□특집□

리눅스에서의 멀티미디어 지원 기술

김 정 원† 김 인 환†† 이 승 원†† 정 기 동†††

◈ 목 차 ◈

1. 서 론

최근 인터넷, VOD(Video on Demand) 등 멀티미디어 데이터에 대한 요구가 증가하고 있다. 본질적으로 멀티미디어 데이터는 대용량성과 고대역폭을 요구한다. 예를 들어 MPEG 2 데이터의 경우 1시간을 기준으로 2GB이상의 저장공간과 4~60Mbps의 대역폭을 요구한다[1]. Unix와 같은 운영체제는 평균 수십에서 수백 킬로바이트 크기의 파일을 고려하여 파일시스템이 작성되었다. 따라서, 멀티미디어 데이터를 효율적으로 지원할 수 없다. 한편, 리눅스는 엔터프라이즈 시장이나 학계, 연구계에서 무료 운영체제라는 점과 Unix 호환이며 월등한 안정성으로 인해 주목받고 있다. 본 논문에서는 리눅스상에서 멀티미디어를 효율적으로 지원하기 위해 고려해야할 기반 기술들을 논하고자 한다. 즉, 리눅스의 멀티미디어 지원을 위해 필요한 멀티미디어 파일시스템(MFS : multimedia file system)의 요소 기술, 버퍼 관리 기법 및 프로세스 스케줄링 등을 다룬다. 본 논문의 구성은 다음과 같다. 2 장에서는 멀티미디어 파일의 특징을 먼저 살펴보고, 3장에서 리눅스에서 멀티미디어 파일을 지원하기 위한 기반 기술들을 소개한다. 마지막으로 4장에서 결론을 맺는다.

† 정회원 : 부산대학교 전자계산학과 박사수료
†† 정회원 : 부산대학교 전자계산학과 박사과정
††† 정회원 : 부산대학교 정보컴퓨터공학부 교수

2. 멀티미디어 파일의 특징[2]

저장장치는 프로세스, 네트워크 속도의 발전에 크게 미치지 못한다. 이러한 차이를 극복하기 위해서 새로운 저장장치나 파일시스템적인 측면에서의 저장(storage) 및 검색(retrieval)구조에 대한 연구가 활발히 진행되고 있다[3]. 멀티미디어 파일은 텍스트와 같은 데이터에 비해 다음과 같은 특징을 가지고 있다.

- 실시간(real time) : 멀티미디어는 마감시간(deadline)안에 프리젠테이션 되어야 하는데, 허용되는 지터(jitter)는 적어야만 한다. 따라서, 저장과 검색관련 알고리즘은 이 시간의 제약성을 고려해야하고, 데이터 스트림(stream)의 전송을 평활화(smoothing)시키기 위한 추가의 버퍼가 요구된다.
- 파일크기 : 작고, 비구조적(unstructured)인 텍스트 데이터에 비해 멀티미디어는 대용량이고 구조적이므로 제한된 저장 공간을 효율적으로 사용하기 위해서 멀티미디어 데이터 블록은 디스크 상에서 효율적으로 배치되어야만 한다.
- 다중 데이터 스트림 : 동시에 상이한 미디

어를 처리해야 하므로, 모든 스트림에게 충분한 자원을 공급해야할 뿐만 아니라, 미디어들 사이에도 동기화를 제공해야 한다.

3. 멀티미디어 지원을 위한 기반 기술

본 장에서는 리눅스 운영체제가 멀티미디어를 지원하는데 필요한 기반 기술과 관련 연구들을 살펴본다.

3.1 대용량 파일 지원

Alpha 및 Ultra SPARC 플랫폼은 64비트의 주소체계를 가지므로 2GB이상의 파일을 이미 지원하고 있다. 그러나 대다수의 리눅스가 탑재되는 인텔 플랫폼은 32비트 주소체계를 가지므로 최대 2GB까지의 파일만 저장할 수 있다. 리눅스 ext2 파일시스템의 inode의 *i_size* 필드는 long 형의 데이터이므로 2GB의 데이터 오프셋을 유지한다. 한편, GNU C 컴파일러는 long long 데이터 타입을 지원하므로 64비트 오프셋이 가능하다. 따라서, 리눅스 운영체제의 파일시스템과 가상메모리 구조가 이를 지원해야 한다.

