멀티미디어 End-to-End 서버용 리눅스 기반 I/O 시스템 설계 및 구현

남 상 준[†]·이 병 래[†]·박 남 섭[†]·이 윤 정[†]·김 태 윤[††]

요 약

최근 인터넷의 확산으로 인하여 멀티미디어 서비스에 대한 사용자의 요구가 증가되고 있다. 그러나 서버 시스템들은 이러한 사용자들에게 멀티미디어 데이터를 효과적으로 공급하지 못하고 있다. 멀티미디어 응용 프로그램들은 같은 데이터를 여러 장치들 사이에서 경로를 유지하기 때문에 부하가 발생하고, 효과적인 I/O(Input/Output) 서브시스템이 요구되어지고 있다. 멀티미디어 데이터를 효과적으로 공급하지 못하는 비효율적인 I/O의 원인으로 잦은 데이터 복사와 문맥교환이라고 인식되어 왔다. 따라서 본 논문에서는 리눅스 시스템에 새로운 *Multimedia Stream System Call*(MSSC) 메커니즘을 제안한다: RTP(Real-time Transport Protocol) 프로토콜과 함께 이 메커니즘은 커널 영역에서 수행하도록 하는 것이다. 이 MSSC 메커니즘을 사용한 결과 일반적인 리눅스 시스템보다 12.5%~14%의 성능이 향상됨을 보여준다.

The Design and Implementation of the Multimedia End-to-End Server I/O System based on Linux

Sang-Jun Nam[†] · Byung-Rae Lee[†] · Nam-Sup Park[†]
Yoon-Jung Rhee[†] · Tai-Yun Kim[††]

ABSTRACT

In recent years, users' demands for multimedia service are increasing because of a diffusion of internet. Server systems, however, offer inefficient multimedia data service to users. Multimedia applications often transfer the same data between shared devices at very high rates, and therefore require an efficient I/O subsystem. Data copying and context switching have long been identified as sources of I/O inefficiency. Therefore we propose the new *Multimedia Stream System Call*(MSSC) mechanism, which is inserted into a Linux kernel: The MSSC mechanism operates in kernel domain with RTP (Real-time Transport Protocol). We present measurements indicating that use of our techniques resulted in a 12.5%~14% gain in throughput as compared with a conventional Linux system.

키워드 : 멀티미디어 서버, 리눅스, MSSC, RTP

1. 서 론

최근 멀티미디어 서비스에 대한 사용자의 요구에 따른 대단위 데이터 처리로 인해 서버 시스템의 부하가 증가하고 있다[1]. 이러한 요구를 충족시키는 서버 시스템은 지금까지의 한정적이고 일반적인 범위에서 벗어나, 실시간 처리 능력과 대용량의 데이터를 효율적으로 전송하는 새로운 서버 시스템이 요구되고 있다. 회사에서는 네트워크를 통해 서로 다른 지사들과 화상 회의를 하고, 자신이 보고자 하는 TV를 시청하고, 상호 작용하는 게임을 하고 자신이 찾는 동영상이 실린 뉴스를 찾아보고 다양한 비디오와 오디오를 볼 수 있다. 이러한 모든 서비스들은 서버 시스템에게 빠른 피드백을 요구한다[2].

이러한 서버 시스템은 저장 장치와 네트워크 하부 시스템 사이에서 주기적으로 데이터를 주고받는다. 따라서 기대되는 작업에 대해 서버 시스템은 높은 성능으로 디자인되고 전송되어야 한다. 이 방안으로 멀티미디어 데이터 전송을 위한 RTP(Real-time Transport Protocol)[3] 프로토콜이 사용되지만, 저장 장치와 네트워크 하부 시스템 사이의 효과적인 경로는 제어하지 못한다[4]. 대다수의 멀티미디어 응용 프로그램들은 사용자 영역과 커널 영역에서 과도한 데이터 복사와 문맥 교환이 발생한다[5]. 멀티미디어 데이터 전송과 같이 한번 접속된 서비스가 다른 변경 없이 연속적으로 전송된다면, 데이터 복사의 횟수를 줄이고 빠른 데이터 경로와 전송을 이룰 수 있다.

본 논문에서는 멀티미디어 데이터 전송을 위한 RTP 프로토콜을 MSSC(Multimedia Stream System Call) 메커니즘과

† 준 회 원 : 고려대학교 대학원 컴퓨터학과
†† 종신회원 : 고려대학교 컴퓨터학과 교수
논문접수 : 2001년 8월 16일, 심사완료 : 2001년 11월 7일

함께 커널에 내장함으로써 비디오와 오디오 같은 실시간 서비스에 만족하는 효율적인 전송 메커니즘을 제안한다.

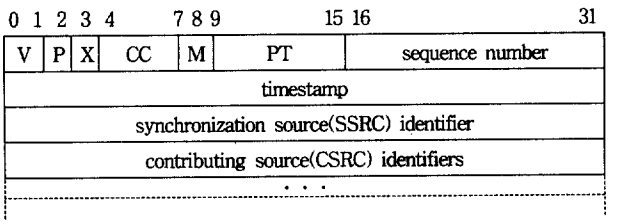
빠른 전송을 수행하기 위해 UDP 전송 메커니즘을 커널 내부에서 수행하도록 MSSC 메커니즘을 설계하고 RTP 프로토콜과 함께 커널 영역에 내장하여 이를 기반으로 리눅스 환경에서 구현과 성능 평가를 한다. 이러한 MSSC 메커니즘은 멀티미디어 데이터 전송의 서비스 품질을 향상시키고 서버 시스템의 I/O(Input/Output) 성능을 향상시킨다.

