 가능한지를 판단하기 위해서 프로세서 이용률(Processor Utilization)을 이용한다. n 개의 태스크를 가지는 집합 A에 있어서, 각 태스크의 주기를 p, 예상 수행 시간을 e 라 할 때 프로세스 이용률 $U(A)$는 다음과 같은 식을 따른다.

$$U(A)=\sum_{i=1}^{n}\frac{e_i}{p_i} \tag{1}$$

이때 대부분의 실시간 스케줄링 기법에서 예상 수행 시간은 최악 수행 시간을 적용하게 된다. 여기서 $U(A)$의 의미는 프로세서를 얼마나 이용하고 있는지를 나타내주는 수치가 되며, 가용한 프로세서 이용률을 초과해서는 안 된다. 예를 들어 $U(A)$의 값이 0.5인 경우, 태스크의 집합 A의 수행으로 인하여 프로세서가 대략 50 퍼센트 정도 사용되고 있음을 나타낸다. $U(A)$의 값은 스케줄링 가능성을 테스트하는 값으로 사용된다. 멀티미디어 스케줄러로 널리 사용되고 있는 마감시간 우선(Earliest Deadline First) 스케줄러의 경우 한계 값이 1이며, 따라서, 마감시간 우선 스케줄링을 하는 경우 $U(A)$의 값은 1을 초과할 수 없다[18].

다단계 스케줄링 알고리즘은 프로세스의 집합을 연성 실시간 프로세스의 집합 R과 시스템 프로세스 집합 S 그리고 일반 사용자 프로세스의 집합 N이 있다고 가정한다. 그리고 연성 실시간 프로세스가 제한된 시간 내에 수행되기 위해서는 연성 실시간 프로세스의 집합 R의 프로세서 이용률 $U(R)$과 시스템 프로세스의 집합 S의 프로세스 이용률 $U(S)$의 합계가 1을 초과해서는 안 된다.

$$0\le U(R)+U(S)\le 1 \tag{2}$$

이때, 사용자 프로세스들은 연성 실시간 프로세스와 시스템 프로세스가 수행하고 남아있는 여유 프로세서 이용률을 이용해서 수행해야 한다. 아래 식은 사용자 프로세스의 집합이 사용할 수 있는 프로세서 이용률의 범위를 보이고 있다.

$$0\le U(N)\le 1-U(R)-U(S) \tag{3}$$

다단계 스케줄링 알고리즘은 2장에서 살펴본바와 같이 멀티미디어 태스크의 프로세스 이용률이 높아지는 경우에 시스템 프로세스도 비례적으로 프로세스 이용률이 높아진다는 사실을 가정하고 있다. 따라서 새로운 연성 실시간 태스크 τ 이 동적으로 유입되는 경우에 태스크 τ가 수행가능한지 스케줄링 테스트를 하는 단계는 다음과 같다.

$$U(R_{new})=U(R_{old})+U(\tau)$$
$$U(S_{new})=U(S_{old})+U(\tau)*\frac{U(S_{old})}{U(R)+U(S_{old})} \tag{4}$$

식에서 새롭게 유입되는 연성 실시간 태스크 τ의 프로세스 이용률만큼 전체 연성 실시간 프로세스의 이용률이 보장되어야 하며, 시스템 프로세스의 이용률은 새롭게 유입된 연성 실시간 프로세스의 프로세스 이용률에 비례해서 늘어나게 된다. 따라서 새롭게 유입된 연성 실시간 프로세스가 제한된 시간 내에 수행되기 위해서는 아래와 같은 수식을 만족시켜야 한다.

$$0\le U(R_{new})+U(S_{new})\le 1 \tag{5}$$

4.2 사용자 피드백을 고려한 스케줄링 기법

멀티미디어 서비스가 보편화되면서 실시간 처리 및 높은 대역폭을 요구하는 서비스가 점차 늘어나고, 그 종류도 다

양해지고 있다. 많은 사용자가 동시에 다양한 멀티미디어 서비스를 요청하는 경우, 사용자에게 일정한 대역폭을 할당한 다음 지속적으로 사용자가 만족할만한 서비스 품질 수준을 계속 보장할 수 있는 메커니즘이 있어야 한다. 이러한 개념으로 등장한 것이 QoS(Quality of Service)이며, 주로 네트워크 분야에서 많이 사용되어 왔다. 하지만, QoS의 개념은 네트워크 분야에 한정되지 않고, 사용자가 요청하는 작업에 대해서 일정한 수준의 서비스 품질을 제공해 주어야 하는 다양한 분야에서 널리 활용되고 있다. 네트워크 분야에서 QoS는 전송 시스템의 전송 품질과 서비스 이용도를 나타내는 성능 단위이며, 네트워크의 전송율(Transmit Rate), 에러율(Error Rate) 및 기타 특성들로 나타낼 수 있다.