또한 Unix의 ufs(unix file system), ffs(fast file system)와 같은 전통적인 파일시스템의 block map 구조는 three-step indirection 구조이다. 이 구조는 파일의 크기가 크기 않은 텍스트의 경우는 three-step indirection을 통과할 경우가 적으므로 유리하지만 MPEG과 같은 대규모의 동영상 파일의 경우는 빈번하게 double indirection 이상을 요구하게 되므로 block map의 오버헤드가 증가한다. 리눅스 ext2 파일시스템의 inode 구조체의 i_node 필드는 15개의 4바이트 크기 정수형이다. 이중 12개는 direct block이고, 나머지 3개는 one-stage, two-stage, three_stage indirect block이다. 따라서, 리눅스의 MFS가 대용량 파일을 지원하기 위해서는 block map의 오버헤드를 감소와 large block을 지원해야 한다. 예를 들어, 인접 블록을 (시작 블록번호, 개수)로 표현함으로써, block map의 오버헤드를 감소시킬 수 있다. 또한, 현재 리눅스 커널 버전 2.0 이후로 읽기 연산시는 file-oriented memory page caching 기법을 사용하고 쓰기 연산시는 block buffer caching을 사용한다[4]. 따라서 파일시스템의 블록 크기는 가상 메모리의 페이지 크기에 제한된다. 버전 2.2의 경우, 4KB의 페이지 크기를 지원하므로 파일시스템의 한 블록의 크기도 최대 4KB이다. 일반적으로 연속미디어를 지원하기 위해서는 large block size가 파일시스템의 메타데이터 처리의 오버헤드와 디스크 스케줄링의 탐색 시간을 감소시킬 수 있으므로 효율적이다. 이를 지원하기 위해서는 64비트 주소체계를 지원해야 하고 커널의 페이지도 4KB이상이 되어야 한다.

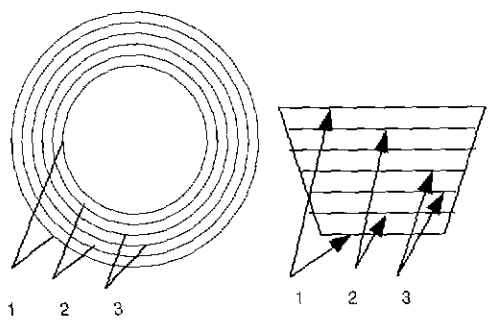
3.2 디스크 특성과 배치 기법

〈표 4〉 Seagate 디스크의 존당 전송률과 저장 공간

zone	0	1	2	3	4	~	19	20	21	22
size (MB)	140.0	67.0	45.0	46.0	44.0		37.0	25.0	34.0	20.0
전송률 (MB/s)	4.17	4.08	4.02	3.88	3.83		2.62	2.53	2.43	2.33

마그네틱 디스크의 저장 공간은 해마다 25% 이상 증가하고 있다[5]. 대부분의 멀티미디어 파일은 현재 가격대비 속도가 월등한 디스크에서 저장되고 검색된다. 파일시스템은 물리적인 디스크 공간을 논리적인 공간으로 매핑하므로 디스크의 특성을 반영하는 것이 바람직하다. 종래의 디스크는 CAD(constant angular density)로 모든 실린더의 전송률과 용량이 동일하였다. 그러나, 최근 디스크 저장 기술의 발달로 CLD(constant linear density) 디스크가 등장하였는데 모든 실린더의 저

장 밀도가 동일하다. 따라서, 각 실린더마다RPM (revolution per minute)이 동일하므로 VBR(variable bit rates) 특성을 나타내고, 안쪽 실린더는 바깥쪽 실린더보다 적은 저장공간과 전송률(transfer rates)을 지닌다. <표 1>은 Seagate ST31200W 디스크의 특성을 나타낸다. 동일한 저장공간과 전송률을 지닌 실린더를 존(zone)이라고 하는데 <표 1>에서 보듯이 존 0가 존 22보다 두 배의 전송률과 저장공간을 가진다. 따라서, 단일 존을 가지는 CAD 디스크 상에서 연속미디어를 배치하는 [6, 7, 8]과 같은 기법들은 ZBR(zone bit recording)디스크에 적용할 때 디스크 자원의 낭비를 가져올 수 있다. 이를 해결하기 위한 여러 가지 연구 결과는 다음과 같다.



(그림 1) Track-paring

[9]의 Track-paring 기법에서는 (그림 1)에서와 같이 전송률이 높은 외곽쪽 실린더와 전송률이 낮은 내곽쪽 실린더를 그룹화하고 있다. 이 기법은 응용 프로그램에게 데이터 블록이 저장된 존에 상관없이 동일한 대역폭을 보장하는데 있다.

[10]에서는 연속미디어의 접근 패턴에 따라 인기도가 높은 파일부터 ZBR 디스크의 바깥쪽 존에서 안쪽 존으로 배치하는 선형배치(linear placement) 기법 및 접근 확률이 높은 섹터 기준으로 중간 섹터부터 접근 확률이 높은 파일을 배치하는 SH(skewed heuristic)기법을 제안하였다.

3.3 파일의 구조와 디스크 배치

전통적인 파일시스템의 주요 목적은 내부 및 외부단편화(internal and external fragmentation)를 최소화하여 저장 공간을 효율적으로 사용하고, 파일의 삽입과 삭제시 오버헤드를 감소시키는 것이었다[2]. 일반적으로 파일의 내부 단편화의 경우, 마지막 블록이 평균 반이 낭비되는 것으로 알려져 있다. 따라서, 파일시스템의 블록 크기가 클수록 내부 단편화로 인한 저장 공간의 낭비도 커진다. 외부 단편화는 한 파일에 속한 블록들을 연속적으로 저장할 때 주로 발생한다. 파일의 삭제로 발생한 파일과 파일 사이의 갭에는 갭의 크기와 같거나 작은 크기의 파일만 채워질 수 있다. 따라서, 연속 미디어 파일의 경우, 두 파일사이의 사용되지 않는 공간의 낭비가 일반 텍스트 파일보다 크다.