본 논문의 구성은 다음과 같다. 2장은 멀티미디어 데이터 전송을 위한 RTP 프로토콜에 대해 살펴보고, 일반적인 서버 전송 메커니즘과 MARS(Massively-parallel And Real-time Storage) 메커니즘[13]의 문제점을 분석한다. 3장은 리눅스 서버의 전송 메커니즘을 기반으로 MSSC 메커니즘을 제안한다. 4장은 멀티미디어 환경의 일반적 리눅스 서버와 MSSC 메커니즘을 내장한 리눅스 서버와의 비교 및 성능 평가를 한다. 마지막 5장은 본 논문의 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 RTP를 사용한 멀티미디어 데이터 전송 스택

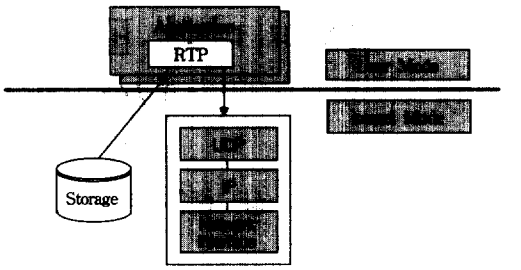
최근에 인터넷상에서 멀티미디어 서비스들의 전송에 대한 과부하와 실시간 전송을 위한 연구에 관심이 집중되고 있다. 인터넷폰의 성장, 리얼 오디오상의 오디오와 비디오 스트림, Mbone이 실례이다[6]. RTP는 멀티캐스트 또는 유니캐스트 상에서 음성, 화상, 또는 모의 데이터와 같은 실시간 데이터를 전송하는 응용에 적합한 단대단 트랜스포트 기능을 제공한다. (그림 1)은 RTP 패킷 형식을 나타낸 것이다.



(그림 1) 실시간 데이터 전송을 위한 RTP 패킷 헤더

헤더는 고정 크기를 가지며 멀티미디어 정보에 따라서 헤더 뒤에 특정 정보 및 데이터를 첨가한다. 즉 RTP 패킷들은 망을 통해 전달되면서 종단 시스템에서는 여러 노드들로부터 온 RTP 패킷들을 받고 이들을 적절히 조합시켜서 데이터를 받아본다. RTP 프로토콜은 전송 계층과 네트워크 계층의 상위에서 작동하도록 디자인 된 프로토콜이다[3]. RTP 프로토콜은 RTP 계층에서 만들어지고 번역되지만, 다중화와 체크섬은 그 하부 계층인 UDP를 이용한다. (그림 2)는 현재 멀티미디어 데이터 전송에 널리 사용되는 RTP 프로토콜의 스택이다.

(그림 2)에서 사용자 영역에서 멀티미디어 데이터와 함께



(그림 2) RTP 기반 멀티미디어 데이터 전송 스택

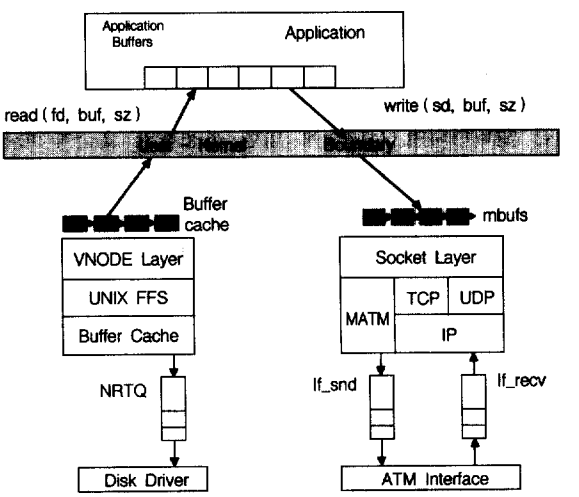
만들어진 RTP 헤더는 그 하위 계층인 UDP 프로토콜로 전달되고 네트워크 인터페이스로 전달되는 모습을 볼 수 있다. 바로 이러한 점에서 RTP 패킷의 생성과 하부 계층으로의 전송을 무수히 반복하는 멀티미디어 데이터의 경우는 과도한 데이터 복사와 문맥 교환이 발생한다.

현재까지 RTP 프로토콜은 멀티미디어 데이터의 실시간 전송을 위해 만들어 졌다. 응용 프로그램상에서 전송 속도를 향상시키는 연구와 QoS에 대한 연구[6, 7]는 계속 진행중에 있으나 RTP 프로토콜의 전송 스택의 변화에 대한 연구는 활발하지 않다.

현재 RTP는 사용자 응용 프로그램에서 작업이 이루어진다. RTP도 하나의 프로토콜이므로 운영체제의 커널 영역에 두면 RTP 패킷을 만들 때 빠르게 작업이 이루어 질 수 있다. 이러한 스택의 변화를 본 논문에서 제안한다.

