멀티미디어 스트림의 QoS를 보장하는 통합형 파일시스템

(An Integrated File System for Guaranteeing the Quality of Service of Multimedia Stream)

김 태 석 † 박 경 민 †† 최 정 완 ††† 김 두 한 ††††
(Tae-seok Kim) (Kyung-min Park) (Jung-wan Choi) (Doo-han Kim)

원 유 집 ††††† 고 건 †††††† 박 승 민 ††††††† 김 정 기 †††††††
(Youjip Won) (Kern Koh) (SungMin Park) (JeongKi Kim)

요 약 통합형 파일시스템이 차세대 파일시스템으로 주목받으면서 디지털 셋톱박스나 스트리밍 서버에서 멀티미디어 서비스의 QoS를 보장함과 동시에 텍스트 기반의 웹 문서나 그림 파일 등의 입출력 요청을 처리하는 것이 중요한 이슈로 부각되고 있다. 그러나 하나의 파일 서버에서 다양한 요구사항을 지닌 입출력들을 동시에 처리해야 하는 경우, 입출력 요청들의 디스크 탐색시간을 최소화하는 것을 목표로 하는 기존의 파일시스템에서는 멀티미디어 스트림의 QoS를 만족할 수가 없다. 본 논문에서는 이처럼 다양한 입출력 요청이 혼재하는 통합형 환경에서 멀티미디어 스트림의 QoS를 보장하는 파일시스템 개발에 대해 논의하고자 한다. 먼저 다양한 응용프로그램들의 입출력 요청을 QoS 요구사항에 따라 크게 두 가지-주기적인 요청, 산발적인 요청-로 구분하고, 입출력 요청 처리 수준에서 주기적인 요청에 높은 우선순위를 부여함으로써 멀티미디어 스트림의 QoS를 보장하도록 하였다. 또한, 이러한 메커니즘을 리눅스 운영체제에 구현하여 그 성능과 효과를 검증하였다.

키워드 : 디스크 스케줄링, 멀티미디어 서비스, QoS 보장

Abstract Handling mixed workload in digital set-top box or streaming server becomes an important issue as integrated file system gets momentum as the choice for the next generation file system. The next generation file system is required to handle real-time audio/video playback while being able to handle text requests such as web page, image file, etc. Legacy file system provides only best effort I/O service and thus cannot properly support the QoS of soft real-time I/O. In this paper, we would like to present our experience in developing the file system which can guarantee the QoS of multimedia stream. We classify all application I/O requests into two category: periodic I/O and sporadic I/O. The QoS requirement of multimedia stream could be guaranteed by giving a higher priority to periodic requests than sporadic requests. The proto-type file system(Qosfs) is developed on Linux Operating System.

Key words : disk scheduling, multimedia services, the QoS guarantee

· 본 논문은 2000년도 한양대학교 교내연구비에 부분 지원 되었음
† 비 회 원 : 서울대학교 컴퓨터공학부
tskim@oslab.snu.ac.kr
†† 비 회 원 : (주)삼성전자
kyungmin78@hotmail.com
††† 비 회 원 : LG전자 기술원
chrys@lge.com
†††† 비 회 원 : (주)삼성전자
lissom33@ece.hanyang.ac.kr
††††† 종신회원 : 한양대학교 전자전기컴퓨터공학부 교수
yjwon@ece.hanyang.ac.kr
†††††† 종신회원 : 서울대 컴퓨터공학부 교수
kernkoh@oslab.snu.ac.kr
††††††† 비 회 원 : 한국전자통신연구원 임베디드 S/W 기술센터
minpark@etri.re.kr
jkk@etri.re.kr
논문접수 : 2003년 3월 6일
심사완료 : 2004년 5월 18일

1. 서 론

최근 급속도로 성장한 광대역폭을 기반으로 한 인터넷 동영상 서비스, 디지털 방송 서비스가 상용화됨에 따라 멀티미디어 데이타를 저장하고 처리하는 스트리밍 서버에 대한 관심이 날로 증가하고 있다. 주문형 비디오, 인터넷 폰, 비디오 회의, 원격 교육 등 인터넷을 통한 멀티미디어 서비스에서 멀타미디어 데이타는 매우 높은 전송 대역폭과 저장 공간을 요구한다. 또한 멀티미디어 데이타는 실시간적인 특성 때문에 텍스트나 이미지 형태의 데이타와는 다른 매우 높은 계산 요구조건 및 입출력 처리 제약을 가지고 있다. 따라서 사용자가 원하는 정보를 보다 빠르고 정확하게 제공하기 위해서는 CPU, 저장장치 및 파일 시스템, 네트워크 등에서 정확한 자원의 할당과 스케줄링이 요구된다.

하나의 디스크 파티션에서 이처럼 실시간 특성을 지니고 있는 멀티미디어 데이타와 그렇지 않은 텍스트 기반의 데이타(데이타베이스 검색, 웹 브라우징 등)를 동시에 서비스하는 경우 멀티미디어 스트림의 QoS를 제대로 보장하기가 어렵다. 즉, 데이타베이스 검색이나 메일 메시지 처리 등의 예기치 못한 산발적인 입출력 요청이 기존의 서비스중인 스트리밍에 영향을 주어서 멀티미디어 파일에 끊김 현상(jitter)이 발생하게 된다. 이는 다양한 요구사항을 지닌 데이타 입출력 요청을 처리하기 위해서 입출력 수준에서 요청들의 구분이 필요한데, 디스크의 탐색시간을 줄이는데 최적화된 기존의 디스크 스케줄링 기법에서는 이처럼 다양한 입출력 요청들을 구분할 수가 없기 때문이다.

이와 관련하여 하나의 디스크 파티션에서 다양한 입출력 요구사항을 효과적으로 처리하는 연구들이 최근에 많이 진행되어 왔다[1-6]. 이들 연구의 대부분은 멀티미디어 서버에서 디스크 스케줄링의 기본적인 접근방법 중의 하나인 라운드 기반 스케줄링를 가정하고 있다. 하지만, 이러한 방법들은 매우 정교한 모델링을 필요로 하며 실제 구현하는 데 있어서 다소 어려운 점이 많다.

