EXT3NS 파일 시스템을 위한 버퍼 캐시의 설계 및 구현

손성훈* · 정성욱**

Design and Implementation of Buffer Cache for EXT3NS File System

Sung-Hoon Sohn* · Sung-Wook Jung**

이 논문은 2006년도 상명대학교 일반연구기관 선발과제의 연구비를 지원 받았음

요 약

EXT3NS 파일 시스템은 Network-Storage Card(NS 카드)라는 전용 하드웨어를 기반으로 하는 멀티미디어 스트리밍 서버를 위한 파일 시스템이다. EXT3NS는 NS 카드 상의 PCI memory(PMEM)를 이용, 스트리밍 시 디스크에 있는 데이터를 메인 메모리를 거치지 않고 바로 네트워크 카드로 내보냄으로써 스트리밍 서버의 성능을 향상시킨다. 본 논문에서는 NS 카드에 있는 PMEM을 기반으로 한 버퍼 캐시를 추가 설계, 구현하고, 이 버퍼 캐시가 EXT3NS 파일 시스템을 채용한 서버의 멀티미디어 스트리밍의 성능을 개선시킬 수 있음을 보인다. 또한, 제안된 버퍼 캐시 상에서 다수의 동시 사용자를 지원하는 멀티미디어 스트리밍 서버를 위한 버퍼 캐시 교체 정책인 Old New Section(ONS) 교체 기법을 제안한다. 제안된 ONS 교체 기법은 멀티미디어 스트리밍 환경에서 기존의 교체 정책들 보다 좋은 성능을 보인다. 성능 평가 결과, 다시 읽기동작과 무작위 읽기 동작에서 기존의 EXT3NS 파일 시스템 보다 PMEM에 캐시를 사용한 EXT3NS가 각각 평균 33MB/sec, 2.4MB/sec의 성능이 향상 되었다. 또한, 다시 읽기 동작의 경우 기존의 Least Frequently Used(LFU) 교체 정책을 사용한 경우보다 ONS교체 정책을 사용한 경우가 약 600KB/sec의 성능 증가를 보인다. 이러한 결과는 동시에 여러 명의 사용자 요구를 처리해야 하는 대용량 멀티미디어 스트리밍 서버에서 보다 효과적인 읽기 동작을 처리 할 수 있음을 나타낸다.

ABSTRACT

EXT3NS is a special-purpose file system for large scale multimedia streaming servers. It is built on top of streaming acceleration hardware device called Network-Storage card. The EXT3NS file system significantly improves streaming performance by eliminating memory-to-memory copy operations, i.e. sending video/audio from disk directly to network interface with no main memory buffering. In this paper, we design and implement a buffer cache mechanism, called PMEMCACHE, for EXT3NS file system. We also propose a buffer cache replacement method called ONS for the buffer cache mechanism. The ONS algorithm outperforms other existing buffer replacement algorithms in distributed multimedia streaming environment. In EXT3NS with PMEMCACHE, 다시 읽기 operation is 33MB/sec and random read operation is 2.4MB/sec. Also, the buffer replacement ONS algorism shows better performance by 600KB/sec than other buffer cache replacement policies. As a result, PMEMCACHE and an ONS can greatly improve the performance of multimedia steaming server which should support multiple client requests at the same time.

키워드

EXT3NS, file system, buffer cache, replacement policy, multimedia streaming server

* 상명대학교 소프트웨어학부 조교수
** 상명대학교 대학원 컴퓨터과학과 석사과정

접수일자 : 2006. 10. 10

Ⅰ. 서 론

분산 멀티미디어 스트리밍 서비스에서 스트리밍 서버의 성능은 전체 스트리밍 서비스의 품질을 좌우하는 중요한 요소이다. 기존의 스트리밍 서버는 스트리밍 과정에서 디스크로부터 읽어 들인 데이터를 네트워크 인터페이스를 통해 전송하는 과정에서 서버 내부에서 여러 번의 메모리 간 복사가 발생하여, 이로 인한 성능 향상의 한계를 지니고 있다.

NS 카드는 대규모 스트리밍 서버의 성능을 향상시키기 위하여 고안된 장치이다. NS 카드는 서버의 스트리밍 연산 시 앞서 언급한 서버 내의 불필요한 메모리간 복사를 최소화하는 것을 목적으로 한다. 이를 위해 NS 카드는 디스크 제어기, 네트워크 인터페이스, 그리고 PMEM이라는 메모리 버퍼로 구성되어 있으며, 스트리밍 시 디스크에서 읽어 온 데이터를 주기억장치로 복사하지 않고 PMEM를 거쳐 바로 네트워크 인터페이스로 내보낼 수 있도록 설계 되었다.

EXT3NS 파일 시스템은 대용량 멀티미디어 파일의 입출력을 위한 파일 시스템으로, 기존 EXT3 파일 시스템을 기반으로 SDA 장치를 통한 입출력을 제공하도록 수정 설계된 파일 시스템이다. EXT3NS는 기존에 리눅스 커널이 정의한 파일 입출력 관련 시스템 호출 인터페이스를 그대로 사용하여 SDA 장치를 통한 입출력을 가능하게 한다. 또한, 사용자가 입출력 시 PMEM 을 버퍼로 사용할 수 있도록 PMEM 영역을 할당/해제하는 PMEM 관련 라이브러리를 제공한다. 따라서 고성능 입출력을 필요로 하는 스트리밍 응용 프로그램은 기존의 파일 입출력 인터페이스와 새로 제공되는 PMEM 라이브러리를 사용하여 SDA 장치를 통한 고성능 입출력이 가능하다.

기존 EXT3NS 는 PMEM 을 이용한 입출력 시 버퍼 캐시를 제공하지 않고 있다. 본 논문에서는 EXT3NS 파일 시스템에 PMEM을 이용한 버퍼 캐시를 설계하고 구현했다. 또한, 대용량 멀티미디어 스트리밍 환경에서 보다 좋은 성능을 얻기 위하여 사용자가 동시에 특정 파일을 사용하게 되는 멀티미디어 스트리밍 서버의 특징에 따라 버퍼 캐시의 교체 정책인 ONS 교체 정책을 새로 설계했다. 이러한 교체 정책의 추가는 읽기 동작의 경우 LFU 교체 정책을 사용한 경우보다 전체 입출력 대역폭 상 평균 1.5 MB/sec 이상의 성능 개선이 이루어 졌으며, 다시 읽기 동작의 경우 LFU 교체정책을 사용 한 경우보다 평균 600KB/sec 이상의 성능 개선이 이루어 졌다. 이러한 결과는 멀티미디어 스트리밍 서버에 여러 명의 동시 사용자가 특정 파일의 스트리밍을 동시에 요구 했을 경우 좋은 성능을 보여 줄 수 있다는 것을 보여 준다.