멀티미디어의 경우 가용한 자원의 상황에 따라서 서비스 품질을 조절함으로써, 사용자의 요구를 만족시킬 수 있다. 일례로 동영상 데이터의 경우, 프로세서 자원이 부족한 경우 일부 비디오 프레임을 드롭(Drop)시키거나, 동영상 프레임의 크기를 줄임으로써 디코딩에 소요되는 프로세싱 파워를 줄일 수 있다. 이와 같이 멀티미디어 응용은 가용한 자원의 상태에 따라서 적응적인 방법을 동원해서 수행가능하다. 이를 위해서 시스템 상황에 따라서 사용자의 피드백을 수용하거나 가용한 자원 상태에 따라서 어떻게 대처해야 하는지에 대한 사용자의 요구를 수용할 수 있어야 한다. 본 연구에서는 상위 스케줄러에서 사용자의 피드백을 받아서 커널 내부의 스케줄링 정보를 갱신해주는 방법을 통하여 이러한 문제를 해결하고 있다. 상위 스케줄러의 기본적인 동작 원리는 (그림 4)와 같다.

```
Top_scheduler()
{
 while(){
  wait_event(); wait user event, or timer
  if(user event){
    if(realtime process){
      scheduling_test(new_process) ;
      if (possible) then add to realtime group;
      else renegotiate;
    }
    else if(system process){
      scheduling_test(new_process) ;
      if (possible) then add to system group;
      else renegotiate;
    }
    else if(user process)
      priority adjustment : nice();
  }
  else { //timer event
    gather system resources;
    show system resource status;
  }
  select(); // periodically controls top_scheduler()
 )
}
```

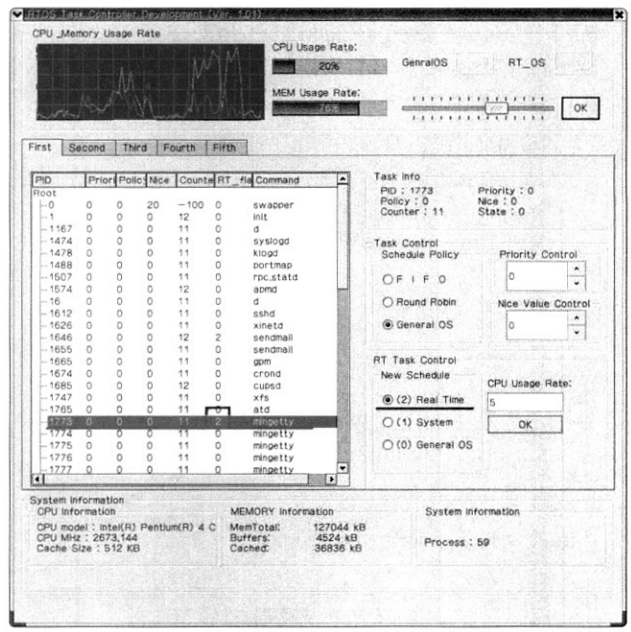
(그림 4) 상위 스케줄러 의사 코드

상위 스케줄러는 실시간 태스크 중에서 가장 우선순위가 높게 되어 있으며, 언제든지 사용자의 피드백을 통해서 전체 시스템의 프로세스 우선순위 및 태스크 그룹을 재조정할 수 있다. 사용자 피드백은 상위 스케줄러에 의해서 사용자 이벤트로 처리되며, 사용자 이벤트가 없는 경우에는 타이머에 의해서 일정한 시간마다 주기적으로 실행되도록 하였다. 따라서 사용자 이벤트에 의해서 새로운 프로세스의 유입이나 우선순위 조정이 발생하는 경우 이를 처리하고, 타이머는 실시간 프로세스의 수행여부를 관찰하며 자원에 대한 관리를 수행한다.

사용자가 런타임에 하위 스케줄러로 보내는 피드백 정보는 다음과 같다. 첫 번째, 태스크의 그룹 조정을 통하여 현재 수행중인 태스크를 연성 실시간 클래스, 시스템 클래스, 범용 클래스로 이동시킬 수 있다. 이를 통하여 태스크를 실시간적으로 수행시키거나 과부하 상황에서 중요하지 않은 태스크가 실시간 태스크로 수행되는 것을 방지할 수 있다. 두 번째, 연성 실시간 태스크의 프로세서 용량 조절을 통하여 연성 실시간 태스크 사이에 프로세서 용량에 대한 경쟁을 하는 것을 사용자가 조절할 수 있다. 세 번째, 태스크의 우선 순위값 조절을 통하여 연성 실시간 클래스, 시스템 클래스 그리고 범용 클래스에 속하는 태스크들 사이에 우선순위 조절을 할 수 있다.