본 논문에서는 이러한 라운드 기반의 디스크 스케줄링 대신에 비교적 간단하면서도 효율적인 디스크 프레임웍을 제시한다. 먼저, 디스크 입출력 요청들을 두 가지 클래스 즉, 실시간 특성을 가지는 주기적인(periodic) 요청 클래스와 시간적인 제약이 없는 산발적인(sporadic) 요청 클래스로 나눈다. 그리고, 주기적인 요청의 경우 마감시간(deadline)내에 서비스되어야 자료재생의 의미가 있으므로 산발적인 요청에 비해 높은 우선순위를 부여하여 먼저 처리를 하고, 주기적인 요청이 없는 경우 산발적인 요청들은 디스크 헤드 움직임을 최소화하는 방향으로 스케줄링한다. 이렇게 하면 멀티미디어 스트림의 상영 도중 다른 입출력 요청이 발생하더라도 멀티미디어 스트림의 QoS를 보장할 수 있을 뿐더러 텍스트 기반의 산발적인 입출력 요청도 일정 수준 이상의 서비스를 받을 수 있다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 관련 연구에 대해서 자세히 알아보고, 3장에서는 통합형 환경에서 끊김 없는 멀티미디어 파일의 재생 요건에 대해 살펴본다. 그리고, 4장에서는 멀티미디어 스트림의 QoS를 보장하는 통합형 파일시스템 구현에 대해 자세히 알아보고, 실험 결과는 5장에, 결론은 6장에 실었다.

2. 관련 연구

최근 스트리밍 서버에서 멀티미디어 파일 서비스뿐만 아니라 텍스트나 이미지 등의 다양한 종류의 입출력 서비스의 제공이 필요해짐에 따라 멀티미디어 이외의 서비스를 스트리밍에 영향을 주지 않으면서 제공할 수 있는 기법에 대한 많은 연구가 진행되어 왔다.

Won의 논문에서는 진행하고 있는 멀티미디어의 재생에 영향을 주지 않으면서 산발적으로 발생하는 텍스트나 이미지 등의 입출력 요청을 서비스하는 파일시스템을 제안하고 있다[1]. 일반적으로 멀티미디어 스트림 요청과 산발적인 입출력 요청이 동시에 발생할 경우 산발적인 입출력 요청이 멀티미디어 재생에 영향을 주게 되는데, 여기에서는 라운드 기반의 디스크 스케줄링에서 라운드의 길이를 증가시킴으로써 이 문제를 해결하고 있다.

현재 많이 사용되는 SCAN 계열의 디스크 스케줄링 기법들은 다양한 요구를 서비스 하는데 있어서 부적합하다. 이러한 문제를 해결하고자 Shenoy et al.은 스케줄링을 두 개의 수준으로 나누어 다양한 요구를 수용하는 Cello 디스크 스케줄링 프레임웍을 제안하였다[2,3]. 즉, 디스크 스케줄러를 클래스 종속 스케줄러(class specific scheduler)와 클래스 독립 스케줄러(class independent scheduler)로 구분하여 처리하는데, 여기에서 클래스 독립 스케줄러는 각 클래스별로 할당된 가중치에 비례해서 디스크 자원을 배분하는 전역 스케줄러 역할을 한다. 매 라운드 시작시 클래스 독립 스케줄러가 각각의 클래스 종속 스케줄러를 호출해서 각 클래스에 할당된 디스크 대역폭만큼 입출력 요청을 입출력 대기큐로 넘겨준다. 이에 반해, 클래스 종속 스케줄러는 각 클래스의 요구사항에 맞게 미리 스케줄링을 한 후에 - 실시간 요청의 경우 EDF, 빠른 응답시간이나 많은 처리량을 요구하는 클래스의 경우에는 FCFS - 클래스 독립 스케줄러로부터 호출이 될 때, 자신의 큐에서 해당되는 양만큼만 입출력 요청을 삭제해서 입출력 대기큐로 옮긴다. 이때, 실시간 요청의 QoS를 보장하면서 일반 입출력 요청의 빠른 응답시간을 얻기 위해 슬랙 스

틸링(slack stealing) 기법을 사용하기도 한다.

이와 비슷한 연구로 Nerjes et al.와 Reddy et al.의 논문을 들 수가 있는데, 이들 연구들도 기본적으로 다중 큐와 라운드 기반의 스케줄링을 바탕으로 하고 있다. Nerjes et al.은 서브라운드 개념을 도입하고 여기에 서비스 주기정책, 서비스 입출력 요청 제한 정책, 디스크 스케줄링 정책 등에 대해 소개하고 있다[4,5]. Reddy et al.은 Shenoy et al.이 제안했던 Cello 스케줄러와 유사한 구조를 제시하고, 여기에 VBR처리 등의 부가적인 연구를 소개하였다[6].

차세대 파일시스템을 위한 효율적인 서버의 구조에 대한 연구도 있었다[7]. 분할형 파일 서버는 각각의 응용 프로그램 클래스가 다른 파일시스템을 가지고 있으며 스토리지 자원도 각각 분할하여 가지고 있는 반면에 통합형 파일 서버는 디스크 대역폭과 같은 시스템 자원을 모든 클래스가 공유한다. 실험 결과, 클래스들 간의 디스크 대역폭을 공유하는 통합형 파일시스템이 분할형 파일 서버에 비해 성능 상의 우수함을 보였다.

멀티미디어 운영체제(OS)와 멀티미디어 미들웨어 중 어느 환경에서 멀티미디어 응용 프로그램이 더 효율적으로 동작하는가에 관한 연구도 진행되었다[8]. 몇 가지 실험 결과, 멀티미디어 미들웨어가 실행시간 오버헤드의 영향에도 불구하고 비교적 괜찮은 성능을 보였다. 하지만, 과부하시에는 OS 커널 메커니즘이 미들웨어 시스템보다 상당히 우수한 것으로 나타났다.

이들 기법들은 멀티미디어 스트림의 실시간 특성과 빠른 응답시간을 요구하는 산발적인 입출력 모두에 대해 최선의 서비스를 목표로 제안되었다. 그러기 위해 대부분의 연구에서 라운드 기반의 디스크 I/O 스케줄링 기법과 각 요청 클래스 별로 큐를 가지는 다중 큐를 사용하고 있다. 이러한 방식은 정교한 모델링을 필요로 하여 구현하는 데 있어서 다소 어려운 점이 많아 연구 중 상당수는 알고리즘만 제안되고 실제로 구현이 되지 않은 경우도 있다. 또한, 각 클래스간의 서비스 할당만 집중하고 수용제어기법을 도입하지 않아 멀티미디어 요청이 많이 들어오는 경우에 멀티미디어 서비스의 품질을 보장해 줄 수 없다.