본 논문은 다음과 같이 구성되어 있다. 2 장에서는 NS 카드와 EXT3NS 파일 시스템에 대해 소개하고, 버퍼 캐시의 필요성에 대해 언급한다. 3 장에서는 본 논문에서 제안한 EXT3NS를 위한 버퍼 캐시인 PMEM 캐시에 대해 자세히 소개하고, 멀티미디어 스트리밍 환경을 위한 버퍼 캐시 교체 정책인 ONS 교체 정책에 대해 다룬다. 또한 4 장에서는 EXT3NS와 버퍼 캐시를 사용한 EXT3NS, 그리고 ONS 교체 정책에 대한 성능 시험 결과를 다룬다. 마지막으로 5장에서는 결론과 향후 과제에 대해서 알아본다.

Ⅱ. 관련 연구

이번 장에서는 멀티미디어 스트리밍 서버의 핵심 장치인 NS 카드와 EXT3NS 파일 시스템의 구조, EXT3NS 파일 시스템의 문제점에 대해서 알아 보겠다.

2.1. EXT3NS 파일 시스템

NS 카드는 네트워크를 통한 분산 멀티미디어 스트리밍 환경에서 스트리밍 서버의 성능을 향상시키기 위하여 고안된 장치이다[1].

EXT3NS 파일 시스템은 대용량 멀티미디어 파일의 입출력을 위한 파일 시스템으로, 기존 EXT3 파일 시스템을 기반으로 NS 카드의 SDA 장치를 통한 입출력을 제공하도록 수정 설계된 파일 시스템이다. EXT3NS는 기존에 리눅스 커널이 정의한 파일 입출력 관련 시스템 호출 인터페이스를 그대로 사용하여 SDA 장치를 통한 입출력을 가능하게 한다. 또한, 사용자가 입출력 시 PMEM 을 버퍼로 사용할 수 있도록 PMEM 영역을 할당/해제하는 PMEM 관련 라이브러리를 제공한다. 따라서 고성능 입출력을 필요로 하는 스트리밍 응용 프로그램은 기존의 파일 입출력 인터페이스와 새로 제공되는 PMEM 라이브러리를 사용하여 SDA 장치를 통한 고성능 입출력이 가능하다.

기본적으로 EXT3NS 는 SDA 드라이버가 미리 정의한 256 KB ~ 2 MB 의 블록 크기를 단위로 데이터를 저장한다. 또한 입출력 시 버퍼로 사용되는 PMEM 영역 또한 이 블록 크기에 맞춰 할당이 이루어진다. 반면, 리눅스 커널

이 수용할 수 있는 파일 시스템의 최대 디스크 블록 크기는 시스템 페이지 크기인 4 KB 에 불과하다. 따라서 EXT3NS 의 큰 블록 사이즈는 높은 입출력 성능은 보장할 수 있으나, EXT3NS 를 리눅스 커널 내의 가상 파일 시스템 계층 (virtual file system layer) 아래의 한 파일 시스템으로 존재하게 하기 어렵게 만든다. 이는 결과적으로 파일을 사용하는 기존 응용 프로그램들이 수정 없이 EXT3NS 내의 파일을 사용할 수 없음을 의미한다. 이러한 문제점을 해결하기 위해 EXT3NS 는 기존 리눅스 커널 의 페이지 캐시 입출력 (page cache I/O) 을 동시에 지원한다. 이 경우의 동작은 기존 EXT3 파일 시스템과 거의 유사하며, 사용자는 기존 파일 접근 응용 프로그램이나 cp, mv 등과 같은 기본적인 리눅스의 파일 관련 명령어들을 그대로 사용할 수 있다.

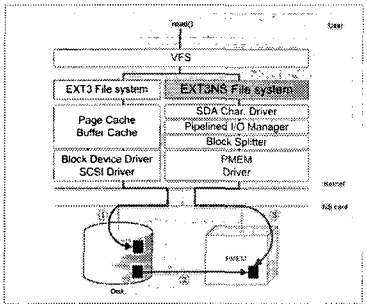

그림 1. EXT3NS 파일 시스템 동작
Fig. 1. Operation of EXT3NS file system

그림 1 은 EXT3NS 파일 시스템의 동작에 대해서 보여주고 있다. 응용 프로그램이 EXT3NS 파일에 대한 입출력 시스템 호출을 요청하면, 이는 VFS 계층을 거쳐 EXT3NS 파일 시스템 계층으로 온다. EXT3NS 파일 시스템 모듈은 사용자가 제공한 읽기 또는 쓰기 버퍼가 메인 메모리 영역인지 PMEM 영역인지 구분하여, 이에 따라 기존의 페이지 캐시 입출력을 사용할 것인지, SDA 입출력을 사용할 것인지를 구분한다. 메인 메모리 버퍼인 경우 기존 페이지 캐시 입출력과 동일한 동작이 일어나고, SDA 입출력인 경우 SDA char 디바이스 드라이버 인터페이스를 통해 PMEM 을 버퍼로 하는 디스크 입출력을 수행한다.

일반적인 EXT3 파일 시스템은 512, 1024, 2048, 4096단위의 블록 사이즈 만을 제공한다. 이것은 SMART 서버 구조[1]에서 제공하는 256KB~2MB단위의 블록 사이즈를 지원할 수 없다. EXT3NS 파일 시스템은 256KB~2MB의 블록 입출력을 제공한다. 하지만, 사용자가 EXT3NS파일 시스템에 기존의 EXT3 파일 시스템에서 사용하는 블록 사이즈를 사용해야 할 경우가 발생한다. 특정 파일에 대한 메타 데이터 등이 이러한 경우에 속하게 되는데 이 경우에는 EXT3NS의 256KB~2MB 단위의 블록 사이즈를 사용하는 것보다 기존의 EXT3 파일 시스템의 구조를 그대로 쓰게 하는 것이 효율 적이다. 따라서, EXT3NS 파일 시스템은 기존의 EXT3 파일 시스템의 구조를 그대로 사용할 수 있게 했다. 4KB단위의 입출력을 하게 될 경우 EXT3NS 파일 시스템은 기존에 EXT3 파일 시스템이 제공하는 페이지 캐시와 버퍼 캐시를 그대로 사용할 수 있다.