2.2 일반적인 서버 전송 메커니즘과 MARS 메커니즘

(그림 3)은 현재 Linux 운영체제 시스템에서의 네트워크 I/O 계층구조를 보여 준다. 응용 프로그램에서 데이터를 요구하고 그 데이터가 네트워크 인터페이스로 전송되는 일반적인 과정을 나타낸다. (그림 3)의 서버 시스템은 사용자 공간에서 네트워크 장치까지 두 단계의 데이터 복사가 사용자 영역과 커널 영역 사이에서 발생한다.



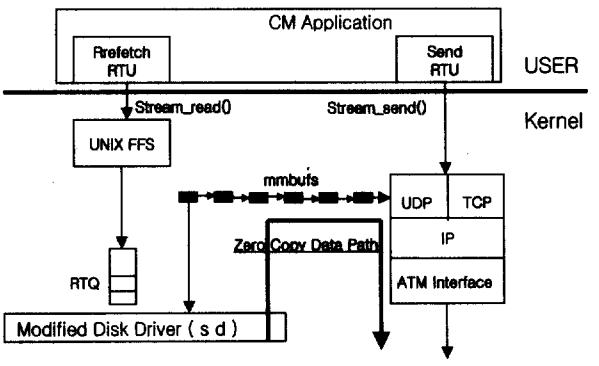
(그림 3) Linux 서버 시스템의 데이터 전송 메커니즘

첫 번째 복사는 read 콜이다. 이것은 커널의 버퍼 캐쉬에

서 사용자 버퍼 공간까지 데이터를 옮기는 경우이다. 두 번째 복사는 write 콜이다. 이 접근은 일반적인 작은 크기의 입출력을 행하는 전통적인 텍스트와 바이너리 파일은 전송이 잘 이루어진다. 그러나 멀티미디어 데이터(오디오, 비디오, 애니메이션 등)와 같은 캐싱 속성을 가진 것들은 기대하는 것만큼의 효과를 보지 못한다. 또한 이 서버 시스템은 메모리 공간을 많이 사용한다. 데이터는 종종 이것이 재 사용될 수 있음에도 불구하고 대체될 수 있다. 성능상의 불이익이 잉여 복사로 인해 이루어진다[8, 9]. 데이터 복사를 최소화하는 연구는 이전부터 시작되었다. 보호되는 영역을 만들어 물리적 데이터 전송을 최소화하는 fbufs[10], 수신측의 커널 영역에 식별자를 두어 네트워크 장치와 애플리케이션의 버퍼 사이의 경로를 줄이는 Splice[11], 성능 향상을 위해 프로그래밍 언어 수준의 제어와 데이터 구조를 명확히 하는 SPIN[12], MARS[13] 메커니즘 등이 있다. 이러한 연구들 중 일반적인 서버 전송 메커니즘의 단점인 잦은 데이터 복사의 보완이 MARS(Massively-parallel And Real-time Storage) 메커니즘이다[13].

(그림 4)는 디스크에서 네트워크 I/O로의 새로운 데이터 경로와 제어를 제시한 MARS 메커니즘이고 특징을 살펴보면 다음과 같다.

디스크와 네트워크 I/O 사이의 데이터 전달은 Zero Copy를 위해 mmbufs(Multimedia Memory Buffers)라는 새로운 버퍼 관리 메커니즘을 디자인하고 구현하였다.



(그림 4) MARS 메커니즘

MARS 메커니즘은 정교한 버퍼 캐쉬인 mmbufs를 커널에서 관리함으로써 사용자 공간과 저장장치 사이의 전송 수단으로 사용된다. (그림 4)와 같이 mmbufs를 통해 저장장치와 네트워크 사이의 Zero-copy와 같은 빠른 데이터 경로가 공급될 수 있다.

그러나, MARS 메커니즘 또한 다음과 같은 문제점이 발생하게 된다.

- mmbuf는 2개의 포인터 리스트로 되어 있다. 이러한 2개의 포인터를 찾아 다니면서 디스크에서 네트워크로 데이터를 전송하게 된다. 포인터 연산으로 인한 불필요한 오버헤드를 발생시키게 된다.
- 새로운 mmbuf라는 커널내 구조체 생성으로 전송량이 많은 경우 과도한 메모리 할당이 이루어진다. 만약 제한된 메모리보다 많은 mmbuf가 생성 될 경우 시스템에게는 치명적인 다운 현상이 발생한다.
- 디스크 장치와 네트워크 장치 사이에 새로운 구조체 생성으로 인한 다른 장치와의 호환성 결여 문제가 발생한다. 다른 장치와의 호환성을 고려할 경우 추가의 mmbuf의 데이터 구조를 정의해야 하는 단점이 발생한다.

3. 제안된 MSSC 메커니즘 설계 및 구현

2장에서 설명한 바와 같이 RTP 프로토콜은 현재 사용자 응용 프로그램상에서 동작되므로 커널 내부에서 동작하는 것보다 데이터 복사와 문맥 교환에서의 지연 시간이 발생하는 단점이 있다. 그리고 MARS 메커니즘도 커널 내부에서 동작하나 새로운 구조체의 생성과 포인터 연산으로 커널에 부하를 준다.