본 연구에서는 멀티미디어 스트림 재생 중 산발적인 입출력 요청이 들어오더라도 끊김 없는 스트림 재생이 가능한 통합형 파일시스템을 개발하고자 하였다. 즉, 끊김 없는 멀티미디어 스트림 재생에 최우선을 두고, 그 외의 비실시간성의 입출력 요청은 최대한 효율적으로 서비스하고자 하였다. 여기에서는 라운드 기반의 스케줄링과 다중큐를 사용하지 않고, 단일큐를 사용하여 위의 목표를 충족하는 스케줄러를 구현하였고, 또한 멀티미디어 스트림 요청시 수용제어모듈을 사용하여 기존 서비스중인 멀티미디어 재생의 QoS를 보장하도록 하였다.

3. 멀티미디어 시스템에서의 디스크 스케줄링

이 절에서는 멀티미디어 시스템에서 끊김 없는 재생을 위한 요구 사항을 디스크 대역폭 측면에서 면밀히 살펴본다.

3.1 스트리밍 서비스의 연속성

끊김 없는 멀티미디어 재생을 위해서는 일정량의 디스크 대역폭이 반드시 보장되어야 한다. 그런데, 디스크의 비동기적인 특성 때문에 다수의 멀티미디어 세션에게 각 세션이 요구하는 대역폭으로 데이타를 전송하는 것은 보다 정교한 모델링과 시스템의 작동에 대한 심층적인 이해를 필요로 한다. 멀티미디어의 재생은 동기적인 작업이라 할 수 있는데, 이는 단위시간당 미리 정해진 개수의 프레임을 화면에 보여주어야 하는 것을 의미한다. 따라서 이러한 비동기적으로 이루어지는 디스크의 데이타 읽기 동작과 동기적으로 이루어지는 데이타 소모 작업(재생)을 연동시키기 위해서는 일정량의 주기억장치 버퍼가 각 스트림에 할당되어야 한다.

앞에서도 언급했듯이 멀티미디어 디스크 스케줄링에서는 라운드 기반 스케줄링 방식이 주로 사용된다. 라운드 기반 스케줄링 방식은 멀티미디어 재생을 위하여 전체 시간을 라운드라 불리는 단위로 쪼개어 각 라운드마다 일정량의 데이타를 디스크로부터 읽어 들인다. 이 때 디스크에서 한 세션에게 읽어 들이는 데이타의 양은 해당 세션이 한 라운드에서 소모하는 데이타의 양보다 많아야 한다[9].

동영상 서비스를 위해서 파일시스템이 만족시켜야 할 제약 조건을 다음과 같이 기술할 수 있다. (i)한 라운드에서 각 스트림에게 읽혀지는 데이타의 양은 해당 스트림이 한 라운드에 소모하는 데이타양보다 많아야 한다. (ii)모든 스트림이 한 라운드동안 소비하는 전체 데이타를 디스크로부터 읽어 들이는데 걸리는 시간은 라운드의 길이보다 짧아야 한다. 이러한 연속성 제공을 위한 조건을 식 (1)과 식 (2)로 표현할 수 있는데, 여기에서 n_i, m, b, r_i, B_{max}는 각각 한 라운드에서 읽어 들이는 데이타 블록의 수, 스트림의 수, 입출력 단위 크기, i번째 스트림의 재생률 그리고, 디스크의 최대 전송율을 나타낸다.

식 (1)은 단일 주기 T 동안 읽혀지는 데이타량이 동일 시간의 상영에 요구되는 데이타 블록의 양보다 커야 한다는 것을 나타낸다. 그리고, 식 (2)는 모든 스트림의 한 라운드 상영을 위한 데이타 블록을 읽어 들이는 시간이 T보다 작아야 한다는 것을 나타내고 있다. 여기에서 $O(m)$은 m개의 스트림에 데이타를 공급하는 데 소요되는 디스크 오버헤드 즉, 탐색 시간과 회전 지연 시

간이다. 식 (1)과 식 (2)에서 명시된 조건은 디스크 스케줄링(예를 들어 SCAN, FIFO 등)기법에 관계없이 항상 만족되어야 한다.

$$T \cdot ri \leq n_i b \quad (1)$$

$$T \geq \left\{ \sum_{i=1}^{m} \frac{n_i b}{B_{\max}} \right\} + O(m) \quad (2)$$

식 (1)과 식 (2)를 풀면 총 버퍼 크기가 $O\left(\frac{1}{1-\rho}\right)$의 형태로(ρ는 디스크 이용률, $\left(\frac{\sum_{i=1}^{m} r_i}{B_{\max}}\right)$) 증가하게 됨을 알 수 있다[9]. 이러한 특성은 디스크 스케줄링 정책에 관계없이 발생하며, 특히 스트리밍 서버의 부하가 많이 걸릴 때($\rho \rightarrow 1$) 과도한 버퍼 소모현상을 야기한다.

3.2 멀티미디어 파일 시스템에서 산발적 디스크 작업

그림 1은 세 개의 스트림 요청과 산발적으로 도착하는 입출력 요청을 함께 서비스하는 디스크 하부 시스템의 동작을 나타낸다. 크게 세 부분으로 나눌 수 있는데, 제일 왼쪽 부분은 각 스트림이 데이타를 공급받고 있는 상황을 나타내고 있다. 여기에서 매 라운드마다 각 스트림에 공급되는 데이타양은 각 스트림이 한 라운드에서 소비하는 데이타양보다 커야 된다.

그림 1의 가운데 부분은 디스크 하부 시스템의 동작을 확대한 것이다. 디스크 하부 시스템은 세 스트림에 필요한 데이타 블록을 읽음과 동시에 산발적인 입출력 요청도 서비스하고 있으며, 총 시간은 라운드의 길이인 T보다 작아야 한다. 여기에서 각 사각형의 면적은 데이타 블록의 양, 그리고 너비는 해당 데이타 블럭을 읽는데 걸리는 시간으로 볼 수 있다. 그림 1에서는 산발적으로 도착하는 입출력 요청이 멀티미디어 데이타를 검색하는 데 영향을 주지 않고 있는데, 각 라운드에는 이러한 요청을 처리할 수 있는 충분한 여백의 시간이 있다. 이러한 여백의 시간을 슬랙(slack)이라 하는데, 예상하지 못했던 산발적인 입출력 요청이 도착할 때 디스크는 각 라운드의 슬랙을 이용하여 그 요청들을 서비스하게 된다. 물론, 라운드내의 공백 길이에 비하여 산발적으로 도착한 입출력요청 서비스에 필요한 시간이 훨씬 클 경우에는 진행중인 스트림 요청을 정해진 시간까지 처리하지 못하거나, 산발적 디스크 입출력을 처리하는데 입출력 지연이 발생하는 등의 문제가 발생한다.