2.2. EXT3NS 의 문제점

EXT3NS 파일 시스템은 기존의 EXT3 파일 시스템의 기능을 그대로 사용하면서 256KB~2MB로 블록 사이즈를 조정할 수 있게 만들었기 때문에 4KB단위의 기본적인 입출력의 경우는 EXT3 파일 시스템이 제공하는 버퍼 캐시를 사용하게 된다. 그러나, EXT3 파일 시스템이 제공하는 버퍼 캐시의 블록 사이즈는 4KB이므로 256KB~2MB의 입출력에서는 버퍼 캐시를 사용할 수 없다. 따라서, 본 논문에서는 EXT3NS 파일 시스템에 버퍼 캐시를 구현 하는 것을 목적으로 하고 있다.

디스크로부터 하나의 블록을 읽어 오는 것은 메모리로부터 하나의 블록을 읽어 오는 것보다 많은 시간이 소비된다. 따라서, 매번 디스크로부터 블록을 읽어 오는 것은 컴퓨터 입장에서 많은 자원의 낭비를 가져 오게 되는 것이다. 따라서, 캐시라는 개념이 나왔다. 느린 장치와 보다 빠른 장치의 중간에 캐시라는 것을 두어 두 장치의 속도의 차이를 조금이나마 줄여 보려는 노력이다. 디스크 캐시는 버퍼 캐시라고도 불려지며, 이러한 버퍼 캐시의 사용은 메모리와 메모리 사이의 이동으로 속도가 빠를 뿐 아니라 버퍼 캐시에 있는 블록은 다른 프로세스들이 입출력을 원할 경우에도 사용할 수 있기 때문에 매우 효율 적이다. 이런 버퍼 캐시를 설계 함에 있어서 대두되는 문제는 버퍼 캐시의 공간이 전부 사용 되었을 경우 특정 블록에 대한 수정 문제이다. 특정 블록을 수정 함에 있어서 오랫동안 사용되지 않은 블록이 아닌 다음에 사용될 가능성이 높은 블록을 먼저 수정하게 되는 경우는 다음 입출력 시 디스크에서 다시 읽어와야 하므로 매우 비효율 적이라 볼 수 있기 때문에 최대한 효율적으로 블록을 선택해서

수정해야 하는 문제를 가지고 있다.

이러한 문제를 버퍼 캐시의 교체 정책이라 한다. 가장 많이 연구 되고 사용하는 버퍼 캐시의 교체 정책은 대략 3가지를 들 수 있다. 첫 번째로 First In First Out(FIFO)이다. FIFO는 말 그대로 가장 처음에 버퍼 캐시에 들어온 블록을 가장 먼저 수정하는 것이다. 이것은 구현이 간단하다는 장점을 가지만, 계속해서 입출력이 되는 블록이 있더라도 시간이 지나 가장 오래된 블록이 되면 자동적으로 캐시에서 수정되므로 비효율 적이라 볼 수 있다. 두 번째는 Least Recently Used(LRU)이다. LRU는 말 그대로 가장 오랫동안 사용하지 않은 블록을 교체 하는 것이다. 이것은 스택을 사용하며, 가장 최근에 사용되는 되는 블록은 계속해서 스택의 탑에 쌓게 된다. 계속해서 이런 현상이 발생 되다 보면, 스택의 최하위에 있는 블록은 가장 오랫동안 사용되지 않은 블록이 되며, 버퍼 캐시가 차서 수정이 필요할 경우 스택의 최하위 블록을 수정한다. 마지막으로 세 번째는 LFU이다. 이것은 가장 사용 빈도가 낮은 블록을 교체 하는 방법으로 버퍼 캐시에 올라온 블록은 count를 가지고 있고, 매번 버퍼 캐시에 블록이 참조 될 때마다 count는 증가 한다. 버퍼 캐시가 차서 블록의 수정이 필요하면, count값을 비교하여 가장 적은 count인 블록을 버퍼 캐시에서 수정한다. 이러한 3가지 중 현재는 LFU가 가장 많이 사용되고 있으며, 다른 교체 정책보다 효율성이 높은 것으로 연구 되었다.

또한, 최근까지 연구된 교체정책은 교체 할 블록을 선택하는 조건에 따라 크게 두 가지로 분류 할 수 있다. 그 첫 번째로는 블록이 버퍼에 있었던 시간을 가지고 교체할 블록을 선택하는 방법으로 LRU, Segmented Least Recently Used(SLRU), LRU-K, 2Q등의 알고리즘이 있다. 두 번째로는 블록이 참조된 횟수를 가지고 교체할 블록을 선택하는 방법으로 LFU와 Frequency Based-Replacement(FBR) 등이 있다. Ramakrishna Karedla and J.Spencer Love and Bradley G. Wherry.는 SLRU 교체 정책을 제안 했다[6]. 이는 버퍼를 두 개의 Segment인 Probationary Segment와 Protection Segment로 나누고 각각의 Probationary Segment에는 처음 참조된 블록들이 들어간다. 그 후 Probationary Segment에 있는 블록이 재 참조 될 시에는 그 블록은 Protection Segment로 이동하고, 이렇게 이동된 블록은 오랫동안 버퍼 캐시에 존재 한다. 버퍼 캐시에서의 블록 교체는 Probationary Segment에서 가장 오래된 블록을 삭제하고 새로운 블록을 삽입한다. 2Q 교체 정책은 SLRU 교체 정책과 비슷한 정책으로 버퍼 캐시를 A1과 Am으로 분리 한 후 처음 참조되는 블록들은 A1에 넣고 블록에 대한 포인터만을 가진 후 교체 시킨다[8]. 이후 그 블록에 대한 재 참조가 되면 해당 블록은 Am에 넣는다. 또한, 연속적인 참조를 해결하기 위해서 A1을 두 개의 부분인 A1in, A1out으로 나누고 A1in에서 재 참조 되는 블록은 연속된 블록으로 판단하여 Am으로 이동시키지 않으며, A1out에서 참조될 경우에만 Am으로 이동시킨다. 2Q 교체 정책은 LRU-K를 변형 시킨 알고리즘으로 LRU-K에 비해 시간 복잡도가 줄어 들었다는 장점이 있다. John T. Robinson과 Murthy V. Devarakonda는 FBR을 제안했다[4]. 이 교체 정책은 기본적인 LRU 교체 정책을 사용하면서 이러한 캐시 범위를 new sectiony와 middle section, old section로 구분했다. 또한, 각 블록에는 Reference Count를 두어 블록을 교체해야 할 필요가 있을 때, old section에 있는 블록 중에서 Reference Count가 가장 작은 블록을 수정하는 방식을 고안했다. new section에서 참조가 일어날 경우에는 연관된 참조라 생각하여 Reference Count를 증가 시키지 안아 Reference Count는 실제로 middle section과 old section에서만 증가한다. ALAN JAY SMITH는 버퍼 캐시의 분석 방법 및 구현에 있어서 고려 해야 하는 사항들에 대해서 연구 했다[3]. ALAN JAY SMITH는 3가지의 서로 다른 특성을 가진 서버에서 캐시는 어느 정도의 사이즈가 적당한지, 캐시의 위치는 어디가 좋은지, Block size는 어느 정도가 적당한지, 알고리즘은 어떤 것이 가장 좋은지, 캐시는 한 개만 존재 하는 것이 좋은 것인지 등에 대한 캐시의 miss ratio분석을 했다.