(그림 5)는 상위 스케줄러의 동작 모습을 보이고 있다. 사용자들은 현재 수행되는 프로세스에 대한 정보를 볼 수 있으며 해당 프로세스를 클릭하면 상단에 프로세스 사용 정보 및 메모리 사용 정보 등을 실시간으로 볼 수 있다. 또한 수행중인 태스크의 속성을 실시간 프로세스로 전환하거나 실시간 프로세스를 일반 프로세스로 전환하는 작업을 선택할 수 있다.

하위 스케줄러는 실제적으로 프로세스를 스케줄링 해주는



(그림 5) 상위 스케줄러 수행 모습

일을 담당하고 있다. 실시간 태스크가 유입될 때 각 태스크마다 프로세스 용량(Processor Capacity)이 할당되며, 할당된 용량을 적게 소모한 태스크의 우선순위를 높여주고 할당된 용량을 초과해서 사용하는 태스크의 우선순위를 낮추는 방법을 사용하고 있다. 따라서 실시간 태스크들이 주어진 프로세서 용량의 범위 내에서 공평하게 수행될 수 있는 방법이며 멀티미디어 태스크를 수행시킴에 있어서 효과적으로 사용될 수 있다. 사용한 프로세서 사용량의 값이 할당된 프로세서 사용량의 값보다 커지는 경우 우선순위를 낮추는 방법을 사용해서 실시간 프로세스가 주어진 프로세서 사용량 범위 내에서 수행될 수 있도록 관리한다. 이때에 하나의 실시간 프로세스가 오랜 시간동안 프로세서를 독점하는 것을 방지하기 위해서 실시간 프로세스 내에서 타임 슬라이스(time slice)를 할당하는 방법을 취하고 있으며 주어진 타임 슬라이스를 사용하는 경우 다른 실시간 프로세스에게 프로세서를 양도한다.

실시간 프로세스 중에서 스케줄링 대상이 없는 경우에 시스템 프로세스 내에서 스케줄링 대상을 찾는다. 이때 전체 시스템 프로세스에 할당된 프로세서 용량을 초과해서 사용하는 경우에 사용자 프로세스 내에서 스케줄링 대상을 찾게 된다. 즉 실시간 프로세스는 개별 프로세스가 독점적으로 수행할 수 있는 프로세서 용량을 할당받아서 수행되는 반면에 시스템 프로세스는 전체 시스템 프로세스 단위로 프로세서 용량이 할당되며, 할당된 범위 내에서 가장 우선순위가 높은 시스템 프로세스가 수행되는 것이다.

```
Bottom_scheduler()
{
target = NULL;

for(all realtime process){
 task[i]->ratio = used_PC/allocated_PC;
 if((task[i]->ratio < 1) && (task[i]->ratio < tar-
 get->ratio)
     && (task[i]->slice > 0)){
     target=task[i];
     return;
  }
}

system->ratio = used_PC/allocated_PC;
  if((system->ratio < 1) && (system->slice > 0))
     target = high priority system process;
  }
if(target == NULL)
  target = high priority user process;
}
```

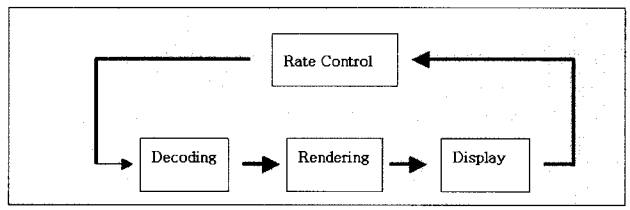
(그림 6) 하위 스케줄러 의사 코드

5. 실험 결과 및 성능 분석

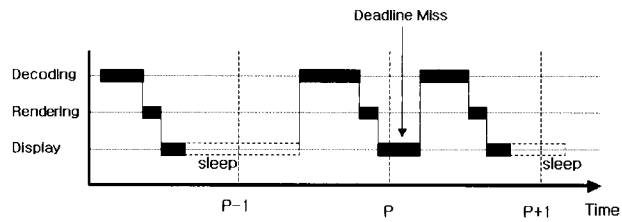
제안된 기법의 유용성을 보이기 위해서 리눅스 커널 상에서 구현 및 실험을 하였다. 실험에 사용된 운영체제는 Linux 커널 버전 2.4이며, Intel Pentium 500MHz, 메모리 64MB. 실험을 하였으며, 버클리 대학의 MPEG-1 동영상 플레이어를 연성 실시간 프로세스로 사용하였다. 버클리 대학의 MPEG-1 동영상 플레이어의 구조를 살펴보면, (그림 7)과 같이 디코딩(Decoding) 모듈, 렌더링(Rendering) 모듈, 디스플레이(Display) 모듈, 그리고 비율 제어(Rate Control) 모듈로 구성되어 있다.