본 논문에서는 멀티미디어 데이터의 효율적 전송을 위해 RTP 프로토콜을 MSSC(Multimedia Stream System Call) 메커니즘 내부에서 수행하게 하고, 이 MSSC 메커니즘을 커널 내부로 내장하는 방법을 제안한다.

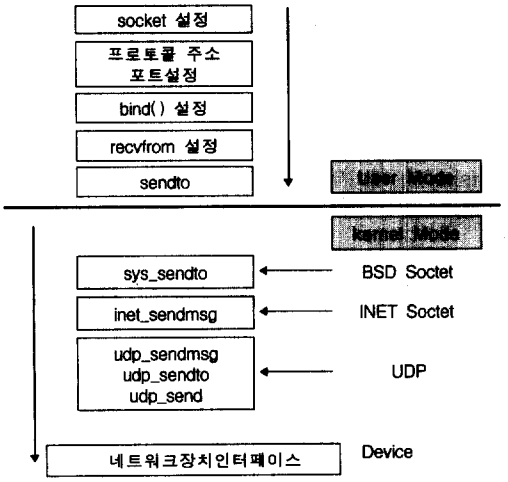
3.1 MSSC 메커니즘의 설계

2.2절의 MARS 메커니즘을 살펴보면 멀티미디어 환경의 서비스에 대해 일반적인 서버 시스템보다 빠른 전송을 하기 위함이다. 기존 운영체제를 멀티미디어 서버로 사용하기 위해서 다음과 같은 문제점의 개선이 필요하다.

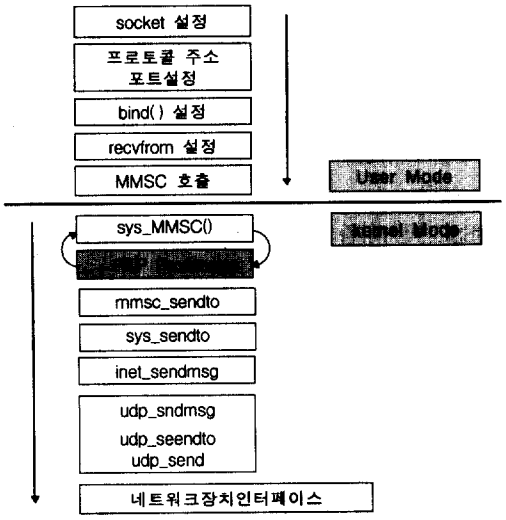
- 사용자 영역과 커널 영역 사이에서의 잦은 데이터 복사가 일어난다.
- 응용 프로그램 수행시 발생하는 과도한 문맥 교환은 커널의 작업보다 많은 오버헤드를 발생시킨다.

멀티미디어 데이터를 전송하는 서버에서 오디오나 비디오 데이터의 경우는 변경이 많이 일어나지 않는 연속적인 스트림의 전송이 요구된다. 이와 같은 전송은 항상 새로운 데이터를 생성해서 전송하는 것이 아니라 저장 장치로부터 읽어온 연속적인 데이터를 전송하는 것이다. 이와 같이 다음에 전송할 데이터가 무엇인지 아는 멀티미디어 데이터인 경우 반복적으로 사용자 영역과 커널 영역을 넘나들며 데이터를 이동하면서 불필요한 데이터 복사와 문맥 교환으로 인한 오버헤드가 발생하게 된다. 잦은 데이터 복사와 문맥 교환을 줄이기 위해 연속적인 데이터를 전송할 때 사용자 영역에서는 한번의 시스템 호출을 하고 연속적인 데이터는 커널 영역에서 수행하는 전송 메커니즘이라면 전송시간을 단축시킬 수 있을 것이다. 리눅스 기반의 일반적인 UDP 전송 메커니즘의 커널 경계 부근에서 데이터 전송의 경로는 (그림 5)의 (a)와 같다. (그림 5)의 (a)는 응용 프로그램

에서 UDP 프로토콜을 거쳐 네트워크 인터페이스로 데이터 전송의 경로를 나타낸 것이다[14-17].



(a) 일반적인 리눅스 전송 메커니즘



(b) MSSC 메커니즘 추가 전송 메커니즘

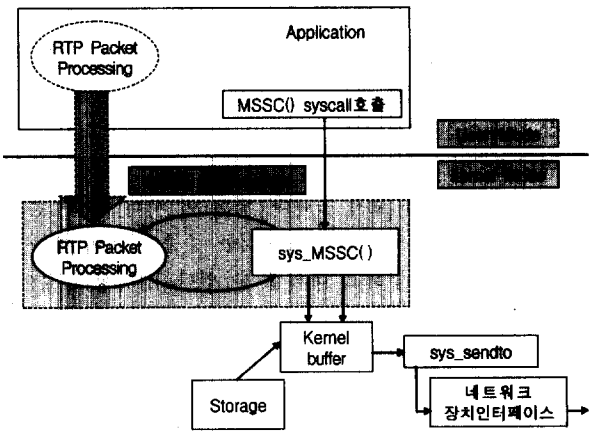
(그림 5) MSSC 메커니즘 호출 전과 후의 커널영역과 사용자 영역 전송 메커니즘

(그림 5)의 (b)와 같이 RTP 헤더를 커널에 삽입해 네트워크 인터페이스로 보내는 MSSC 메커니즘을 본 본문에서 제안한다.