Ⅲ. EXT3NS 파일시스템의 PMEMCAHCE의 설계 및 구현

이번 장에서는 EXT3NS 파일 시스템에서 PMEMCACHE의 자료구조 및 동작과 멀티미디어 스트리밍 서버를 위한 ONS 교체 정책의 동작에 대해서 설명한다.

3.1. PMEMCACHE 의 개요

PMEMCACHE 는 디스크에서 읽어와 NS 카드의 PMEM 이라는 메모리에 미리 존재 하는 블록을 재사용하기 위한 EXT3NS 파일 시스템의 버퍼 캐시를 의미 하는 것이다. PMEM 은 NS 카드의 읽기 동작의 특징 때문에 사

용하는데, 사용자가 NS 카드에 달려있는 디스크에서 특정 블록의 읽기를 파일 시스템에 요구 하면, 파일 시스템은 디스크에서 특정 블록을 읽어 PMEM 에 특정 영역에 넣는다. 이러한 PMEM 의 특정 영역은 사용자가 read 시스템 콜을 사용하여 읽기 동작을 하기 전에 미리 할당 받은 공간으로 메인 메모리와 같은 방식으로 사용할 수 있게 설계 되어 있다. 따라서, 버퍼 캐시가 구현 되어 있지 않은 EXT3NS 파일 시스템은 사용자의 읽기 요구에 매번 디스크에서 PMEM 으로 사용자가 요구한 블록을 읽어 와야 하는 낭비가 있었다. 따라서, 커널의 EXT3NS 파일 시스템을 수정하여 PMEMCACHE 라는 리스트 구조체를 만들고 매번 읽기 동작 전에 이 리스트를 확인 함으로써 미리 PMEM 에 읽어온 블록을 재사용 할 수 있게 했다. 이러한 PMEM CACHE 의 구성은 그림 2 과 같다. 사용자는 read 시스템 콜을 사용하여 특정 블록을 요구하게 된다(1번). 이러한 읽기 동작은 VFS 레이어를 거쳐 EXT3NS 파일 시스템으로 전달된다. EXT3NS 파일 시스템은 PmemList 에서 해당 블록이 있는 지를 확인하고 있을 경우 해당 블록에 대한 포인터를 반환한다(2번).

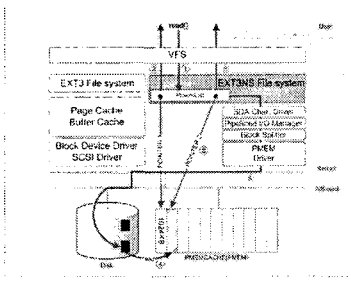

그림 2. PMEMCACHE의 동작
Fig. 2. Operation of PMEMCACHE

블록이 없을 경우 EXT3NS 파일 시스템은 NS 카드의 여러 드라이버들을 거쳐 디스크에서 PMEM 으로 해당 블록을 읽어오고(4번), PmemList 에 해당 포인터를 추가한다(5번). 그 후 해당 포인터를 반환한다(6번).

3.2. 자료구조

PMEMCACHE 의 자료구조는 아래 그림 4 과 같다. PMEMCACHE 는 리스트 구조로 이루어 져있는 구조체들의 집합이다. 이러한 리스트의 제일 처음을 pmemList 라는 포인터가 가르키고 이것은 PMEMCACHE 를 접근하기 위한 방법으로 사용한다. 리스트의 크기는 512개로 제한 되어 있으며, 구조체의 개수가 PMEMCACHE 의 리스트에 제한된 개수를 넘어가면 교체 정책에 의해 블록을 삭제 하고 새로운 블록을 삽입한다. 리스트에 매달리게 되는 구조체의 멤버는 다음과 같은 구성을 가지고 있다. 리스트 구조를 유지 하기 위한 이중 연결 리스트의 포인터, 현재 구조체가 가리키고 있는 블록 번호, 블록 번호에 해당하는 실제 PMEM 의 포인터, 블록을 사용한 횟수 등이 그것이다. 또한, 현재 PMEMCACHE 에 가지고 있는 구조체의 개수를 가지고 있는 변수를 두어 PMEMCACHE 의 현재 크기를 확인한다. 이러한 자료 구조를 유지 하기 위해 몇 가지 기본적인 함수들을 구현 했다. 하는 블록이 있는지를 확인하는 검색 함수, 새로 디스크에서 읽어 들인 블록을 PMEMCACHE 의 리스트에 삽입 하기 위한 함수, 현재 PMEMCACHE 의 리스트를 삭제 하기 위한 함수, 원구조체 멤버의 PMEM 에 대한 포인터가 사용자가 넘겨준 포인터와 일치 하는지를 확인하는 함수, 만약, 포인터와 일치 하지 않을 경우 사용자가 넘겨준 함수의 포인터를 수정하는 함수 등이 가장 기본이 되는 함수라 할 수 있다. 여기에 교체 정책의 알고리즘 별로 특성에 맞는 함수를 추가하여 구현 했다.