동영상 플레이어는 MPEG-1 데이터를 읽어서 디코딩, 렌더링, 디스플레이를 수행하며, 일정한 프레임 비율(Frame Rate)에 따라서 시간 간격을 두고 이 과정을 반복하게 된다. 여기서 프레임 비율은 초당 디스플레이 되는 프레임의 개수를 지칭하며, 프레임 비율의 역수는 주기를 의미한다. 즉 프레임 비율이 20이라는 것은 초당 20장의 프레임이 디스플레이 되는 것이며, 이것은 100밀리 세컨드의 주기를 갖는다. 디코딩, 렌더링, 그리고 디스플레이가 수행이 완료된 후, 다음 주기까지 대기하기 위해서는 비율 제어 모듈에서 다음 주기가 시작되는 부분까지 지연을 시키게 된다. 디코더의 비율 제어 모듈은 매주기마다 동영상 데이터 처리가 시작될 때 현재 시스템 시간을 읽고 주기를 더한 값을 다음 프레임이 처리되는 시작 시간을 결정한다.

그리고 디코딩, 렌더링 및 디스플레이를 수행한 후에 비율 제어 모듈에서 현재 시간과 다음 주기의 시작시간 값을 비교한다. 이때 다음 주기의 시작 시간보다 현재 시간이 작은 경우 다음 주기의 시작 시간까지 일정한 시간을 수면 상태로 전환하게 되며, 만약 다음 주기의 시작 시간보다 늦게 작업이 끝난 경우 다음 주기의 작업을 계속 수행하게 된다. (그림 8)은 앞에서 언급한 방식을 통해서 MPEG-1 동영상 디코더가 수행되는 단계를 보여 주며, 연성 실시간 태스크의 제한시간 실패의 의미를 그림으로 보이고 있다.



(그림 7) MPEG-1 디코더의 내부 구조

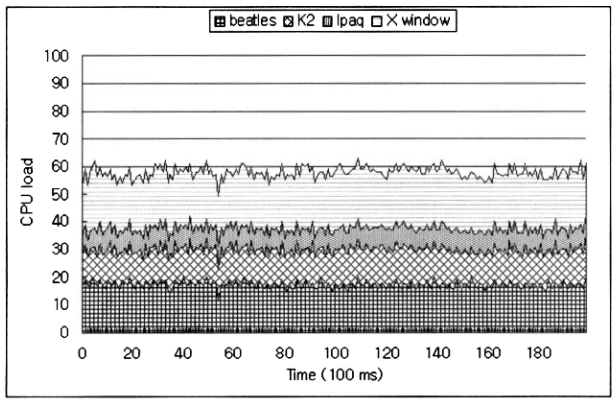


(그림 8) MPEG-1 수행 단계

5.1 실험 환경

실험은 각 동영상 플레이어에게 일정한 프로세서 용량을 지정해주고 다단계 스케줄러가 주어진 자원을 제대로 할당해주는지 테스트하였다. 그리고 동영상 플레이어가 수행되는 중간에 일반 사용자 프로세스를 유입시켜서 과부하에 어떻게 반응하는지 살펴보았다. 이때 사용한 실험 태스크는 다음과 같다.

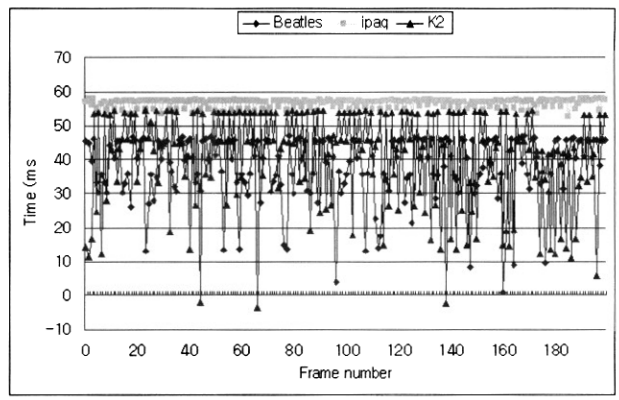
- mpeg_play : 버클리 대학에서 개발한 MPEG-1 디코더이며, 주어진 비디오 데이터를 일정한 주기마다 한번씩 디코딩하는 연성 실시간 태스크이다. mpeg_play에서 사용된 비디오 데이터는 Beatles, K2, Ipaq이다. 이때 각 비디오 데이터가 사용하는 프로세서 용량은 (그림 9)에서 보는 것과 같이 Beatles: 20%, K2: 15%, Ipaq: 5% 이다.
- cscope : 소스 분석을 위한 툴로써, 소스에서 발견되는 심볼들 즉, 변수, 함수 그리고 문자열에 대해서 상호 참조할 수 있는 소스 데이터베이스를 구성함으로, 쉽게 소스 코드를 분석할 수 있게 도와주는 프로그램이다. 리눅스 커널 코드에 대해서 소스 데이터베이스를 만드는 작업을 수행시켜서 과부하를 발생시킨다.
- gcc : 리눅스 커널 소스 코드를 컴파일 한다. 주로 프로세서를 활용하는 작업이 대부분이므로 CPU 집중적인 특성을 가지고 있으며, 많은 물리 메모리를 필요로 한다. 리눅스 커널 코드에 대해서 컴파일하는 작업을 수행함으로써 과부하를 발생시킨다.