MFS에서 멀티미디어 데이터의 실시간 전송은 파일의 구조에 따른 배치와 밀접한 관계가 있는데 두 가지 접근법을 고려할 수 있다. 첫째는 스트림을 위한 충분한 버퍼와 실시간 디스크 스케줄링의 제공이다. 이 방법은 데이터 블록이 분산(scatter)되어서 외부 단편화를 방지할 수 있고, 디스크 캐싱의 효과가 크다. 그러나, 블록이 분산되어 있으므로 스트림 전송시 과도한 탐색시간이 발생할 수 있고, 이를 방지하기 위해 충분한 버퍼가 제공되어야 한다. 또한, 초기 지연(initial delay)도 길어진다.

둘째는 스트림의 저장과 검색(retrieval)시간을 감소하기 위해서 미디어 블록을 조직적으로 배치하는 것이다. 연속미디어는 write-once -read-many의 특성을 보이고, 동시에 저장된 스트림은 동시에 재생될 가능성이 높기 때문에 연속적으로 배치하는 것이 타당하다. 이 방법은 동시에 읽혀질 파일들이 디스크의 물리적 공간에 연속적으로 저장되므로 버퍼 요구량과 탐색 시간이 감소된다. 그러나 이 방법의 문제점은 연속 저장으로 인한

외부단편화와 삽입 및 삭제시 발생하는 오버헤드가 크다는 것이다.

[11]에서는 블록의 크기(M)와 블록들 사이의 갭(G)은 연속 재생을 위한 요구조건에 의해 계산되어 질 수 있음을 보인다.

$$T_{play}(s) \geq \frac{M(\text{sec tors}) + G(\text{sec tors})}{r_{at}(\text{sec tor/s})} \quad \text{(식1)}$$

(식 1)은 미디어 블록(s 개의 섹터)을 읽고 갭을 스킵 하는데 걸리는 시간이 s개의 섹터를 재생하는 시간 $T_{play}(s)$을 초과하지 않으면 연속 재생이 가능함을 의미한다. (식 1)에서 M과 G는 변수인데 가능한 패턴(M, G)은 다수 존재할 수 있다. 즉 위의 식을 만족하는 여러 개의 조합이 가능하다. 여기서 문제는 각 스트림이 요구하는 저장 조건 (M, G)이 항상 만족될 수는 없다는 것이다. 이는 특정 개수의 블록을 선반입하거나 버퍼링을 통해서 만족시킬 수 있다.

3.4 버퍼 캐쉬

파일시스템이 버퍼 캐시를 사용하는 목적은 참조될 데이터 블록을 버퍼 캐쉬 영역에 미리 읽어 두어 디스크 입출력 시간을 감소시키고, 사용자 프로그램에게 신속한 응답시간을 보장하기 위해서이다.

리눅스의 버퍼 캐시 시스템은 Maurice J. Bach의 Unix용 버퍼 캐시 개념을 기반으로 하고 있다[12]. 따라서, 평균 수십에서 수백 킬로바이트의 텍스트 데이터를 위해서 설계된 캐시 시스템이라고 할 수 있는데, 버퍼 캐시 입장에서 멀티미디어 파일은 다음과 같은 특징을 가지므로 MFS의 버퍼 캐시 시스템은 이를 고려해서 설계 및 구현되어야 한다.

- 연속적으로 접근되므로 재사용(reusability)될 가능성이 적다 - 연속 미디어는 한번 읽은 블록의 재사용 가능성이 적으므로 리눅스의 LRU(least recently used) 버퍼 재배치(replacement) 정책은 버퍼 적중률(hit ratio)이 낮다.
- 미디어 데이터는 스트림의 특성에 따라 소비율이 다양하며 높다 -리눅스 버퍼 캐시는 미리읽기(read ahead)를 지원하는데 데이터 소비율이 높은 스트림은 단위 시간당 버퍼 요구량이 높다. 따라서, 소비율이 낮은 스트림은 미리읽기로 읽어둔 버퍼를 소비율이 높은 스트림에게 반환할 가능이 매우 높아진다. 이는 전역적인 버퍼 리스트 관리에 기인한다. MFS는 각 미디어 스트림이 요구하는 데이터 소비율에 적응할 수 있는 버퍼 관리 기법을 도입해야 한다.
- 연속적인 재생을 위해서는 일정한 지연시간(delay)을 보장해야 한다. 멀티미디어 시스템에는 동시에 다수의 스트림이 존재하는데 각 스트림이 요구하는 블록을 마감시간(deadline) 내에 서비스해야 한다. MFS는 여러 스트림의 요구 중에서 어떤 파일의 블록을 먼저 서비스할 것인가를 결정해야 한다. 한편, 동시에 요구를 받더라도 읽혀진 블록이 실제 재생되는 시점은 다를 수 있으므로 디스크 스케줄링시 이를 반영해야 한다.

위의 세 가지 고려사항을 바탕으로한 MFS의 버퍼 캐시 시스템의 구현에는 크게 두 가지가 있을 수 있다. 첫 번째는 기존의 버퍼 정책을 수정하는 것이고 두 번째는 버퍼 캐시 시스템을 사용하지 않는 것이다.