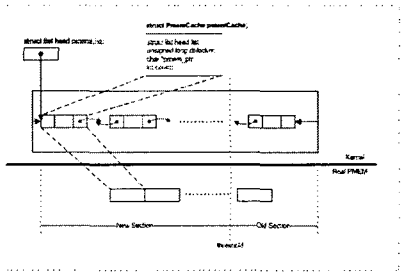

그림 3. PMEMCACHE의 구성
Fig. 3. Structure of PMEMCACHE

이러한 PMEMCAHCE 는 그림 3 의 기본적인 구조를 유지하면서 원하는 교체 정책을 쉽게 사용할 수 있게 설계 했다.

3.2. PMEMCACHE 의 설계

PMEMCACHE 의 각 블록들은 다음의 조건을 만족하면서 캐시에 매달린다. PMEM 지역에서 읽기 동작이 일어나 NS 카드의 장치에서 특정 블록을 PMEM 영역으로 읽어 오려 하면, 그 전에 EXT3NS 의 PmemCache 는 현재 커널의 PMEM 에 해당하는 캐시 리스트에서 읽어올 블록

이 있는지 확인한다. 만약 있을 경우 EXT3NS 는 해당 PMEM 의 포인터를 리턴 하고 끝난다. 만약 없을 경우에는 리스트의 개수가 최대 PMEM 리스트의 개수 보다 크거나 같은지 확인하고 작으면, 리스트의 마지막에 현재 블록에 해당하는 구조체를 매단다. 그러나 만약 리스트의 개수가 최대 PMEM 리스트의 개수 보다 크거나 같으면 정해진 교체 정책에 의해서 블록을 삭제 하고 새롭게 매단다. 여기서 PMEM 의 리스트에 매달게 되는 구조체는 몇 가지 정보를 가지고 있어야 한다. 그 첫 번째로 리스트 구조를 이루고 있으므로 리스트의 다음과 바로 전 노드에 대한 포인터를 들고 있어야 하며, 두 번째로 현재 어플리케이션에 의해 읽어 오도록 불려진 블록 번호를 가지고 있어야 하고, 세 번째로 현재 블록에 해당하는 PMEM 의 포인터의 위치를 가지고 있어야 한다. 네 번째로는 해당 블록이 사용된 횟수를 저장하기 위한 변수가 필요하다.

위와 같은 동작 이후 리스트에 해당 블록이 있는데 블록의 해당 PMEM 포인터가 사용자에 의해 어플리케이션에서 넘어온 포인터와 같지 않을 경우, 사용자에서 넘어온 포인터의 위치를 현재 블록에 해당하는 PMEM 포인터로 변경해 주어야 한다. 이를 위해 사용자는 포인터의 포인터를 어플리케이션에서 커널로 넘겨주어야 한다. 이러한 사실은 보통의 read 시스템 콜을 사용하여 읽기 동작을 구현 할 경우 커널의 기본 구성을 바꾸어야 하는 문제가 따른다. 따라서 함수 인자가 보다 자유로운 ioctl 시스템 콜을 사용하여 PMEMCAHCE 를 사용하는 일기 함수를 구현 하고자 했다.

그리고 나중을 생각하여, 최대 PMEM 리스트의 개수는 변경 가능 해야 하고, 리스트의 추가 및 삭제에 해당하는 구조는 변경이 용이하게 설계 했다. 현재 구현 하여 테스트 되는 교체 알고리즘은 FIFO, LRU, LFU 그리고, 새로 개발한 ONS 교체 정책 이다.

그러나, 쓰기 동작의 경우에는 먼저 PMEM 을 할당하고 PMEM 에 쓰기 동작을 한 후에 디스크에 쓰는 방식을 사용하므로 버퍼 캐시의 구현이 어렵다. 따라서, 이 논문에서는 쓰기 동작에 버퍼 캐시 구현을 하지 않기로 하고, 쓰기 동작을 하기 전에는 PMEM 의 리스트에 매달려있는 모든 구조체 들을 삭제 하도록 구현했다.

3.3. ONS 교체 정책

대용량 멀티미디어 스트리밍 서버는 여러 명의 동시 사용자를 위한 서버이다. 이러한 대용량 멀티미디어 스트리밍 서버는 동시에 여러 사용자가 하나의 프로세스 혹은 쓰레드를 가지게 되고 각각의 사용자가 스트리밍을 원한다. 따라서, 본 논문에서는 기존의 LRU 와 LFU 를 혼합하고 개선한 새로운 교체 알고리즘인 ONS 교체 정책을 개발 했다.

새로운 교체 알고리즘인 ONS 교체 정책은 다음과 같은 구조를 이루고 있다. 읽기에 사용된 블록들을 매달게 되는 캐시의 영역은 두 부분으로 이루어져 있다. 첫 번째 영역은 NEW 영역으로서 처음에 디스크에서 읽게 되면 들어오게 되는 공간이다. 이 영역에서 하나의 블록이 나가게 되는 경우는 첫째, 해당 블록이 다른 어떤 프로세스에 의해서 다시 접근 되지 않고, 캐시가 꽉 차서 새로운 블록을 캐시에 넣어야 하는 상황에서 해당 블록이 캐시에서 나가게 된다. 둘째, 해당 영역블록이 미리 정해져 있는 NEW 영역의 캐시 적중 경계 값 보다 캐시의 적중이 많이 되었을 경우 OLD 으로 옮겨지게 된다. 여기서 두 번째 경우는 블록이 들어와서 일정 횟수 이상 사용되고, 더 이상 사용되지 않는데 캐시에 더 머물게 되는 현상을 최대한 줄이기 위한 방법이다. 두 번째 영역인 OLD 영역은 NEW 영역에서 캐시 적중 경계 값을 넘어간 블록들만이 들어오는 것으로 이 블록들은 앞으로 더 사용될 가능성이 있다고 보고, OLD 영역에서 더 머물게 한다. 이러한 NEW 영역과 OLD 영역의 크기는 기본적으로 NEW 영역을 OLD 영역보다 크게 하여 기존의 캐시의 성능을 유지 시켰고, NEW 영역의 캐시 적중 경계 값 은 테스트에 의해 가장 최적의 값으로 설정 했다. 또한, 각각의 영역이 더 이상 블록을 받을 수 없는 상태가 되었을 때, 블록을 교체하는 알고리즘은 LRU 교체 정책과 LFU 교체 정책을 동시에 사용하도록 했다. 기본적으로 LFU 교체 정책을 사용하여 가장 캐시 적중 횟수가 적은 블록 중에서 LRU 교체 정책에 의해 가장 오래 있었던 블록을 삭제 한다.