본 논문에서 MSSC 메커니즘은 RTP 프로토콜을 응용 프로그램에서의 성능 향상이 아니라 커널로 흡수하는 것이다. RTP 프로토콜은 UDP 프로토콜 기반으로 전송이 수행되므로 MSSC 메커니즘과도 잘 호환될 수 있다.

본 논문에서 제안하는 MSSC 메커니즘의 구조를 좀 더 자세히 보면 (그림 6)과 같다.

(그림 6)의 응용 프로그램에서 각각의 패킷을 생성하는 RTP 프로토콜을 커널 내부로 옮겼다. 이로 인해 저장 시스템과 사용자 응용 프로그램을 오가며 수행하는 데이터의 과도한 복사를 방지 할 수 있다. 또한 RTP 프로토콜을 커



(그림 6) MSSC 메커니즘 구성도

널에 내장함으로써 비디오, 오디오 기반 응용 프로그램의 단순성을 꾀하였다. MSSC 메커니즘은 리눅스 시스템에 맞추어 설계하였다. 리눅스 시스템은 커널의 변형이 용이하기 때문이다. MSSC 메커니즘은 RTP 패킷의 처리뿐만 아니라 기존의 UDP기반 전송 단계를 MSSC의 호출로 전송 단계를 줄임으로써 성능상의 이득을 볼 수 있다. 이러한 이득을 살펴보면 다음과 같다.

- read/write의 이득
  응용 프로그램에서 MSSC 시스템 콜을 사용하여 데이터 전송을 커널 내부로 옮김으로써 이에 대한 불필요한 데이터 복사와 문맥 교환을 줄였다.
- RTP 패킷 처리
  RTP 패킷을 생성함에 있어, 기존의 방식은 사용자 응용 프로그램에서 이루어지는 것을 커널 내부의 모듈로 내장함으로써 처리 속도의 향상과 RTP 응용 프로그램의 인터페이스를 단순하게 만들 수 있게 하였다.

또한 MSSC 메커니즘은 MARS 메커니즘과 비교해 볼 때 다음과 같은 장점을 가진다.

- MARS의 mmbufs는 두 개의 포인터 리스트를 이용하여 새로운 버퍼의 할당이나, 디스크의 읽기 작업이 시작되면 해당 mmbufs의 포인터를 찾아다니는 오버헤드를, 이 MSSC 메커니즘에서는 기존의 전송 방식을 그대로 따르면서 전통적인 전송 단계의 과정을 줄여줌으로써 전송 속도상의 효과를 보고자 하였다.
- MARS 메커니즘의 경우처럼 전송량이 많아질 경우 과도한 mmbuf의 할당이 아닌 기존의 버퍼 캐쉬를 사용함으로써 적응성을 유지하였다.
- MARS 메커니즘과는 달리 다른 장치들과의 호환성 측면에서도 새로운 구조체를 생성하지 않음으로써 유연성을 가지게 하였다.

이전부터 진행되어 오던 연구의 큰 목표는 사용자 영역과 커널 영역의 과도한 데이터 복사의 회피이다[10, 11, 13].

장치들의 인터페이스와 구조를 변경해서 좋은 메커니즘을 만들 수도 있으나 운영체제 내부에 새로운 구조가 들어간다는 것은 오버헤드를 발생시키는 원인이 되고 비용 측면에서도 비효율적이다.

3.2 MSSC 메커니즘 구현

리눅스 시스템의 커널 소스가 공개 방식이므로 새로운 MSSC 메커니즘을 이용한 전송을 적용시켜 구현하였다.

현재의 리눅스 시스템은 멀티미디어 데이터 전송에 적합한 전송 메커니즘을 가지고 있지 않다. 이러한 리눅스 시스템과 MSSC 메커니즘을 커널에 내장한 리눅스를 비교하여 보았다.

RTP 프로토콜이 UDP 전송 계층을 기반으로 동작하므로 리눅스 시스템에 RTP 프로토콜을 커널 수준에서 내장하였다. 리눅스의 UDP 전송 메커니즘은 3.1절 (그림 5)의 (a)와 같이 동작하기 때문에 데이터 전송시 커널 영역과 사용자 영역 사이에서 데이터 복사와 문맥 교환이 발생한다.

커널 내부에 UDP 기반 RTP 패킷 전송을 담당하는 MSSC 시스템 콜을 등록한다.

등록한 MSSC 함수의 RTP 패킷 전송 메커니즘의 알고리즘은 (그림 7)과 같다.

```
01 : asmlinkage int sys_MSSC(fd, RTP_INFO)
02 : {
03 :     ...
04 :     mssc_read(fd, buf, len) ;     /* Data Read */
05 :     rtp_packet = rtp_hdr_build(RTP_INFO, buf, len) ;
                    /*RTP Packet Generate*/
06 :     server_len = mssc_sendto(rtp_packet) ;
                    /*Packet Send*/
07 :     ...
08 :   return 1 ;
09 : }
```

(그림 7) 커널에 내장한 MSSC 메커니즘

(그림 7)의 알고리즘은 크게 3부분의 함수로 구성되어 있다. 사용자 응용 프로그램에서 넘겨준 RTP 헤더의 정보를 가지고 mssc_read, rtp_hdr_build, mssc_sendto 함수들로 멀티미디어 데이터 전송 메커니즘이 이루어지게 된다.