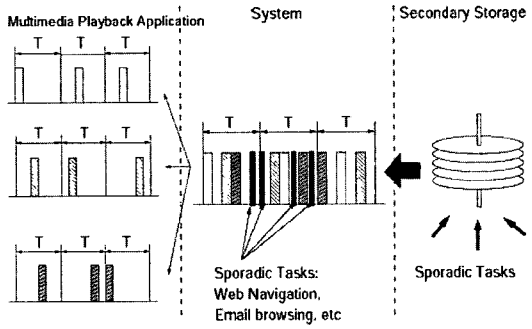


그림 1 산발적 디스크 작업이 있는 멀티미디어 세션의 지원

4. 통합형 파일시스템 구현

본 논문에서는 주기적으로 도착하는 멀티미디어 입출력 요청과 산발적으로 도착하는 비주기적인 입출력 요청을 효과적으로 처리할 수 있는 기법을 제안한다. 3절에서 살펴본 바와 같이 라운드 기반에서 슬랙을 사용하여 산발적인 요청을 처리할 수도 있지만, 이 기법들은 매우 복잡한 모델링을 필요로 하여 실제 시스템 구현에 있어서는 비효율적이다. 또한, Shenoy et al.이 제안했던 Cello 스케줄링 기법 등은 각 요청 클래스별로 해당되는 양만큼만 디스크 대역폭을 할당하기 때문에 실제로 많은 주기적인 멀티미디어 입출력 요청이 들어오게 되면 각 멀티미디어 스트림의 QoS를 보장할 수 없다. 이러한 이유들 때문에 여기에서는 라운드 기반의 스케줄링 방식을 사용하지 않고, 그보다 훨씬 간단하면서도 효율적인 몇 가지 구성요소들을 이용한 통합형 파일시스템 구현에 대해서 논의한다.

4.1 통합형 파일시스템 구조

본 논문에서는 응용 프로그램별로 다양한 요구사항을 충족시키기 위해 그림 2와 같은 구조를 제시한다. 이 그림을 살펴보면 크게 응용 프로그램 클래스, 수용제어 모듈 그리고, 디스크 I/O 스케줄러로 구성되어 있다.

기본 개념은 간단하다. 입출력 대기큐에 있는 요청들을 마감시간(deadline)을 기준으로 순서를 정한 후, 급한 마감시간을 가지고 있는 요청들을 먼저 서비스한다.

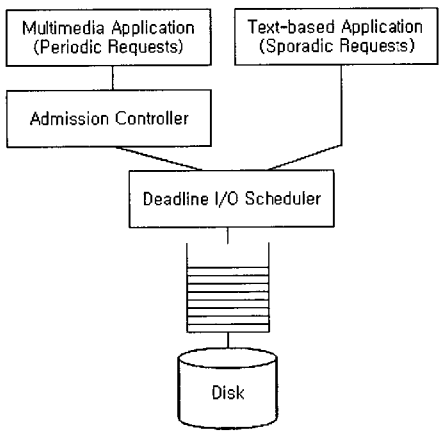


그림 2 QoS기반의 통합형 파일시스템 구조

만일 같은 마감시간을 가진 입출력 요청이 있다면 이들은 기존의 SCAN 알고리즘을 그대로 사용하여 디스크 오버헤드를 최소화하여 서비스한다. 따라서, 이처럼 입출력 대기큐에서 마감시간을 기준으로 서비스하려면 응용 프로그램 수준에서 마감시간을 계산한 후, 이 값을 커널의 여러 루틴을 거쳐 입출력 요청큐까지 전달해주면 된다. 그러기 위해서는 커널에서 마감시간 정보가 추가된 새로운 API를 제공하고, 이 API를 사용하는 멀티미디어 플레이어를 새로 구현해야 한다. 그리고, 이미 앞에서 언급하였듯이 멀티미디어 세션 요청이 많이 들어오는 경우에 기존의 서비스중인 스트림의 QoS 완전 보장을 위해서 수용 제어 모듈 또한 필요하다. 그림 2의 각 부분에 대해 좀더 살펴보기로 한다.

먼저 응용 프로그램 계층을 살펴보면 다음과 같다. 기존의 몇몇 연구에서는 응용 프로그램을 주기적인 요청, 비주기적 요청 그리고 인터랙티브 요청으로 나누었다[2,6]. 그러나 본 논문에서는 멀티미디어 스트리밍과 같은 주기적인(periodic) 요청과 간헐적으로 발생하는 텍스트나 이미지 기반의 산발적인(sporadic) 요청 두 가지로 구분한다. 비주기적인 요청과 인터랙티브 요청을 하나로 합친 이유는 실제로 I/O 수준에서는 이들 요청을 구분하기가 어렵기 때문이다. 물론 I/O 수준에서 요청 클래스를 구분하기 위해서 커널에 특정 API를 만들어서 제공할 수도 있으나 이는 이미 사용되고 있는 기존 응용 프로그램이 변경된 API를 사용해서 다시 작성되어져야 하는 것을 의미하기 때문에 현실적으로 불가능하다. 주기적인 요청도 다른 입출력 요청과 마찬가지로 커널 수준에서 구분하기 어렵지만, 주기적인 요청에 대해서는 이 요청을 따로 구분해서 처리할 수 있는 API를 제공하고 또 이 API를 사용하는 응용 프로그램을 제작하였다.

I/O 스케줄러는 Reddy et al.이 [10]에서 제안했던 SCAN-EDF와 유사한 스케줄링 기법을 사용한다. 멀티미디어 파일에 대한 요청이 있을 경우에는 각 입출력 요청마다 마감시간(deadline)을 가지고 있기 때문에 우선적으로 스케줄링을 하고, 이러한 시간적인 제약을 가지고 있지 않은 산발적인 요청들에 대해서는 디스크의 탐색 오버헤드를 가장 최소화하는 순서로 서비스를 한다. 이렇게 마감시간이라는 시간정보와 디스크상의 데이타 위치정보를 모두 고려하여 스케줄함으로써 다양한 응용프로그램들의 QoS 요구사항을 어느 정도 만족시킬 수 있다.