첫 번째는 버퍼와 적절한 디스크 스케줄링의 지원으로 멀티미디어 스트림을 지원하는 것이다. 이에는 크게 slack time을 이용하는 방법과 버퍼 소비량을 이용하는 방법이 있다. slack time은 스트림이 한 주기 상에서 버퍼를 소비하고 남는 여유시간이다. 다수의 스트림중 slack time이 가장

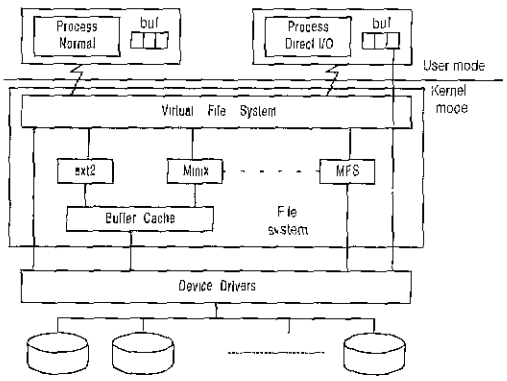
작은 스트림부터 스케줄하는 것이다[13, 14, 15]. 버퍼 소비량을 기준으로 하는 기법은 각 스트림의 주기당 버퍼 소비량을 최소로 되게 스케줄링하여 남는 버퍼를 효율적으로 사용하는 기법이다[16]. 또한 버퍼의 크기는 한정되어 있으므로 정적으로 할당하는 방법과 동적으로 할당하는 방법이 있다. [13, 14, 17]은 파일의 오픈에서 종료될 때까지 버퍼의 크기가 고정되는 방법인데 MPEG과 같은 VBR 요구를 수용할 수 없다. [18, 19]는 각 파일마다 주기당 필요한 블록의 수를 메타 데이터로 관리하여, 주기마다 스트림의 버퍼를 재할당 하는 방법이다. 이 기법은 버퍼 사용률은 최적화 되지만, 각 스트림마다 메타데이터를 유지해야 하는 부담이 있다.

두 번째, 버퍼 캐시를 스킵 하는 정책은 멀티미디어 데이터가 재사용 가능성이 적고, 데이터블록의 복사가 시스템 내부적으로 과도하게 발생하여 버퍼 캐시의 실용성이 낮다는 인식에서 나온 결과이다. 이를 구현하는 데에는 크게 direct I/O와 zero copy의 두 기법이 있을 수 있다.

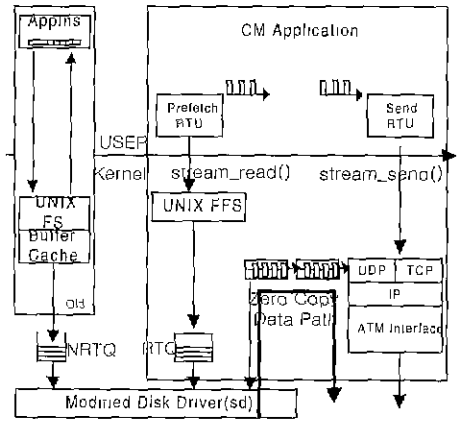
direct I/O는 리눅스의 버퍼 캐시 시스템을 스킵하여 직접 사용자 버퍼의 주소 공간으로 데이터를 복사하는 방법이다. (그림 2)에서 보듯이 기존의 ext2 파일시스템에서는 디바이스 드라이브는 요구받은 섹터를 커널이 유지하는 버퍼위치로 일단 복사를 한다. 그리고 커널의 페이징 시스템은 디스크로부터 복사가 종료된 페이지부터 사용자 메모리로 copy_to_user()나 put_user()함수로 데이터를 복사한다. 이것은 ufs(unix file system)의 raw read와 유사한 방식이다. 수백 메가바이트 이상의 멀티미디어 파일의 경우 불필요한 커널의 데이터 복사를 제거할 수 있기 때문에 사용자 프로그램에게 보다 신속한 응답시간을 제공할 수 있다.

zero-copy[20] 는 (그림 3)에서 보듯이 사용자 버퍼로는 데이터가 복사되지 않고 커널내의 Mbuf에 전송되어야 할 데이터 블록이 연결 리스트로 유지된다. 이 리스트 구조는 기존 커널이 가지는 버퍼 헤드 리스트와는 별개로 유지된다. 따라서 사용자가 stream_read()로 커널에 읽기 연산을 수행하면 이 Mbuf에 데이터가 디스크로부터 읽혀지고, stream_send() 연산을 수행하면 커널의 프로토콜 스택은 Mbuf에 있는 데이터 블록을 참조하여 네트워크로 데이터를 전송하게 된다. 따라서 zero-copy도 역시 데이터 복사 회수를 감소시킬 수 있다.