MSSC 메커니즘의 동작 과정은 다음과 같다.

- 클라이언트의 요청에 따라 서버의 응용 프로그램이 열린다.
- 클라이언트와의 통신을 위한 설정이 세팅된다.
- 서버 응용 프로그램은 클라이언트의 요청에 맞는 멀티미디어 데이터의 정보를 MSSC 시스템 콜을 사용하여 커널에 전달한다.
- mssc_read 함수는 전송하고자 하는 저장 장치의 위치 포인터로 데이터를 읽어 온다.
- rtp_hdr_build 함수는 mssc_read 함수가 읽어온 데이터와 사용자 응용 프로그램에서 넘어온 RTP 헤더 필드로 RTP 패킷을 만든다.
- 만들어진 RTP 패킷은 mssc_sendto 함수를 이용하여 RTP 패킷을 네트워크 인터페이스에게 보내어진다.

위와 같은 단계로 본 논문에서 제안하는 MSSC 메커니즘은 사용자 공간과 커널 공간사이의 불필요한 데이터 복사를 제거할 수 있었다. 사용자 응용 프로그램에서는 단지 커널 내부에 새로 만든 MSSC() 시스템 콜만을 호출하면 된다.

서버 응용 프로그램의 MSSC 메커니즘 호출 알고리즘은 (그림 8)과 같다.

```
01 : /* file name : MSSC_Serv.c */
02 : _syscall2(setting parameter)
03 : int      main(void)
04 : {
05 :     socket 설정 ;
06 :     프로토콜, 주소, 포트 설정 ;
07 :     ...
08 :     bind 설정 ;
09 :     start time ;
10 :     n = MSSC(fd, RTP_INFO) ;
11 :     end time ;
12 : }
```

(그림 8) 서버 애플리케이션에서 MSSC 메커니즘 호출

(그림 8)의 알고리즘은 커널에 내장한 MSSC 메커니즘을 호출한다. 그리고 MSSC 함수에서 전송하고자 하는 파일의 포인터와 RTP 패킷의 정보를 보낸다. RTP_INFO의 내용은 RTP 패킷에서 주요한 정보를 나타내는 헤더의 필드를 의미하는데 버전 정보와 순서 번호와 같이 일괄적인 정보는 커널의 MSSC 함수에서 설정하고 데이터 종류와 같은 필드는 응용 프로그램에서 설정해서 커널 내부로 정보를 넘겨준다.

4. MSSC 메커니즘의 성능 평가 및 분석

위의 3장에서 제안한 MSSC 메커니즘의 성능 평가를 리눅스 시스템에서 수행하고 일반적인 리눅스 시스템과 MSSC 메커니즘 내장 리눅스 시스템의 전송 시간을 microsecond 단위까지 비교하였다. 이러한 실험을 통해 이 논문의 제안 모델인 MSSC 메커니즘 성능이 향상됨을 보여준다. 실험 환경의 구성을 위해 2개의 리눅스 시스템을 사용한다. 한쪽은 서버 시스템으로 RTP 패킷을 전송하고 다른 한쪽은 이 패킷을 수신함으로써 성능을 비교해 보았다. 실험에 필요한 시스템적 하드웨어 사양은 <표 1>과 같다.

〈표 1〉 시스템 및 운영체제의 소프트웨어 사양

	리눅스 시스템 I (Server)	리눅스 시스템 II (Client)
CPU	인텔 펜티엄 III 600MHz	인텔 펜티엄 III 600MHz
RAM	128M	128M
HDD	20G	30G
리눅스 배포판	Alzza Linux 6.0	Power Linux 6.0
Kernel Version	2.2.12-20	2.2.5-15
GCC Version	2.91.66	2.91.66
Libc6 Version	2.1.1	2.1.1

현재 오디오와 비디오 데이터 전송은 송신측에서 패킷마다 간격을 두어 전송하므로 패킷 손실을 최소화 하였으며, RTP 패킷 전송 간격은 오디오 스트림의 간격인 30ms의 간격을 두어 전송하였다. 또한, 서버 시스템이 서비스하는 클라이언트가 다수임을 가정하여 10개의 클라이언트가 서버 시스템에 동시에 접속하는 것으로 하였다. MSSC 메커니즘의 성능 실험은 다음 <표 2>와 같이 2가지 경우로 실험을 수행하였다.

〈표 2〉 실험 1과 실험 2의 환경 비교

	패킷 크기	패킷당 전송 간격	세션 수	세션당 패킷 개수	과부하
실험 1	32 bytes	30ms	10	1000	×
실험 2	32 bytes	30ms	10	1000	○

4.1절의 실험 1은 패킷 하나당 크기를 32 바이트 크기로 고정하고, 1000개의 패킷을 30ms의 속도로 전송하고, 10개의 세션이 서버 시스템에 연결되어 전송하였다. 4.2절의 실험 2는 현재 서버 시스템의 경우 데몬이나 프로세스와 같은 서비스로 인해 서버 시스템에 과부하가 발생한다. 이러한 환경에 맞추어 과부하시 패킷 전송을 수행하고, 다른 실험 조건은 실험 1과 동일하다.