수용제어 모듈은 멀티미디어 파일에 대한 요청이 들어왔을 경우에만 수행되는 것으로 일반 멀티미디어 서버에서의 수용제어 기능과 같다. 일반적으로 멀티미디어 서버는 여러 사용자의 요청을 처리해야 함과 동시에 시스템 자원이 한정되어 있으므로 이미 서비스 중인 사용자의 실시간 입출력 요구를 보장하면서 새로운 사용자의 요구를 서비스할 수 있는지 검사할 필요가 있다. 이때 사용되는 것이 수용제어 모듈인데, 이는 처리방법에 따라 크게 결정적 수용제어(deterministic admission controller), 통계적 수용제어(statistical admission controller), 측정기반의 수용제어(measurement based admission controller) 등으로 구분할 수가 있다[11-15].

4.2 추가된 시스템 콜

우리는 멀티미디어 스트림의 QoS를 보장할 수 있는 통합형 파일시스템(QoSfs)을 리눅스 커널 2.4.20에 구현하였다. 앞에서 언급하였듯이 구현된 시스템은 주기적인 요청과 산발적인 요청 두 가지 클래스를 지원하고, 이를 I/O 수준에서 구분할 수 있도록 시스템 콜과 몇 가지 함수들을 추가하였다. 표 1은 추가한 시스템 콜에 관한 설명이다. QoSfs 파일시스템은 VFS 계층에 멀티미디어 스트림의 QoS를 위한 몇 가지 API가 추가된 것으로 볼 수 있는데, 따라서 현재 존재하는 어떠한 파일시스템(Ext2fs, NTFS, xfs 등)과도 함께 사용될 수 있다. 그리고, 기존의 리눅스에 탑재되어 있는 엘리베이터(elevator) 알고리즘을 마감시간 I/O 스케줄러(Deadline I/O Scheduler)로 대체하였고, 정확한 수용제어를 위해 시스템 관리자가 수용제어 관련 매개변수 값들을 설정, 변경할 수 있도록 하였다.

표 1 추가한 시스템 콜

시스템 콜	내 용
qosfs_open	수용제어모듈을 호출하고, 승인이 되면 멀티미디어 파일을 open
qosfs_close	멀티미디어 파일을 close하고, 수용제어관련 인자들을 reset
qosfs_read	deadline 값을 인자로 주어 멀티미디어 파일을 read
qosfs_admin	수용제어관련 인자들을 설정하는 시스템 콜

4.3 멀티미디어 플레이어 구현

우리는 커널에서 추가한 시스템 콜을 지원하기 위해서 mpeg2dec라는 공개된 소스코드를 수정하여 멀티미디어 플레이어를 구현하였다. 자세한 멀티미디어 플레이어의 흐름도는 그림 3과 같다. 먼저 main함수에서 handle_args함수를 호출함으로써 입력 인자를 분석, 제어하고, qosfs_open 함수를 이용하여 입력된 멀티미디어 파일을 open한다. open한 파일을 디코딩하기 위한 초기화 과정은 mpeg2_init 함수에서 수행된 후, 플레이어는 qosfs_read 함수를 이용하여 디스크에 기록되어 있는 데이타를 할당된 버퍼로 읽어 들인다.

$$deadline = \frac{BUFFERSIZE}{\sum_{i=currentframe}^{i=ips} framesize(i)} \times deadfactor \quad (3)$$

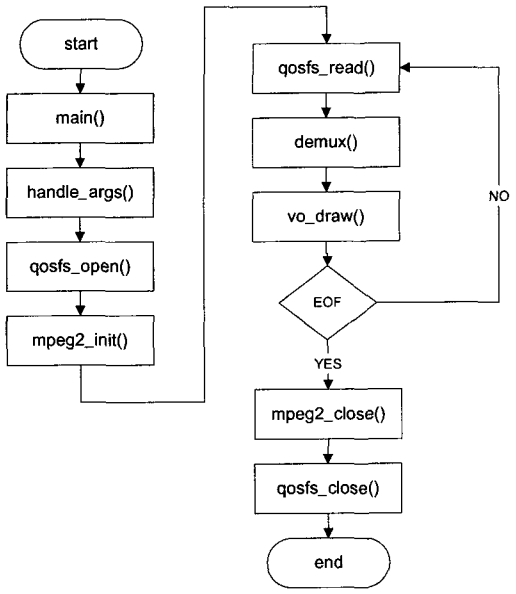


그림 3 멀티미디어 플레이어의 흐름도

본 논문에서 가장 중요한 인자인 마감시간(deadline)을 이 함수에서 사용하는데 마감시간은 식 (3)과 같이 계산된다. 여기에서 BUFFER_SIZE는 한 번의 요청 시 디스크에서 읽어올 데이타의 양을 나타내고, frame_size(i)는 i번째 프레임의 크기를 나타낸다. 또한 fps는 재생 시 필요한 초당 프레임 수이므로, 식 (3)의 분모는 결국 1초 동안 재생하기 위해 필요한 데이타를 바이트 단위로 나타낸 것이다. BUFFER_SIZE는 디스크로부터 공급되는 데이타의 단위이므로 식 (3)에서의 분수를 통해 애플리케이션에서 BUFFER_SIZE만큼 데이타를 읽을 때의 마감시간을 계산할 수 있다. 따라서, 이렇게 계산된 마감시간 내에 요청한 데이타를 읽지 못하면 화면에 끊김 현상이 발생하게 된다. 여기에 추가로 dead_factor를 주었는데, 애플리케이션에서 마감시간을 요청함에 있어서 부가적으로 발생하는 시간적인 오버헤드를 감안한 값으로 0에서 1사이의 값이다.

demux 함수는 버퍼에 저장된 데이타를 디코딩하고, 이렇게 디코딩된 데이타는 vo_draw 함수에 의해 화면에 동영상이 된다. 실제로 멀티미디어 파일을 상영하는 도중에는 이와 같은 과정이 계속해서 반복된다. 파일 재생이 끝나면 mpeg2_close 및 qosfs_close 함수에 의해 디코딩을 위해 사용했던 모든 변수 및 메모리가 제거된 후, 멀티미디어 재생 프로그램은 종료된다.