이러한 ONS 교체 정책을 PMEMCACHE 에 적용하기 위하여 기존의 리스트를 검색하는 함수는 함수의 결과 값으로 블록의 현재 위치가 NEW 영역인지 OLD 영역인지를 확인하고, 해당하는 번호인 1 또는 2를 리턴 하도록 만들었다. 또한, NEW 영역과 OLD 영역의 버퍼 캐시가 꽉 찼을 경우 특정 블록을 삭제 하기 위한 함수를 구현했으며, 사용자가 요청한 블록이 PMEMCACHE 의 NEW 영역이나 OLD 영역에 있어 버퍼 캐시에서 적중 되었을 경우 정해진 알고리즘에 따라 블록을 움직이기 위한 함수를 구현했다. 여기서 정해진 알고리즘이란 PMEMCAHCE 를

검색 하여 블록이 NEW 영역에 위치하면서 블록의 캐시 적중 횟수가 캐시 적중 경계 값보다 큰 경우, 블록이 NEW 영역에 위치하면서 블록의 캐시 적중 횟수가 캐시 적중 경계 값보다 작은 경우, 블록이 OLD 영역에 위치 하는 경우까지 3가지 경우에 대해 첫 번째 경우는 버퍼 캐시의 구조체 정보를 OLD 영역으로 넘기고, 두 번째 경우 NEW 영역의 최상위로 버퍼 캐시의 구조체 정보를 옮기고, 세 번째 경우 OLD 영역의 최상위로 버퍼캐시의 구조체 정보를 옮기는 알고리즘을 이야기 한다.

이러한 함수들의 구현을 통하여 ONS 교체 정책을 구현 했고, 대용량 멀티미디어 스트리밍 서버에 적합한 알고리즘임을 테스트를 통해서 알게 됐다.

Ⅳ. 성능평가

본 논문은 성능평가를 위하여 NS 카드를 장착한 특별한 서버를 사용했다. 이 서버는 Intel Xeon 3.0GHz CPU 4개, 1GB의 메인 메모리와 NS 카드에 연결되어 있는 73GB의 스카시 하드 디스크 8개를 가지고 있다. 운영체제는 Fedora Core 3 Kernel 2.6.10을 사용했고, IOZONE 의 버전은 3.217 이다. IOZONE 의 경우 3개의 다른 프로그램을 만들었다. 아무런 수정이 되지 않은 원본과 기존의 IOZONE 을 수정하여 PMEM 을 사용할 수 있게 수정한 IOZONE, 그리고 ONS 교체 정책을 위하여 IOZONE 의 읽기 동작 부분을 read 시스템 콜이 아닌 ioctl 시스템 콜로 수정한 IOZONE 을 만들었다. 이러한 IOZONE 의 읽기 동작, 다시 읽기 동작, 무작위 읽기 동작의 의미는 다음과 같다. 읽기 동작은 이미 존재 하는 파일에 대한 읽기 성능을 계산 하는 것이고, 다시 읽기는 최근에 읽었던 파일을 다시 읽을 경우에 대한 테스트를 수행 한 것이다. 또한, 무작위 읽기 동작의 경우는 파일을 순차적으로 접근 하는 것이 아니라 무작위로 접근하여 읽기 동작의 성능을 테스트 하는 것이다.

4.1. 캐시를 사용하지 않은 EXT3NS 파일 시스템과 캐시를 사용한 EXT3NS 파일 시스템의 비교

대용량 멀티미디어 스트리밍 서버는 여러 사용자의 요구를 동시에 들어 주어야 하는 특성을 가지고 있다. 이러한 특성을 가지고 있는 멀티미디어 스트리밍 서버에 맞는 성능 테스트를 위해 쓰레드의 개수를 10개에서 50개까지 변화 시키면서 여러 개의 쓰레드가 수행 되면서 동시에 하나의 파일에 입출력을 하는 상황을 가정하여 워크로드를 만들고 이러한 워크로드에 대한 성능 평가를 위하여 IOZONE 이라는 파일 시스템 벤치 마크 프로그램을 사용했다. IOZONE 프로그램은 여러 개의 쓰레드를 만들어 여러 개의 입출력 동작에 대해서 테스트를 가능하게 했다.

먼저 EXT3NS 를 그냥 사용하는 것보다 캐시를 사용하는 것이 효율적 임을 보이기 위하여, 캐시의 교체 정책 중 대표적인 알고리즘인 LFU 교체 정책과 캐시를 사용하지 않는 EXT3NS 파일 시스템에 대해서 이러한 IOZONE 을 사용하여 한 개의 쓰레드가 1GB 의 파일을 1MB 단위로 입출력 하게 하고, 쓰레드의 개수를 5가지의 경우(10, 20, 30, 40 50쓰레드)로 변화 주어 테스트 했다.

그림 4 는 각 쓰레드 개수 별 데이터 전송 전체 속도에 대한 그래프 이다. 이 그래프는 여러 개의 쓰레드를 동시에 수행 했을 경우 버퍼 캐시를 사용한 경우가 버퍼 캐시를 사용하지 않은 일반적 경우에 비해 성능 향상을 가져왔음을 알 수 있다. 아래 표 1을 보면 LFU 교체 정책을 사용하여 버퍼 캐시를 구현한 EXT3NS 가 다시 읽기 동작 시 평균 33MB의 성능 증대를 가져 왔고, 무작위 읽기 동작 시에는 평균 2.4MB의 성능 증대를 가져 왔음을 볼 수 있다. 이렇듯 버퍼 캐시를 사용하는 것은 사용하지 않는 것보다 성능 향상에 도움이 되는 것을 알 수 있다.