Direct-I/O는 사용자 버퍼로 데이터를 복사하므로 여러 가지 사용자 수준의 버퍼링을 구현할 수 있으므로 응용 프로그램 개발자에게 다양성을 제



(그림 2) Direct I/O



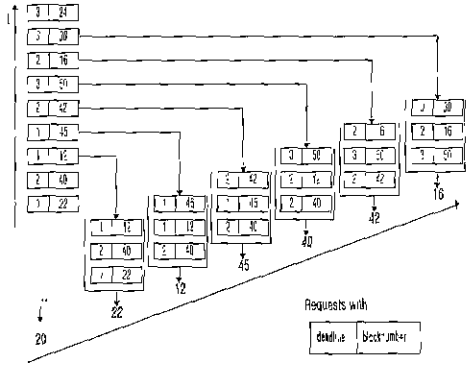
(그림 3) Zero-Copy

공한다. 그러나 zero-copy는 커널에 데이터 블록을 유지하므로 프로세스의 문맥교환(context switching)에 의한 시간 지연이 없으므로 속도는 빠르지만 응용 프로그래머에게 다양성을 제공하지 못한다.

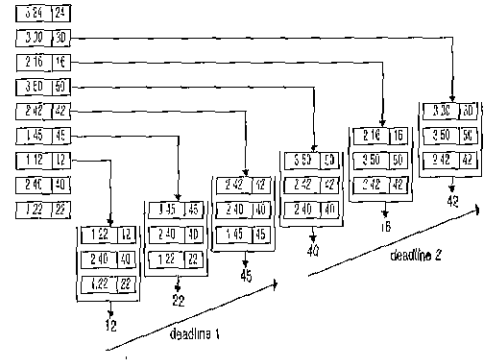
3.5 실시간 디스크 스케줄링

전통적인 디스크 스케줄링 알고리즘의 목적은 처리율을 최대로 하고 각 프로세스마다 공정한 디스크 접근의 보장이다[2]. 그러나, 멀티미디어의 경우 다수의 스트림이 동시에 서비스될 것을 요구한다. 즉 멀티미디어 스트림이 지니는 시간제한적(time-critical)인 테스크의 마감시간이 만족되어야 한다. 일반적으로 리눅스의 장치 독립적(device-independent)인 디바이스 드라이브들은 unidirectional elevator 알고리즘을 채택하고 있고, 개개의 디바이스는 FIFO로 스케줄링하고 있다. 멀티미디어 스트림의 마감시간을 만족하기 위해서는 다음과 같은 실시간 디스크 스케줄링 알고리즘의 적용이 필요하다.

- EDF(earliest deadline first) - EDF는 CPU 스케줄링에서도 사용되는 방식인데 파일시스템에서 적용된다. 즉 마감시간이 가장 최근인 스트림의 블록이 먼저 서비스되는 방식이다. 신속한 프로세스의 응답시간은 보장할지라도 (그림 4)에서 보듯이 디스크 헤드의 움직임이 최적화 되지 못하므로 시스템 전체의 처리율은 저하되고, 과도한 탐색시간을 요구한다.
- SCAN-EDF - EDF의 과도한 탐색시간을 감소하기 위해서 가장 빠른 마감시간을 가진 태스크를 먼저 서비스하되 같은 마감시간을 가진 태스크는 SCAN을 적용하여 처리하는 방식이다. (그림 5)에서 처럼 헤드의 움직임이 12->22->45 처럼 한 방향으로 최적화되어 탐색시간을 EDF보다는 감소시킬 수 있다.



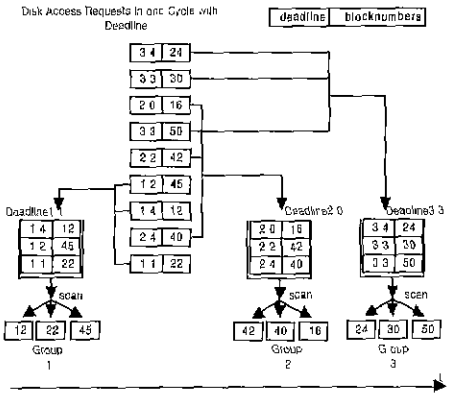
(그림 4) EDF



(그림 5) SCAN-EDF

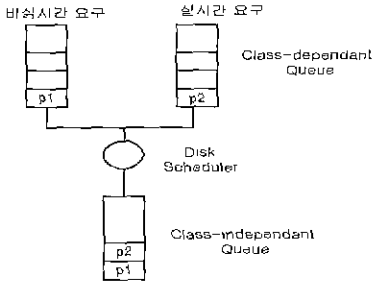
- GSS(group sweeping scheduling) - (그림 6)에서 보듯이 모든 요구를 비슷한 마감시간을 가진 그룹으로 나눈다. 그룹 내에서는 SCAN을 적용하고 그룹간에는 round-robin을 적용하는 방식이다. SCAN에서의 스트림은 한 주기의 첫 번째 슬롯에서 서비스를 받고 다음 주기의 마지막 슬롯에서 서비스를 받는 최악의 경우가 발생할 수 있다. 따라서, 미디어의 연속성을 보장하기 위해서 두 주기동안의 버퍼량이 요구된다. 그러나, GSS는 한 그룹내의 마지막 슬롯에서 반드시 서비스를 받게 되므로 버퍼 요구량이 SCAN보다는 적고, 또 그룹간의 사용된 버

퍼는 재사용이 가능하다. 한편, GSS는 헤드 스케줄링의 복잡성이 단점이라 할 수 있다.