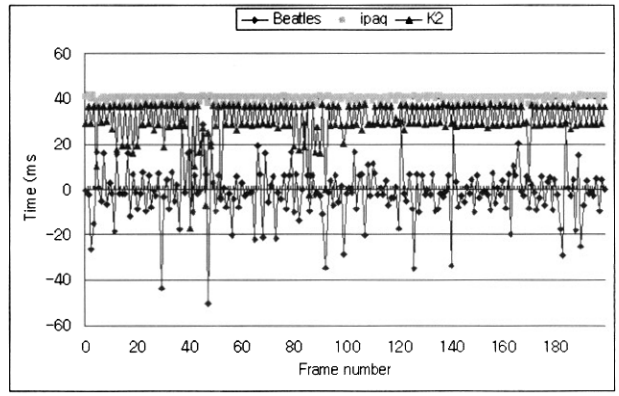
(그림 9) mpeg_play의 프로세서 사용량

5.2 과부하 제어 실험

과부하 제어를 어느 정도 효과적으로 처리하는지 살펴보기 위해서 다단계 스케줄러와 리눅스 스케줄러에 각각 과부하를 발생시켰다. 첫 번째 실험에서는 gcc 프로세스만 추가적으로 수행시켰으며, 두 번째 실험에서 gcc와 cscope를 동시에 수행시키는 gcc+cscope 방법을 사용하였다. 첫 번째 실험에서 (그림 10)의 다단계 스케줄러의 경우에는 각 mpeg_play의 프로세서 용량을 보호해줌으로써 제한시간의 실패가 거의 일어나지 않고 있음을 알 수 있다. 하지만, (그림 11)의 리눅스 스케줄러의 경우는 새로운 프로세스가 유



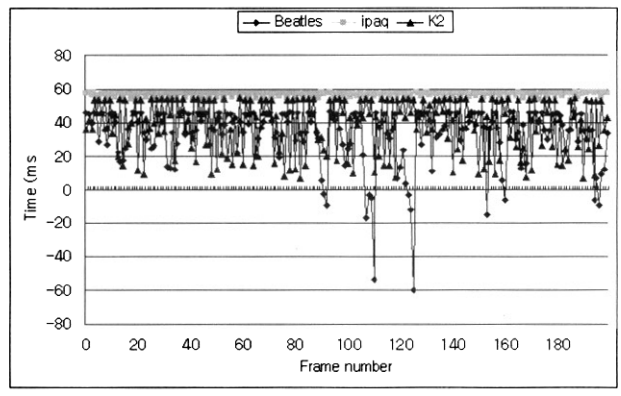
(그림 10) 다단계 : mpeg_play + gcc



(그림 11) 리눅스 : mpeg_play + gcc

입되면서 제한시간 실패가 급격히 높아졌으며 특히 프로세서 용량을 많이 사용하고 있는 Beatles의 경우에 제한시간 실패가 가장 많았다. 그 이유는 리눅스 스케줄러가 프로세서를 많이 사용하는 태스크의 우선순위를 낮추기 때문에 프로세서를 많이 사용하는 Beatles, K2, Ipaq 순으로 제한시간의 실패가 높은 것이다.