표 1. EXT3NS vs LFU 알고리즘을 사용한 EXT3NS의 성능증가

Table 1. Effect of LFU on the performance of EXT3NS

쓰레드 개수	10	20	30	40	50	평균
다시 읽기 (MB)	88.064	28.672	24.576	11.264	15.36	33.5872
Random read (MB)	-9.216	3.072	5.12	12.288	1.024	2.4576

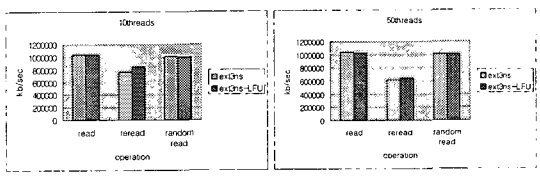

그림 4. EXT3NS vs LFU 알고리즘을 사용한 EXT3NS의 성능 비교

Fig. 4. The performance of LFU replacement policy

4.2. ONS 교체 정책과 다른 여러 교체 정책의 성능 비교

4.1절에서 비교한 EXT3NS 파일 시스템과 LFU 를 사용한 EXT3NS 파일 시스템의 분석은 버퍼 캐시의 필요성을 보여 준다. 하지만, 이러한 버퍼 캐시의 성능은 상황에 따라서 어떠한 교체 정책을 사용하느냐가 영향을 미칠 수 있다. 따라서, 네트워크 스트리밍 서버를 위해 새로 개발한 ONS 교체 정책 교체 정책의 성능을 확인하기 위하여 대표적인 성능 분석 알고리즘인 FIFO, LRU, LFU 와 함께 읽기 동작과 다시 읽기 동작에 대해서 성능 분석을 했다.

그림 5 는 각각의 알고리즘에 대한 성능을 보여주고 있다. 다시 읽기 동작의 경우 ONS 알고리즘이 쓰레드의 개수가 증가해도 계속해서 높은 성능을 보여 주고 있다. 또한, 읽기 동작의 경우는 거의 대부분의 경우에 좋은 성능을 보여 주고 있다. 읽기 동작의 40 쓰레드는 다른 교체 정책들 보다 약 2MB정도의 성능 개선이 이루어 졌으며, 다시 읽기 동작의 경우 LRU 교체 정책과 비교하여 약 600Kbyte 의 성능 개선을 보여준다. 이러한 성능 개선은 여러 명의 동시 사용자가 같은 동영상의 스트리밍을 요구 했을 경우 보다 좋은 성능을 지원해 줄 수 있을 것으로 기대 된다.

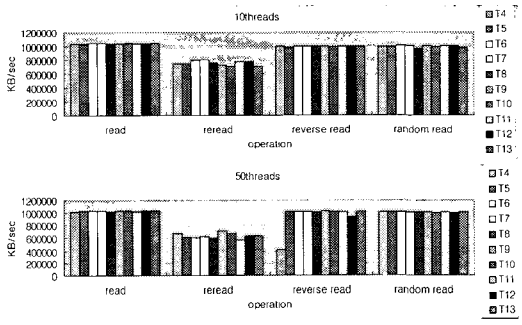

그림 5. 교체 정책 성능 비교
Fig. 5. Comparison on the performance of various replacement policies

4.3. ONS 교체 정책 교체 정책의 경계값별 성능 비교

새로 개발한 ONS 교체 정책 교체 정책은 그 값의 변화에 따라 성능에 영향을 미칠 수 있는 것으로 NEW 영역에서 OLD 영역으로 가기 위한 경계값이 있다. 이 경계값은 NEW영역에 있는 블록의 버퍼 캐시 구조체의 적중 횟수를 비교하여 경계값 이상이 되면 OLD영역으로 옮기기 위해 사용된다. 따라서, 경계값 4에서부터 13까지 총 10가지 경우에 대해서 성능을 비교하여 가장 좋은 성능을 보인 값을 5.2의 테스트에 사용했으며, 결과는 그림 6 과 같다.

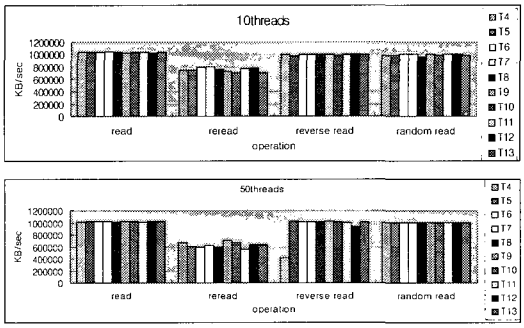

그림 6. Threshold 별 쓰레드 개수 비교
Fig. 6. Threshold versus the number of concurrent threads

4.4. NS 알고리즘을 사용한 ext3ns파일 시스템과 다른 파일시스템의 CPU사용률

그림 7은 디스크 입출력시의 CPU 사용률을 비교 했다. 이를 위해 버퍼 캐시에 ONS 교체 정책을 사용한 EXT3NS 파일 시스템과 버퍼 캐시를 사용하지 않은 EXT3NS 의 파일 시스템 그리고, EXT3 파일시스템에 쓰레드 개수를 10 개부터 30개까지 변화시키면서 전송률과 CPU 사용률을 테스트 했다. 먼저 EXT3 파일 시스템은 전송속도가 다른 두 개의 파일 시스템 보다 반절 정도 떨어 졌다. 하지만 CPU사용률은 각각의 동작에서 다른 두 개의 파일 시스템 보다 높았다. 또한, 버퍼 캐시에 ONS 교체 정책을 사용한 EXT3NS 파일 시스템은 버퍼 캐시를 사용하지 않은 EXT3NS 파일 시스템 보다 무작위 읽기 동작에서 2.5%의 성능 증가를 보여 주었다. 그러나 CPU 사용률은 2.5%의 성능증가를 보인 무작위 읽기 동작에서 5%정도의 증가를 보였다. 이러한 결과는 버퍼 캐시의 ONS 교체 정책을 사용하기 위하여 CPU 를 사용하게 되는 것으로 보인다.

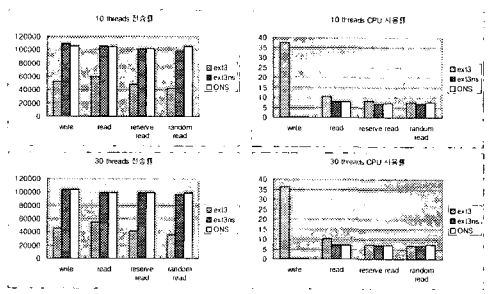

그림 7. 쓰레드 개수 별 전송률과 CPU 사용률
Fig. 7. The number of concurrent threads versus transfer rate and CPU utilization

결국, ONS 교체 정책은 서버가 여유롭게 사용할 수 있는 CPU 사용률을 사용함으로써 전송속도의 증가를 가져 올 수 있었다. 이는 여러 명의 사용자의 요구를 만족시켜야 하는 멀티미디어 스트리밍 서버에서 큰 이점이 된다고 볼 수 있다. 또한, 서버의 남는 CPU 를 사용하여 전송속도를 개선했다는 점에서 이점이 있다.