(그림 6) GSS

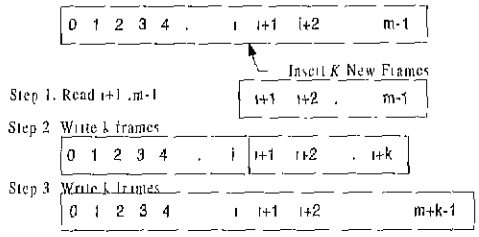
- Mixed strategy - 위의 세 알고리즘은 마감시간을 고려하여 설계된 알고리즘이다. 그러나, 비실시간 데이터와 실시간 데이터가 시스템에 공존할 경우, (그림 7)과 같이 비실시간 데이터의 기아현상(starvation)이 발생하지 않도록 하는 고려가 요구된다.



(그림 7) Mixed strategy

3.6 편집

멀티미디어 응용은 VOD, NOD와 같은 단순 재생을 위한 것뿐만 아니라 동영상 파일을 위한 편집 툴도 다수 사용된다. 멀티미디어 데이터의 편집은 일반 텍스트 데이터의 편집과는 상이한 특징이 있다. 예를 들어 영상의 중간에 새로운 프레임을 삽입하거나 기존 프레임들을 삭제하는 경우가 빈번하다. 따라서, MFS는 편집을 위한 여러 가지 인터페이스를 제공해야 한다. 먼저 ext2, ufs(unix file system)와 같은 기존 파일시스템을 기반으로 하는 임의의 편집용 프로그램의 삽입연산의 예를 보게 되면 다음과 같다. m개의 프레임으로 구성된 비디오 클립에서 프레임 i 와 프레임 i+1 사이에 새로운 k개의 프레임을 삽입하는 경우이다.

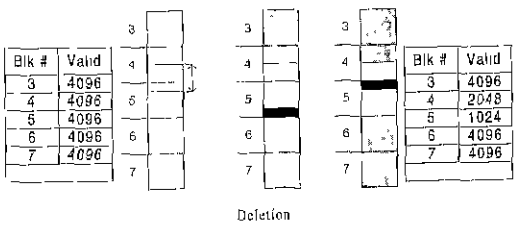


(그림 8) 기존 파일시스템상에서 삽입연산의 예

(그림 8)에서 보듯이 중간에 k개의 프레임을 삽입하기 위해서 단계 1에서 프레임 i+1에서 m-1까지의 프레임을 미리 읽는다. 단계 2에서 k개의 프레임을 추가한다. 마지막으로 단계 3에서는 단계 1에서 읽은 프레임들을 추가한다. 이 방식은 수십에서 수백 메가바이트의 동영상 파일의 경우는 최악의 경우 m-1개의 프레임을 다 읽은 후 다시 write해야 하는 경우가 발생한다. 이것은 편집용 클라이언트의 응답시간을 지연시키는 결정적인 요인이 된다. 따라서 새로운 파일시스템을 위한 프리미티브가 필요하다. 따라서, 본 논문에서는 멀티미디어 데이터를 효율적인 삽입과 삭제를 제공하는 기법을 소개한다.

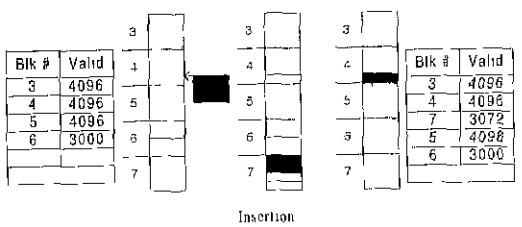
소개하는 파일시스템의 블록은 블록번호(Blk #)와 유효바이트수(valid)로 구성된다고 가정한다. (그림 9)는 4, 5번 블록사이에서 삭제가 발생할 경우 처리하는 방법을 보여주고 있다. 먼저, 4번

블록의 유효바이트 수를 갱신하고, 5번 블록의 남은 바이트들은 5번 블록의 시작위치로 이동시킨다. 남은 5번 블록의 데이터를 4번으로 이동시키지 않는 것은 복사로 인한 오버헤드를 줄이기 위해서이다.



(그림 9) MFS의 삭제 연산의 예

삽입 연산은 더욱 복잡한 연산을 필요로 한다. (그림 10)은 삽입 연산의 예를 보이고 있다. 4번 블록에서 밀려나야 할 부분을 먼저 읽어서 버퍼링해 둔다. 다음, 파일의 끝에 삽입 블록에서 4번 블록에 채운 나머지 부분을 저장한다. 그리고 7번 블록 끝에 4번에서 버퍼링해둔 부분을 추가한다. 마지막으로 삽입 블록으로 4번 블록에서 밀려난 부분을 채운다. 그리고 Index block의 순서를 3, 4, 7, 5, 6으로 변경한다. 새롭게 할당된 7번 블록은 3072 바이트로 부분적으로 채워진다.