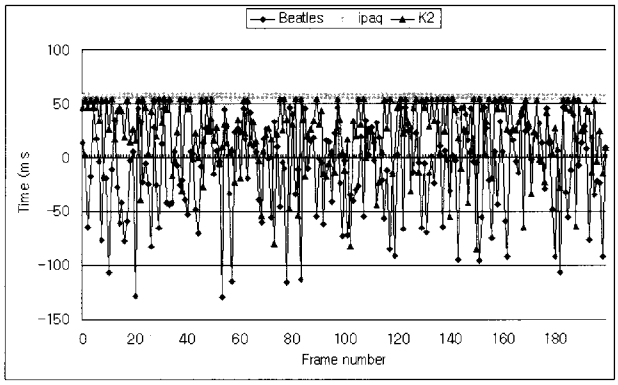
두 번째 실험에서 gcc와 cscope를 동시에 수행시키는 오버헤드를 추가하였다. (그림 12)의 다단계 스케줄러는 일부 구간에서 제한 시간 실패 횟수가 증가하였으나 전체적으로 대부분의 비디오 데이터가 제한시간 내에 수행되고 있음을



(그림 12) 다단계 : mpeg_play + gcc + cscope

〈표 2〉 제한시간 실패횟수

	Processor Capacity	MUSMA		Linux		SMART-Linux	
		gcc	gcc+ cscope	gcc	gcc+ cscope	gcc	gcc+ cscope
Beatles	20	0	13	67	106	8	65
K2	15	3	4	32	87	5	43
Ipaq	5	0	0	2	12	0	2
X 서버	20	n/a	n/a	n/a	n/a	n/a	n/a



(그림 13) 리눅스 : mpeg_play + gcc + cscope

알 수 있다. (그림 13)의 리눅스 스케줄러는 과부하 발생에 대해서 제어를 하지 못하고 있으며, 대부분의 비디오 데이터가 제한 시간을 놓치는 현상이 발생되었다.

5.3 사용자 피드백에 대한 실험 결과

사용자 피드백에 대해서 다단계 스케줄러가 효율적으로 동작함을 보이기 위해서 실시간 태스크와 gcc, cscope 등으로 전체 시스템을 과부하 상황으로 만든 후에, 사용자 프로세스로 수행되면서 K2 비디오 클립을 디코딩하는 mpeg_play를 100 프레임 근방에서 연성 실시간 그룹으로 변경시켜주었다. (그림 14)는 사용자 프로세스 그룹에서 수행되는 mpeg_play를 보이고 있으며, (그림 15)는 연성 실시간 그룹으로 변경된 mpeg_play의 수행 모습을 보이고 있다. 사용자 피드백에 의해서 즉각적으로 제한시간을 놓치지 않는 동영상 출력을 보여주고 있다.

5.4 관련 연구 결과와 비교

본 논문에서 제시하는 다단계 스케줄링 기법의 우수성을 검증하기 위해서 실시간 리눅스 시스템과 비교를 하였다. 리눅스 시스템에서 실시간 스케줄링 기법을 제공해주는 RT-Linux는 실시간 처리를 필요로 하는 작업을 커널 내부에서 태스크로 수행시키고 CUI에서 수행되는 클라이언트 부분을 리눅스 응용 프로그램으로 수행시키는 이중 구조 방식을 취하고 있다. 따라서 멀티미디어 응용을 수행시키기 위해서는 멀티미디어의 실시간 처리 부분이 커널 내에서 수행되어야 하는데, 이러한 접근 방식은 멀티미디어 개발을 어렵게 하고 커널 내에서의 태스크의 오류가 전체 시스템의 오동작을 불러일으킬 수 있는 단점이 있다. 또한 커널 내에서 수행되는 태스크와 GUI 클라이언트는 실시간 큐를 통해서 데이터의 전달이 이루어져야 하는데, 대용량의 데이터를 처리하는 멀티미디어 응용의 구조에 적합하지 않다.

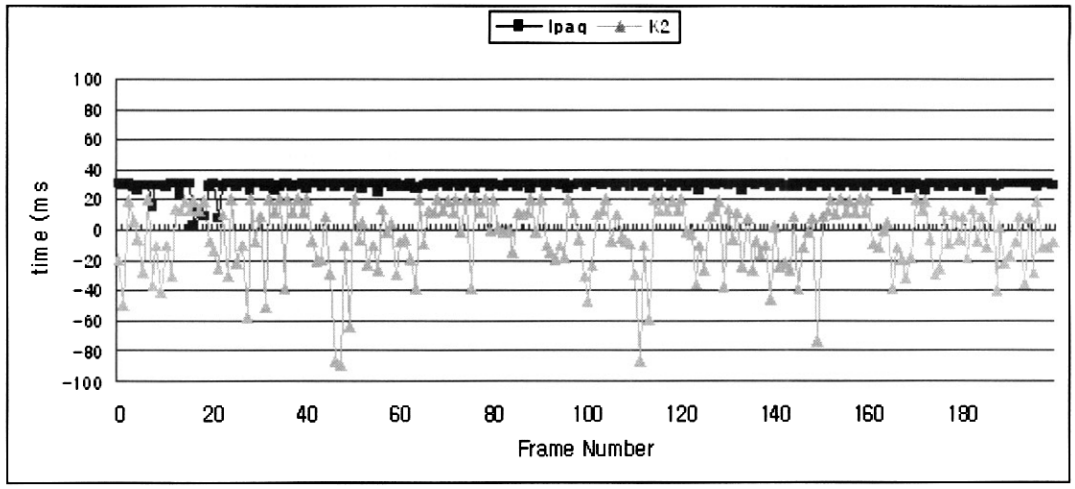
SMART-Linux는 범용 운영체제 환경에서 멀티미디어 처리에 적합한 스케줄링 기법을 제공해주고 있다. 다단계 스케줄러와 동일한 하드웨어 사양에서 실험을 한 결과 제한시간 실패 횟수 측면에서 다음과 같은 결과를 보이고 있다.