4.4 I/O 스케줄러

멀티미디어 플레이어에서 계산된 마감시간은 읽기 요청이 있을 때마다 디스크 입출력 대기큐까지 전달되어야 한다. 그러기 위해서는 리눅스 커널의 file, page, request, buffer 구조체에 마감시간 필드를 추가한 후, 이 필드들을 이용하여 몇 가지 커널 루틴을 거쳐 입출력 요청(request 구조체)까지 전달해 주어야 한다. 앞 절에서 설명된 멀티미디어 플레이어를 통해 요청한 입출력들의 마감시간 값은 플레이어에서 결정되지만, 일반 응용 프로그램에 의한 입출력 요청에는 마감시간 정보가 없으므로 무한대로 설정한다.

그림 4는 마감시간에 기반하여 스케줄링을 하는 코드를 간단히 도시화한 것이다. 일단 새로운 입출력 요청이 들어오면 그 요청을 SCAN 계열의 엘리베이터 알고리즘에 따라 기존의 요청과 합병 여부를 판별하여 가능하면 합병하고 그렇지 않으면 새로운 요청을 입출력 대기큐에 삽입한다. 그리고 나서, 마감시간에 따라 대기큐에 있는 모든 요청들을 새로 스케줄링을 한다. 이렇게 하면 가장 급한 마감시간을 가진 요청들이 먼저 서비스 되고, 그렇지 않은 요청들은 기존의 엘리베이터 알고리즘대로 서비스를 받게 된다.

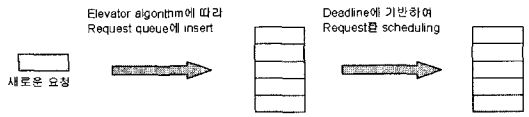


그림 4 마감시간 기반의 스케줄링 구현

4.5 수용제어 모듈

여기에서는 앞에서 언급한 여러 수용 제어 방법 중 최악의 상황을 고려 즉, 캐싱의 효과를 0으로 놓는다거나 또는 디스크의 탐색시간(seek time), 회전지연시간(rotation delay) 등을 최대로 가정한 뒤 수용 여부를 결정하는 방법인 결정적 수용제어를 사용한다[11,12]. 이는 최악의 상황을 가정하기 때문에 자원의 낭비가 심한 단점이 있지만, 간단할뿐더러 어떤 경우에서라도 끊김없는 스트림 재생을 보장할 수 있다(여기에서는 수용제어 모듈을 채택했다는데 의미를 둔다. 실제로는 어떠한 수용제어 기법으로 대체되어도 전혀 상관이 없다).

실제로 수용제어를 하기 위해서는 CPU, 메모리 버퍼, 디스크 대역폭,. 네트워크 대역폭 모두를 고려해야 하지만, 여기에서는 디스크 대역폭만을 고려한다. 디스크 대역폭만을 고려한 수용제어는 식 (4)처럼 간단히 정의할 수 있다. 즉, 요청한 디스크 대역폭(B_{req})이 가용 디스크 대역폭($B_{avail} = B_{total} - B_{used}$)보다 작으면 요청을 수용하고 그렇지 않으면 거부한다.

$$B_{used} + B_{req} \le B_{total} \quad (4)$$

여기에서는 대역폭을 시간 관점에서 계산을 한다. 전체 디스크 대역폭을 1000ms라고 하고, 요청한 디스크

대역폭을 다음의 식 (5)와 같이 계산한다. 여기에서 $B_{transferl}$는 실제로 데이타를 디스크로부터 읽는 데 소모되는 디스크 대역폭을 나타내고 $B_{overhead}$는 그 오버헤드를 나타내는 데, 각각 식 (6)과 식 (7)에 의해 계산된다. 식 (5)에서의 peak ratio는 1보다 큰 값으로 최악의 경우에도 기존의 서비스 중인 스트림의 QoS를 보장하기 위해 설정되는 보정 값이다(기본값으로 1.5를 사용하였다).

$$B_{req} = (B_{transfer} + B_{overhead}) \times peak\ ratio \quad (5)$$

$$B_{transfer} = \frac{bps}{8 \times \max tranferrate} \quad (6)$$

$$B_{overhead} = \frac{bps}{8 \times MAXSECTOR \times 512} \times (T_{seek} + T_{rotation}) \quad (7)$$

식 (6)에서 *bps*는 qosfs_open시 매개변수로 입력되는 값으로 재생하고자 하는 멀티미디어 파일이 요구하는 평균 *bps*를 가리키고, *maxtransferrate*는 디스크의 최대 전송율로 KB/s로 입력된다. 결국, 식 (6)은 요청한 멀티미디어 파일을 디스크의 최대 전송 속도로 읽을 때 필요한 시간을 나타낸다. 식 (7)은 디스크의 오버헤드로서, 탐색과 회전지연으로 발생하는 시간이다. 기본적으로 멀티미디어 파일은 디스크에서 순차적으로 배치되어 있을 가능성이 높기 때문에 각 입출력 요청 내의 섹터 개수를 커널이 허용하는 섹터 개수(*MAXSECTOR*)로 가정하였다. 그리고, 이러한 입출력 요청들끼리는 최악의 경우 모두 디스크 오버헤드가 발생할 수 있음을 가정하였다.

여기에서 사용된 *maxtransferrate*, T_{seek} 및 $T_{rotation}$는 디스크의 매뉴얼에서 제공되는 값으로, 소스 코드에서 이 값들을 설정하거나 앞에서 언급한 qosfs_admission 시스템 콜을 이용하여 재설정할 수도 있다(기본값으로 각각 100,000KB/s, 9ms, 5ms가 설정되어 있다). 표 2는 여기에 사용된 기호들의 자세한 설명이다.

5. 성능 평가

5.1 실험 방법 및 실험 환경

구현한 QoSfs 파일시스템의 성능을 평가하기 위해 엘리베이터 디스크 스케줄링을 사용하는 기존의 리눅스 시스템(Ext2fs)과 비교를 하였다. 각각 동일한 환경에서 멀티미디어 파일과 텍스트 파일을 동시에 요청하여 멀티미디어 스트림의 QoS가 보장되는지를 살펴보았다. 여기에서 사용된 멀티미디어 플레이어는 앞에서 언급한 mpeg2dec의 수정버전이고, 비실시간 요청으로는 파일시스템 벤치마크 툴인 iozone을 사용하였다[16].

수정된 mpeg2dec로 324MB 크기의 9Mbps 멀티미디어 파일을 30프레임/초로 읽게끔 하였고, 비실시간 요청을 위한 백그라운드 작업을 위해 iozone에 적절한 옵션을 주어 실험하였다. 하드웨어적인 실험환경은 다음과 같다.