(그림 10) MFS의 삽입연산의 예

위의 예에서 계속되는 삽입과 삭제 연산으로 인한 내부 단편화 현상이 발생하지만 기존 파일 시스템과 같은 불필요한 복사는 감소된다. 그리고, 내부 단편화는 비정기적인 압축(compaction)으로 해결할 수 있다. 그러나, MFS의 편집 연산 지원은 해결해야 할 많은 문제가 남아 있다.

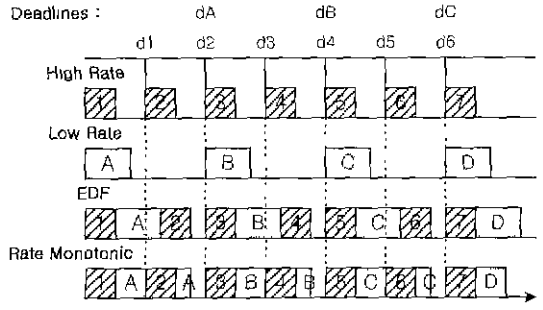
3.7 실시간 프로세스 스케줄링

전통적인 운영체제는 처리율 및 자원 이용율의 최적화와 공정한 스케줄링을 목적으로 하였다. 그러나 실시간 프로세스 스케줄링의 목적은 각 프로세스들의 마감시간을 최대한 만족시키는데 있다. 시간적 제약성을 가지는 멀티미디어 파일과 같은 실시간 데이터를 스케줄링하기 위한 대표적인 알고리즘으로는 EDF, LLF, RMS 등이 있다.

- EDF(Earliest Deadline First) - 가장 먼저 완료되어야 할 태스크, 즉 빠른 마감시간을 가지는 태스크에 가장 높은 우선순위를 부여하는 스케줄링 방식으로 최적 알고리즘이나 과부하 상황에 대처할 수 없다.
- *LLF*(Least Laxity First) - 가용 실행 시간(time available to execute)과 실제 실행 시간(actual execution time) 사이의 차이 값이 슬랙 시간(slack time, laxity)의 정도에 따라 스케줄링하는 기법으로 가장 짧은 슬랙 시간을 가지는 태스크에 가장 높은 우선순위를 할당한다. EDF와 마찬가지로 최적 알고리즘이나 과부하 상황에 대처할 수 없다.
- RMS(Rate Monotonic Scheduling) - 주기적 실시간 태스크를 스케줄링하기 위한 알고리즘으로 가장 짧은 주기를 가지는 태스크에 가장 높은 우선순위를 할당하는 방식이다. 이 알고리즘을 적용시키기 위해서는 다음과 같은 5가지의 가정이 필요하다.
  ① 모든 태스크의 요구 마감시간은 주기적이어야 한다.
  ② 각 태스크는 다음 번의 태스크 실행 전까지 완료되어야 한다.
  ③ 모든 태스크는 독립적이어야 한다.

④ 각 태스크의 실행시간은 일정해야 한다.
⑤ 시스템 내의 비주기적 태스크는 마감시간이 없어야 한다.

일반적으로 EDF는 스케줄링의 재배치가 너무 자주 발생하여 상당한 CPU 오버헤드를 초래할 수 있다. 또 한 시스템 내에서 한 개 이상의 스트림이 병렬적으로 동작할 경우 RMS가 EDF보다 context switching이 잦다. (그림 11)은 두 스트림이 동시에 서비스될 때 RMS가 EDF보다 context switching이 잦음을 보여주는 예이다.



(그림 11) EDF와 RMS의 context switching의 빈도 비교

EDF의 확장으로서 TDS(time-driven scheduling) 알고리즘은 과부하 상황에 대처할 수 있도록 고안한 것인데 더 이상 마감시간을 만족할 수 없는 태스크를 중단시킨다. 더 이상 마감시간을 만족할 수 없는 task를 중단시켜도 과부하 상태가 지속될 경우에는 우선순위가 낮은 태스크를 중단한다. 또 다른 확장으로서 우선순위 기반 EDF 방식이 있다. 이는 각 태스크를 주요부분(mandatory part)과 부부분(optional part)으로 나누어서 주요부분의 마감시간에 따라서 스케줄링한다. 따라서 부분은 과부하 상태의 경우에는 실행되지 않을 수도 있다.

4. 결 론

본 논문에서는 리눅스상에서 멀티미디어 데이터를 효율적으로 지원하기 위한 기술들을 설명하였다. 리눅스에서 MFS는 2GB이상의 대용량 파일의 지원을 위해서 64비트 주소체계와 이를 위한 가상 메모리의 주소 체계의 수정이 필요하다. 디스크의 특성을 고려한 파일의 배치와 스트림을 위한 연속저장 방식을 MFS는 지원해야 한다. 또한, 버퍼캐시를 사용할 경우는 기존 LRU를 대체할 수 있는 연속 미디어에 적합한 캐시 시스템이 요구되며, 버퍼 캐시를 사용하지 않는 경우는 direct I/O와 Zero-copy를 통하여 불필요한 메모리 복사를 제거해야 한다. 또한, 스트림의 마감시간 보장을 위해 실시간 디스크 스케줄링 및 프로세스 스케줄링이 요구되며, 대용량 미디어 데이터의 삽입과 삭제를 위한 새로운 프리미티브의 구현이 필요하다.