4.1 MSSC 메커니즘 성능 평가-실험 1

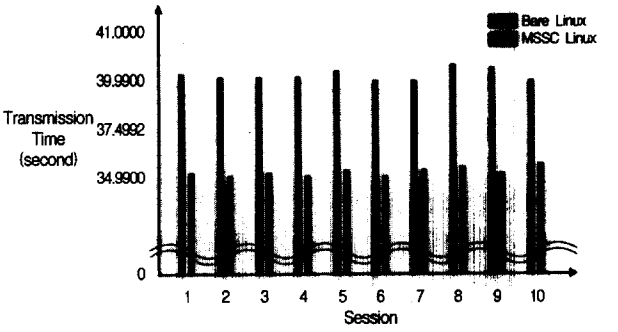
위의 <표 2>와 같은 실험 조건을 구성하여 실험 1을 하였다. 아래 <표 3>은 일반적인 리눅스와 MSSC 메커니즘을 내장한 리눅스의 세션별 전송 속도를 비교한 표이다.

〈표 3〉 일반적 리눅스와 MSSC 메커니즘 리눅스의 세션별 전송 속도 비교

Session		1	2	3	4	5
소요 시간 (초)	일반리눅스	39.995137	39.993740	39.993734	39.993441	39.994387
	MSSC 리눅스	34.996336	34.993574	34.994935	34.994168	34.997810
Session		6	7	8	9	10
소요 시간 (초)	일반리눅스	39.993312	39.993634	39.999628	39.997442	39.990962
	MSSC 리눅스	34.992506	34.998779	34.998490	34.994065	34.998664

각각의 데이터 값은 100회씩 반복하여 평균을 구한 값이고 microsecond 단위까지의 시간을 구했다. <표 3>에서 보는 바와 같이 MSSC 메커니즘을 내장한 리눅스가 일반적인 리눅스보다 각 세션별 성능이 향상됨을 볼 수 있다. 10개 세션 각각이 거의 비슷한 시간으로 1000개의 패킷을 전송하는 것을 볼 수 있다. 이것을 좀 더 구체적으로 그래프로 나타내면 (그림 9)과 같이 나타낼 수 있다.

(그림 9)에서 보듯이 MSSC 메커니즘을 내장한 리눅스가 일반적인 리눅스보다 30ms의 간격으로 패킷 전송을 행할 때 약 12.5%의 성능 향상이 있음을 볼 수 있다. 12.5%라는



(그림 9) 일반적 리눅스와 MSSC 메커니즘 리눅스의 세션별 전송 속도 비교

성능 향상이 낮은 결과일수도 있으나 이 실험에 서는 패킷당 30ms의 단위로 전송하기 때문에 기본적인 지연시간이 있다. 실험 1에서 30ms의 간격으로 1000개의 패킷을 전송했으므로 적어도 30초의 전송 시간은 기본적으로 발생하는 것이다. 위의 결과로 MSSC 메커니즘이 일반적인 리눅스 시스템보다 성능이 향상됨을 알 수 있다.

4.2 과부하시 MSSC 메커니즘 성능 평가-실험 2

서버 시스템이란 여러 운영상의 데몬이나 프로세스도 함께 운영되고 관리되는 것이다. 이러한 경우를 과부하라는 조건을 두어 MSSC 메커니즘의 성능을 측정해 보았다. 실험 1과 동일한 조건으로 수행하되, 다음과 같은 3가지 종류의 과부하를 서버 시스템에 주었다. 3가지 종류의 시스템 과부하는

- while 루프 50개를 응용 프로그램 상에서 주고
- 커널 컴파일을 수행하고
- 순간적인 과부화를 위해 Xwindow를 띄웠다.

이러한 과부하 환경하의 MSSC 메커니즘 성능을 일반적인 리눅스와 비교한 결과가 <표 4>와 (그림 10)이다.

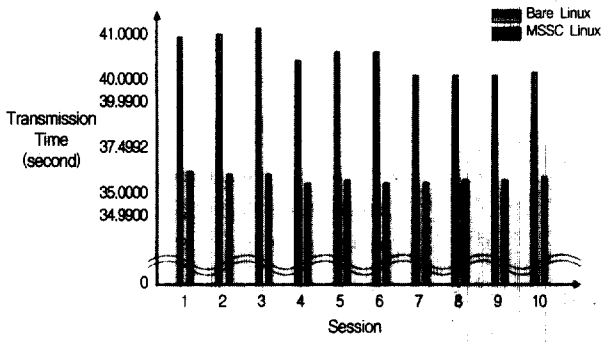
<표 4>과 (그림 10)에서 보듯이 서버 시스템의 과부하 조건에서도 MSSC 메커니즘의 리눅스가 일반적인 리눅스보다 약 14%의 성능 향상을 보였다. 실험 1과 같이 과부하를 주지 않은 상태보다 월등히 느려지지 않았다. 이러한 실험으로 볼 때 본 논문에서 제안한 MSSC 메커니즘 리눅스는 견고성에서도 안정적인 성능이 입증되는 것이다.

실험 1과 실험 2에서 보는 바와 같이 커널 내부에 MSSC 메커니즘을 내장한 리눅스의 전송 성능이 일반적 리눅스보다 전반적으로 약 12.5%~14%의 성능 향상을 보였다.