CPU : INTEL Pentium Ⅳ 2.4 Ghz

RAM : 256 MBytes

DISK : SeaGate ST340016A / 40GB / 7,200RPM

5.2 실험 결과

여기에서 중점적으로 실험한 내용은 재생중인 멀티미디어 파일의 QoS가 텍스트 입출력 요청이 증가함에 따라 어떤 영향을 받는지에 대한 것이다. 즉, 9Mbps의 멀티미디어 파일을 재생하면서 백그라운드 작업을 증가시켜 나갈 때, 멀티미디어 파일의 마감시간이 얼마나 지켜지지 못하는지를 측정하였다. 여기에서 텍스트 클라이언트 1이라 함은 128MB의 파일을 4KB단위씩 쓰고 읽는 iozone 툴을 말한다.

그림 5는 텍스트 클라이언트의 개수를 증가함에 따라 마감시간 내에 서비스되지 못한 입출력 요청 비율을 도시화한 것이다. 즉, mpeg2dec 플레이어에서 요청을 할 때마다 마감시간 정보를 커널에 넘겨주고, 그 요청에 해당하는 데이타가 그 시간 내에 서비스되는지의 여부를 플레이어에서 측정하였다. 기존의 리눅스 시스템의 경우(Ext2fs)에는 백그라운드 클라이언트의 개수가 증가함에

표 2 수용제어 모듈에 사용된 기호 정의

기 호	정 의	비 고
B_{total}	총 사용가능한 디스크 대역폭 (=1000ms)	계산 가능
B_{used}	현재 사용중인 디스크 대역폭	계산 가능
B_{req}	요구하는 디스크 대역폭	계산 가능
$B_{transferl}$	데이타 전송하는데 필요한 대역폭	계산 가능
$B_{overhead}$	데이타 전송시 발생하는 오버헤드	계산 가능
bps	재생하고자 하는 데이타율(bit/sec)	qosfs_open시 매개변수로 입력됨
peak ratio	최악의 상황을 고려한 보정 값	1이상의 값
maxtransferrate	디스크 최대 전송 율	디스크 매뉴얼에서 제공
T_{seek}	디스트 평균 탐색 시간	디스크 매뉴얼에서 제공
$T_{rotation}$	디스크 평균 회전지연 시간	디스크 매뉴얼에서 제공
MAXSECTOR	커널의 입출력 요청 당 섹터 개수	커널 변수

따라 마감시간 내에 서비스되지 못한 입출력 요청들의 비율이 급격히 증가하고 있다. 멀티미디어 플레이어가 한 번에 읽어드리는 버퍼 크기에 따라 다르긴 하지만, 최악의 경우 80%까지 마감시간 미스가 발생하고 있다. 이것은 멀티미디어 재생시 끊김 현상이 매우 심하다는 것으로 다른 백그라운드 작업을 수행하면서 제대로 영화나 음악 감상을 할 수 없다는 것을 의미한다. 여기에서 버퍼 크기가 커짐에 따라 마감시간 미스율이 커지는데, 이는 미스율이 "미스된 요청 개수/총 요청한 개수"로 계산되어지므로 읽는 단위가 커질수록 분모가 작아지기 때문이다. 이에 반해 Qosfs의 경우에는 미스율이 0에 가까워 거의 끊김이 발생하지 않으므로 백그라운드 작업 수행이 영화나 음악 감상에 거의 영향을 주지 않는다고 볼 수 있다.

그림 6은 백그라운드 클라이언트 개수를 증가시키면서 초당 몇 프레임씩 멀티미디어 파일이 재생되는지에 대한 결과를 보여준다. 기존 리눅스 시스템의 경우에는 백그라운드 작업이 많아지면서 재생되는 프레임 수가 현저히 떨어지는 반면에 QoSfs는 백그라운드 작업에 상관없이 초당 30 프레임씩 성공적으로 재생하고 있음을 알 수 있다.

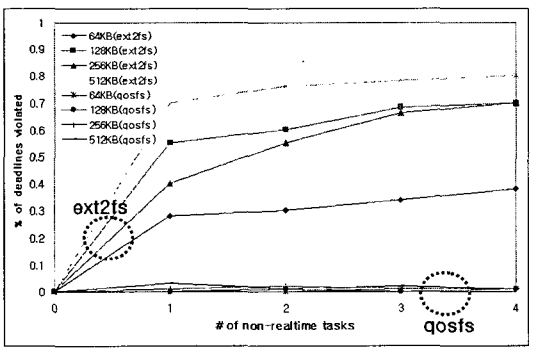


그림 5 백그라운드 프로세스 개수에 따른 마감시간 미스율 비교

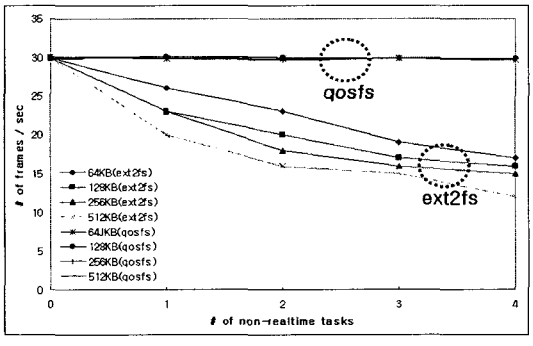


그림 6 백그라운드 프로세스에 따른 재생프레임 수/sec 비교

위의 실험들에서 알 수 있듯이 기존의 리눅스 시스템에서는 디스크 입출력 요청들을 따로 구분할 수 없어서 애플리케이션 각각의 요구사항을 만족시켜줄 수 없지만, QoSfs에서는 멀티미디어 파일의 실시간 요구사항을 만족시킬 수 있었다. 물론, 멀티미디어 요청에 우선권을 줌으로써 일반 텍스트 입출력 요청 서비스가 기존의 엘리베이터 스케줄링에 비해 지연되는 것은 사실이지만, 다양한 요구사항에 따라 입출력 요청을 구분해서 서비스를 해준다 것은 큰 의미가 있다. 구현된 파일시스템의 비디오 데모 클립은 [17]에서 찾을 수 있다.