참고문헌

[1] Prabhat k. andleigh, Kiran thakrar, "Mulltimedia Systems Design", pp.112, Prentice Hall PTR, 1996.
[2] Ralf Steinmetz, Klara nahrstedt, "Multimedia: Computing, Communications & Applications", Prentice Hall PTR, pp.276-277 1995.
[3] S.J.Mullender, "Systems of the Nineties Distributed Multimedia Systems", In Systems of the 90s and Beyond, Intl. Workshop, pp.273-278, 1991.
[4] M Beck, H Bohme, M Dziadzka, U Junitz, R Magnus, D. Verworner, "Linux Kernel Internals, second edition", Addison-Wesley, pp92, 1998.
[5] Shahram Ghandeharizadeh, Seon Ho Kim, Gyrus Ghahabi and Roger Zimmermann, "Placement of continuous media in Multi-Zone Disks", KAP, pp.23-59, 1997.
[6] Anderson, D. and Homsy, G., "A Continuous

Media I/O Server and its Synchronization," *Computer*, October 1991.

[7] Berson, S., Ghanderharizadeh, S.,Muntz,R. and Ju,X., "Staggered Striping in Multimedia Information Systems," Proceeding of the ACM SIGMOD, pp.79-89, 1994.

[8] Berson, S., Golubchik, L. and Muntz, R.R., "A Fault Tolerant Design of a Multimedia Server," Proceedings of the ACM SIGMOD, pp.364-375, 1995.

[9] Yitzhak Birk, "Track-Pairing: a Novel Data Layout for VOD Servers with Multi-Zone-Recording Disks", Proceedings of the international conference on multimedia computing and systems, pp.248-255. 1995.

[10] J.W.Kim, Y.L.Lho, and K.D.Chung, "An Effective Video Block Placement Scheme on VOD Server based on Multi-Zone Recording Disks," *IEEE Multimedia Systems*, pp.29-36, 1997.

[11] P.V.Rangan, T.Kaeppner, and H.W.Win. "Technoques for efficient Storage of Digital Video and Audio", Proceedings of 1992 Workshop on *Multimedia Information Systems*,. 1992.

[12] Maurice J. Bach, "The Design of the UNIX Operating System", pp.41-94, 1986.

[13] D.P. Anderson, Y. Osawa, and R.Govindan, "A File System for Continuous Media," ACM Transaction on Computer System, Vol.10, No.4, 1992.

[14] P.V. Rangan. and H.M.Vin, "Designing File Systems for Digital Video and Audio," Proceedings of the 13th ACM Symposium on Operating System Principles, 1991.

[15] P.Lougher, and D.Shepherd, "The Design of a Storage Server for Continuous Media," The Computer Journal, 1993.

[16] W. Kun-Lung, and S.Y. Philip, "Consumption-Based Buffer Management for Maximizing System Throughput of a *Multimedia System*," IEEE ICMCS, 1996.

[17] B.Ozden, R.Rastogi, and A.Silberschatz, "Buffer Replacement Algorithm for Multimedia Storage Systems," *IEEE ICMCS*, 1999.

[18] I.H.Kim, J.W.Kim, S.W.Lee, and K.D.Chung, "VBR Video Data Scheduling Using Window_ Based Prefetching," IEEE ICMCS, 1999.

[19] S.R.Yeon and K.Koh, "A Dynamic Buffer Management Technique for Minimizing the Necessary Buffer Space in a Continuos Media Server," IEEE ICMCS, 1996.

[20] M.M.Buddihikot, X,J.Chen, D.Wu, and G.M. Parulkar, "Enhancements to 4.4 BSD Unix for Efficient Networked Multimedia in Project MARS," IEEE ICMCS, pp.326-337, 1998.

김 정 원

1995년 부산대학교 전자계산학과 졸업 (이학사)
1997년 부산대학교 전자계산학과 대학원 졸업 (이학석사)
1999년 부산대학교 전자계산학과 박사 수료

관심분야 : 멀티미디어, VOD, 멀티미디어 네트워크

이 승 원

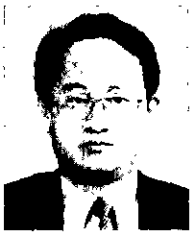
1997년 부산대학교 전자계산학과 졸업 (이학사)
1999년 부산대학교 전자계산학과 대학원 졸업 (이학석사)
1999년-현재 부산대학교 전자계산학과 박사과정

관심분야 : 멀티미디어, VOD, 실시간 운영체제

김 인 환

1997년 부산대학교 전자계산학과 졸업 (이학사)
1999년 부산대학교 전자계산학과 대학원 졸업 (이학석사)
1999년-현재 부산대학교 전자계산학과 박사과정

관심분야 : 멀티미디어, VOD, 실시간 운영체제



정 기 동

1973년 서울대학교 공과대학 공학사
1975년 서울대학교 대학원 석사
1986년 서울대학교 대학원 계산통계학과 박사
1990년-1991년 MIT, South Carolina 대학 교환교수

1978-현재 부산대학교 정보·컴퓨터공학부 교수
관심분야 : 멀티미디어, 병렬처리, 운영체제