SMART-Linux의 경우 새로운 태스크가 유입될 때 이를 제어할 수 있는 수단이 없으며, 시스템에 과부하가 발생하였을 때에 이를 제어할 수 있는 방법이 제공되고 있지 않다는 단점이 있다. 따라서 <표 2>에서 보이는 것처럼 시스템에 과도한 오버헤드가 발생하는 경우에 전체 시스템의 성능이 저하되고 멀티미디어 응용이 제한시간을 놓치는 결과가 발생한다. 또한 태스크의 중요도와 긴급도를 파라미터로 스케줄링을 해주는 것은 좋은 방법이지만 한번 정해진 스케줄링 정책에 따라서 계속적으로 수행되며 사용자의 선호도에 따른 피드백을 처리하는데 한계가 있다.

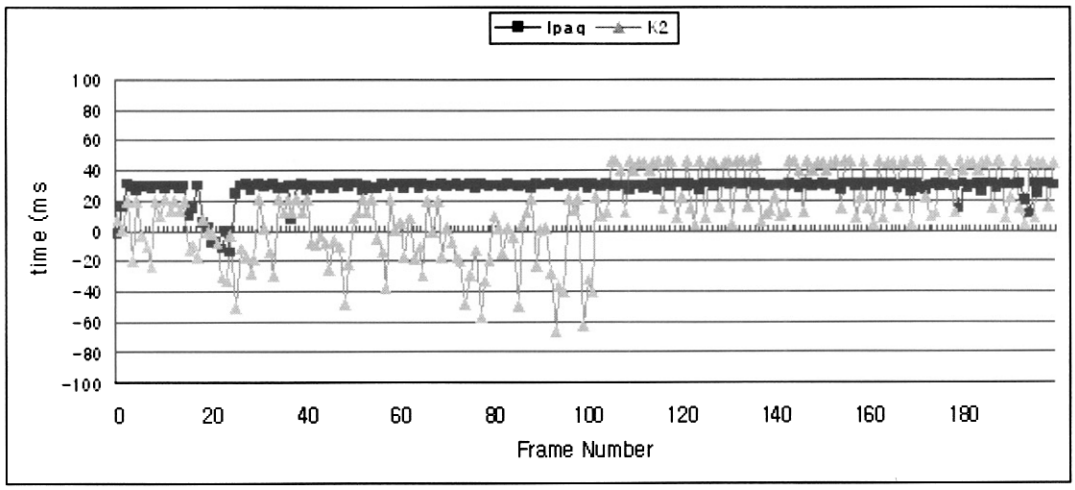
6. 결 론

멀티미디어 응용과 같은 연성 실시간 특성을 가진 프로세스를 효율적으로 수행시키기 위해서 시스템은 실시간 스케줄링 기법 및 다양한 자원 관리 기법이 제공되어야 한다. 본 연구에서는 다단계 스케줄링 기법을 통해서 멀티미디어를 효율적으로 수행시킬 수 있는 방법을 제안하였다. 본 논문에서 제안하는 다단계 스케줄링 기법은 특히 과부하가 발생되었을 때 멀티미디어 응용이 제한 시간을 놓치지 않고 수행될 수 있도록 보장을 해주며 런타임에 사용자의 피드백에 따라서 스케줄링 방식을 수정할 수 있는 유연성을 제공하고 있다. 기존의 연구와 차이점은 멀티미디어 응용의 특성을 파악하고 멀티미디어와 IPC를 통해서 동작하는 시스템 프로세스의 우선순위를 고려한 스케줄링 기법을 적용하고 있다는 점이다. 또한 사용자의 피드백을 수용하여 스케줄링 정책을 결정해주는 상위 스케줄러를 사용한 것이며 이를 통하여 멀티미디어처럼 사용자 피드백이 필요한 시스템에서는 적합하게 사용될 수 있다는 것을 보였다.