6. 결 론

차세대 파일시스템으로서 통합형 파일시스템이 각광을 받으면서 하나의 디스크 파티션에서 다양한 요구사항을 지닌 입출력요청을 처리하는 것이 매우 중요한 이슈로 떠오르고 있다. 여기에서는 하나의 스토리지 서버 파티션에서 멀티미디어 스트림을 처리함과 동시에 간헐적으로 발생하는 비주기적인 요청들 즉, 웹 브라우징, 이미지 파일 처리, 데이타베이스 검색 등을 처리해야 하는 상황을 가정하고 있다. 이런 상황에서는 예기치 못한 비주기적인 입출력 요청들로 인해 기존의 서비스중인 스트리밍 서비스의 QoS를 만족시켜줄 수 없어서 동영상 재생시 화면의 끊김 현상이 발생한다.

본 논문에서는 몇 가지 구성요소들을 통합하여 멀티미디어 스트림의 QoS를 보장할 수 있는 시스템을 제시하였다. 기본적으로 응용프로그램들의 입출력 요청을 두 가지 클래스-주기적인 요청, 산발적인 요청-로 구분한 뒤 이를 입출력 처리 수준에서 주기적인 요청에 더 높은 우선순위를 부여하는 기법을 사용하였다. 또한, 이러한 메커니즘을 현재 널리 쓰이는 운영체제인 리눅스에 구현함으로써 그 효과를 검증하였다. 실험결과, 기존의 Ext2fs보다 본 연구에서 개발한 QoSfs가 멀티미디어 스트림의 QoS 보장에 있어서 훨씬 더 괜찮은 성능을 보였다.

참 고 문 헌

[1] Y. J. Won and Y. S. Ryu, "Handling Sporadic Tasks in Multimedia File System," Proceedings of the eighth ACM International conference on Multimedia, pp.462-464, October 2000.

[2] P. Shenoy, "Cello: A Disk Scheduling Framework for Next Generation Operating System," (extended version), Real Time Systems Journal, January 2002.

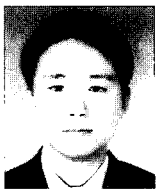
[3] V. Sundaram and P. Shenoy, "Application performance in the Qlinux Multimedia Operating System," Proceedings of the Eighth ACM Conference on Multimedia, Los Angels, CA, pages 127-136, November 2000.

[4] G. Nerjes, P. Muth, M. Paterakis, Y. Rom-

boyannakis, P. Triantafillous and G. Weikum, "Scheduling Strategies for Mixed Workloads in Multimedia Information Servers," 1998.
[5] G. Nerjes, P. Muth, M. Paterakis, Y. Romboyannakis, P. Triantafillous and G. Weikum, "Incremental Scheduling of Mixed Workloads in Multimedia Information Servers," Special Issue of the Journal of Multimedia Tools and Applications, 1999.
[6] R. Wijayaratne and A. L. Reddy, "Providing QoS guarantees or disk I/O," In Proceedings of ACM/Springer Journal on Multimedia Systems, January 2000.
[7] P. Shenoy, P. Goyal and H. Vin, "Architectural Considerations for Next Generation File Systems," (extended version), ACM/Springer Multimedia Systems Journal, 2002.
[8] P. Shenoy, S. Hasan, P. Kulkarni and K. Ramamritham, "Middleware versus Native OS Support: Architectural Considerations for Supporting Multimedia Applications," Proceedings of IEEE Real-time Technology and Applications Symposium (RTAS'02), San Jose, CA, September 2002.
[9] Y. J. Won and J. Srivastava, "smdp: Minimizing buffer requirements for continuous media servers," ACM/Springer Multimedia Systems Journal, 8(2): pp. 105-117, 2000.
[10] A. N. Reddy and J. C. Wyllie, "I/O Issues in a Multimedia System," IEEE Computer 27(3): 69-74 1994.
[11] D. P. Anderson, Y. Osawa and R. Govindan, "A File System for continuos Media," ACM Transactions on computer systems, 1992.
[12] H. M. Vin and P. V. Rangan, "Admission Control Algorithms for Multimedia-On-Demand-Servers," In Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, 1992.
[13] F. Y. S. Lin, "Optimal Real-Time Admission Control Algorithm for the Video-On-Demand (VOD) Service," IEEE Transactions on Broadcasting, 1998.
[14] H. M. Vin, P. Goyal and A. Goyal, "A Statistical Admission Control Algorithms for Multimedia Servers," In Proceedings of the ACM Multimedia, 1994.
[15] I. H. Kim, J. W. Kim, S. W. Lee, K. D. Chung, "Measurement-Based Adaptive Statistical Admission control Scheme for Video-On-Demand Servers," In Proceedings of the 15th International Conference on Information Networking.
[16] http://www.iozone.org
[17] http://www.dmclab.hanyang.ac.kr/research/project/hermes-q/hermes-q_overview.htm

김 태 석

2000년 서울대학교 전산과학과 이학사 2002년 서울대학교 컴퓨터공학부 공학석사. 2002년~현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 멀티미디어 파일시스템, QoS 시스템, 운영체제



박 경 민

2001년 홍익대학교 컴퓨터공학부 학사 2003년 서울대학교 전기컴퓨터공학부 컴퓨터공학부 공학석사. 2003년~현재(주) 삼성전자. 관심분야는 파일시스템, 임베디드 시스템

최 정 완

2000년 순천향대학교 제어계측공학과 학사. 2003년 한양대학교 전자통신전파공학과 석사. 현재 LG전자기술원. 관심분야는 모바일, 멀티미디어

김 두 한

정보과학회논문지 : 시스템 및 이론 제 31 권 제 5 호 참조

원 유 집

정보과학회논문지 : 시스템 및 이론 제 31 권 제 5 호 참조

고 건

정보과학회논문지 : 시스템 및 이론 제 31 권 제 2 호 참조

박 승 민

1981년 울산대학교 전자공학과 공학사 1983년 홍익대학교 전자공학과 공학석사 1983년~1984년 (주)LG전자. 1984년~현재 한국전자통신연구원 임베디드 S/W 기술센터 S/W 개발도구연구팀장, 책임연구원. 관심분야는 임베디드 S/W, 유비쿼터스 컴퓨팅, RTOS, 이동통신망

김 정 기

1992년 전북대학교 컴퓨터공학과 공학사 1994년 전북대학교 컴퓨터공학과 공학석사. 1999년 전북대학교 컴퓨터공학과 공학박사. 1996년~1998년 시스템공학연구소 연구원. 1998년~현재 한국전자통신연구원 임베디드 S/W 기술센터 선임연구원. 관심분야는 임베디드 S/W, 유비쿼터스 컴퓨팅, 병렬 정보검색