4.5. ONS 알고리즘을 사용한 PMEMCACHE 의 적중률

다음 그림 8은 ONS 교체 정책을 사용한 PMEMCAHCE 에서 테스트한 각 쓰레드별 임계값에 따른 적중률이다. 그래프를 보면 10개 쓰레드일 경우 임계값 7과 10을 제외하고는 거의 비슷한 버퍼 캐시 적중률을 보여 주고 있으며, 20개 스레드일 경우에는 5, 6 그리고 7의 임계값에서 높은 적중률을 보여 주고 있다. 또, 30개 쓰레드일 경우에는 1과 5의 임계값에서 높은 적중률을 보여 주고 있다. 이러한 결과는 임계값 5일 경우 각 쓰레드의 개수에서 가장 높은 캐시 적중률을 보여 주는 것을 볼 수 있다.

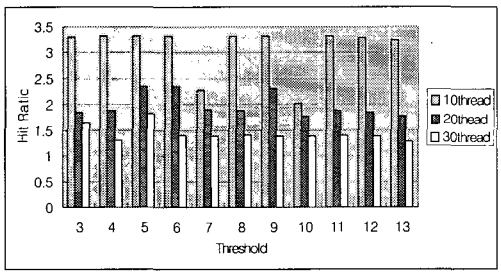

그림 8. PMEMCACHE의 쓰레드별 Hit Raito
Fig. 9. The number of concurrent threads versus hit ratio

V. 결 론

본 논문에서는 기존의 EXT3NS 파일 시스템을 수정하여 PMEM 에 버퍼 캐시를 구현하여 PMEM 만을 사용한 기존의 EXT3NS 파일 시스템보다 성능 개선을 이루었다. 버퍼 캐시를 사용한 파일 시스템이 사용하지 않은 파일 시스템 보다 다시 읽기 동작의 경우 평균 33MB의 성능 개선이 이루어 졌으며, 무작위 읽기 동작의 경우 평균 2.4MB/sec 이상의 성능 개선이 이루어 졌다. 또한, 대표적인 알고리즘인 LFU 교체 정책을 사용한 파일 시스템과 ONS 교체 정책을 사용한 파일시스템의 테스트에서는 ONS 교체 정책을 사용한 파일 시스템이 LFU 교체 정책을 사용한 파일 시스템보다 읽기 동작의 경우 40 쓰레드에서 약 2MB/sec, 다시 읽기 동작에서 약 600kb/sec의 성능 개선을 보인다. 이러한 결과는 멀티미디어 스트리밍 서버에 여러 명의 동시 사용자가 특정 파일의 스트리밍을 동시에 요구 했을 경우 좋은 성능을 보여 줄 수 있다는 것을 보여 주었다. 여기에 CPU 사용률 조사를 통하여 무작위 읽기 동작에서 약 5%의 CPU 사용을 통하여 버퍼 캐시를 사용하지 않은 기존의 EXT3NS 파일 시스템 보다 2.5% 성능 증대를 가져옴을 알 수 있다. 이러한 결과는 기존의 파일 시스템 보다 CPU 를 사용하여 캐시를 사용하는 것이 시스템의 남는 자원을 이용하여 성능을 개선하는데 좋은 성과를 보임을 알 수 있다.

하지만, 읽기 동작의 경우 구현이 될 수 있었던 캐시는 쓰기 동작의 경우에는 먼저 PMEM 을 할당 받고 PMEM 에 쓰기를 하여 kernel로 넘기는 문제에 의해서 쓰기 동작에 대한 버퍼 캐시가 구현 되지 못하는 문제가 있었다. 이러한 문제점에 의해 쓰기 동작이 구현 되지 못했으며, 앞으로는 계속해서 쓰기 동작에 대한 버퍼 캐시가 이루어 질 수 있도록 연구해 나갈 것이다.

참고문헌

[1] Baik-Song Ahn, Sung-Hoon Sohn, Chei-Yol Kim, Gyu-Il Cha, Yun-Cheol Baek, Sung-In Jung, Myung-Joon Kim, Implementation and Evaluation of EXT3NS Multimedia File System, Proceedings of the 12th annual ACM international conference on Multimedia, p588-595, July 2004.

[2] William Stallings, "Operating Systems Internal and Design Principles", Prentice Hall, Fifth Edition, 2004.

[3] Alan Jay Smith: Disk Cache, Miss Ratio Analysis and Design Considerations, ACM Trans. Comput. Syst. 3(3), p161-203 , 1985.

[4] J. T. Robinson and M. V. Devarakonda, Data cache management using frequency-based replacement, Proceedings of ACM SIGMETRICS Conf., p134-142, 1990.

[5] Elizabeth (Betty) O'Neil, Gerhard Weikum, An Optimality Proof of the LRU-K Page Replacement

Algorithm, Journal of the ACM, p92-112, January 1999.

[6] R. Karedla, J. S. Love and B. G. Wherry, Caching Strategies to Improve Disk System Performance, IEEE Computer, Vol.27, No. 3, p38-46, March 1994.

[7] 전홍석, 노삼혁, 선반입을 이용한 효율적인 버퍼 캐쉬 관리 알고리즘, 정보과학회 시스템 및 이론 제27권 제5호, May 2000.

[8] Theodore Johnson and Dennis Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, In Proceedings of the 20th VLDB Conference, pages 439-450, 1994.

저자소개

손 성 훈(SungHoon Sohn)

1991년 서울대학교 계산통계학과 졸업 (학사)
1993년 서울대학교 전산과학과 졸업 (석사)
1999년 서울대학교 전산과학과 졸업 (박사)
1999년 ~ 2004년 한국전자통신연구원 선임연구원
2004년 ~ 현재 상명대학교 소프트웨어학부 조교수
※관심분야: 저 전력 임베디드 소프트웨어, 임베디드 리눅스 운영체제, 멀티미디어 시스템 등

정 성 욱(SungWook Jung)

2005년 상명대학교 소프트웨어학부 졸업 (학사)
2005년 ~ 현재 상명대학교 일 반대학원 컴퓨터과학과 재학 (석사과정)
※관심분야: 저 전력 임베디드 소프트웨어, 임베디드 리눅스 운영체제, 멀티미디어 시스템 등