본 연구 결과를 바탕으로 향후 연구방향은 프로세서 자원뿐만이 아니라 메모리 및 입출력 자원에 대하여 다단계 스케줄링 알고리즘을 확장하는 것이며, 이를 통해서 범용 운영체제 환경에서 다양한 형태의 멀티미디어 응용이 원활히 수행될 수 있도록 할 것이다.



(그림 14) 사용자 피드백이 없는 경우의 mpeg_play



(그림 15) 사용자 피드백을 적용한 mpeg_play

참 고 문 헌

[1] Raj Yavatkar, K. Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications," In 6th International NOSSDAV Workshop., 1995.

[2] P. Goyal, X. Guo, H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," Proceedings of the Second Symposium on Operating Systems Design and Technology, June, 1997.

[3] J, Nieh and Monica S. Lam, "The Design, implementation and Evaluation of SMART: A Scheduler for Multimedia Applications," Proceedings of 16th ACM Symposium on Operating Systems Principles, St Malo, France, October, 1997.

[4] Michael Barabanov, A Linux-based real-time operating system, master's thesis, New Mexico institute of Mining and

[5] Balaji Srinivasan, A Firm Real-Time Implementation using Commercial Off-The-Shelf Hardware and Free Software, Master's thesis., University of Kansas, 1998.

[6] http://www.ime.usp.br/~dilma.

[7] S. Childs and D. Ingram, "The Linux-SRT integrated multimedia operating system: bringing QoS to the desktop." In 7th Real-Time Technology & App. Symp., p.135, May, 2001.

[8] S. Oikawa and R. Rajkumar, "Linux/RK: A Portable Implementation, Seattle, WA, pp.107-122, Oct., 1996.

Resource Kernel in Linux," In 19th IEEE Real-Time Systems Sumposium, Madrid, Spain, pp.2-4, Dec., 1998.

[9] Yu-Chung Wang and Kwei-Jay Lin, "Implementing a general real-time scheduling framework in the RED-Linux real-time kernel," In Proc. of the 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, pp.246 - 255, December, 1999.

[10] Vijay Sundaram, Abhishek Chandra, Pawan Goyal, Prashant Shenoy, Jasleen Sahni, and Harrick Vin, "Application performance in the QLinux multimedia operating system,"

In Proc. of the 8th ACM Conf. on Multimedia, Los Angeles, CA, November, 2000.

[11] Veronica Baiceanu, Crispin Cowan, Dylan McNamee, Calton Pu, and Jonathan Walpole. "Multimedia applications require adaptive cpu scheduling," In Proceedings of the Workshop on Resource Allocation Problems in Multimedia Systems, Washington, DC, USA, 1996.

[12] John Regehr and John A. Stankovic. "Augmented CPU Reservations: Towards Predictable Execution on General-Purpose Operating Systems," In Proceedings of the 7th Real-Time Technology and Applications Symposium(RTAS 2001), Taipei, Taiwan, May, 2001.

[13] A. Goel, L. Abeni, C. Krasic, J. Snow, and J. Walpole. "Supporting Time Sensitive Applications on a Commodity OS," In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation. Usenix, Dec., 2002.

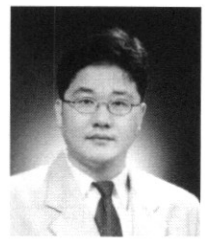
[14] J. W.-S. L. Z. Deng, "Scheduling Real-Time Applications in a Open System Environment," in Proceeding of the 18th IEEE Real-Time Systems Symposium(RTSS97), June, 1997.

[15] Yu-Chung and Kwei-Jay Lin. "Enhancing the real-time capability of the Linux kernel," In IEEE Real Time Computing Systems and Applications, October, 1998.

[16] K. Jeffay, G. Lamastra, "A Comparative Study of the Realization of Rate-Based Computing Services in General Purpose Operating Systems," Proceedings of the Seventh IEEE International Conference on Real-Time Computing Systems and Applications, Cheju Island, South Korea, pp.81-90, December, 2000.

[17] http://www.xfree86.org.

[18] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, Vol.20, No.1, pp.40-61, 1973.



고 영 웅

e-mail : yuko@hallym.ac.kr

1997년 고려대학교 컴퓨터학과(학사)

1999년 고려대학교 컴퓨터학과(이학석사)

2003년 고려대학교 컴퓨터학과(이학박사)

2003년 고려대학교 정보통신연구소 박사후 연구원

2003년~2005년 한림대학교 정보통신공학부 전임강사

2005년~현재 한림대학교 정보통신공학부 조교수

관심분야 : 임베디드 시스템, 실시간 시스템, 유비쿼터스 시스템