본 논문은 리눅스 환경 서버 시스템에서 멀티미디어 데이터 전송을 커널 내부에 내장할 경우 기존 리눅스 환경의 전송보다 성능 향상이 뛰어남을 보였다. 사용자 응용 프로그램에서 수행하는 RTP 프로토콜이 MSSC 메커니즘과 함께 커널 내부에 있으므로 적어도 2번 이상 발생하는 데이터 복사가 1번으로 줄었고 문맥 교환도 커널 내부에서 발생하므로

〈표 4〉 과부하시 일반적 리눅스와 MSSC 메커니즘 리눅스의 세션별 전송 속도 비교

Session		1	2	3	4	5
소요 시간 (초)	일반리눅스	41.10991	41.11984	41.11984	40.8087	40.99878
	MSSC 리눅스	35.00197	35.00002	35.00805	34.9987	34.99992
Session		6	7	8	9	10
소요 시간 (초)	일반리눅스	40.99898	40.00853	40.00873	40.00893	40.00948
	MSSC 리눅스	34.99797	34.99799	34.99804	34.99812	34.999925



(그림 10) 과부하시 일반적 리눅스와 MSSC 메커니즘 리눅스의 세션별 전송 속도 비교

그 수행 속도가 응용 프로그램보다 빠르게 일어난다.

<표 5>는 실험 결과를 바탕으로 MARS 메커니즘, 일반적 리눅스, 본 논문의 제안인 MSSC 메커니즘 리눅스를 비교한 표이다.

〈표 5〉 제안된 모델과 기존 모델간의 비교

(○ : High △ : Low × : None — : None Compare)

	MARS 메커니즘	일반적 리눅스	MSSC 리눅스
Read/Write 이득	○	×	○
전송 버퍼의 수정	○	×	△
전송 속도의 향상	—	×	○
RTP 패킷 처리	×	×	○
커널 메모리 소비	○	×	△
다른 장치와의 유연성	△	○	○

이러한 성능 향상을 보인 MSSC 메커니즘을 이용하여 멀티미디어 데이터 서버 시스템에 접목한다면 멀티미디어 서비스에 대한 사용자의 요구를 충족시킬 수 있을 것이다.

5. 결론 및 향후 연구 과제

멀티미디어 서비스를 제공하는 서버 시스템이나 네트워크의 발전보다 훨씬 빠른 속도로 사용자가 요구하는 데이터의 양이 증가하고 있다. 인터넷을 통한 전화, 화상 회의, 원격 교육 등이 그것이다. 서버 시스템에서의 성능 향상 즉, 라우터와 VoIP 전용 게이트웨이 등 실시간 전송을 보장해야 하는 멀티미디어 전용 서버의 연구가 활발히 진행중에 있고, 또한 사용자의 요구를 충족시키기 위해 서버 시스템과 네트워크 대역폭이라는 두 가지 방향으로 많은 연구가 진행되고 있다. 본 논문에서는 이러한 요구에 의해 멀티미디어 데이터의 효율적 네트워크 전송 지원을 위한 MSSC 메커니즘을 제안하였다. MSSC 메커니즘을 이용하여 실제 멀티미디어 데이터 생성과 전송은 커널 내부의 MSSC 메커니즘이 담당하게 되고, 응용 프로그램은 휴지 상태로 들어갈 수 있도록 설계하였다. 따라서 커널 공간에서 사용자 공간으로의 데이터 복사가 필요하지 않고, 이를 수행하기 위한 멀티미디어 데이터 전송의 사용자 프로세스를 스케줄링 할 필요가 없는 성능상의 이득이 있었다.

본 논문에서의 실험 1, 2의 결과에서 보듯이 리눅스 환경하에서 RTP 프로토콜을 사용자 응용 프로그램에서 수행하지 않고 커널 내부에 MSSC 메커니즘과 함께 내장함으로써 기존의 리눅스 환경보다 전송 속도가 약 12.5%~14% 정도의 성능 향상이 있었다. 이는 인터넷 상황과 멀티미디어 데이터를 요구하는 각각의 서버 환경(VoIP 서버, 오디오서버, 비디오서버, 화상 회의용 서버 등)에 따라서 적절하게 응용할 수 있고 늘어나는 멀티미디어 데이터를 효과적으로 처리할 수 있을 것이다.

현재의 일반적인 운영체제 네크워크 전송 단계는 기존의 텍스트 위주의 데이터 처리에 적합하게 제작되어 있어서 멀티미디어 데이터를 처리하기에는 적합하지 않다[7]. 향후 연구 과제로는 이러한 속도면에서 향상된 MSSC 메커니즘을 실시간 패킷에 우선 순위를 지원하는 스케줄러를 연구하여 합치면 멀티미디어 서버 시스템의 핵심 기술로 활용할 수 있을 것이다.

참 고 문 헌

[1] Millind Buddhikot and Guru Parulkar, "Efficient Data Layout, Scheduling and Playout Control in MARS," ACM/Springer Multimedia Systems Journal, pp.199-211, Vol.5, No.3, 1997.

[2] Millind Buddhikot, Guru Parulkar and Gopalakrishnan, R., "Scalable Multimedia-On-Demand via World-Wide-Web (WWW) with QOS Guarantees," Sixth International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV96, Zushi, Japan, pp.23-26, April, 1996.

[3] H. Schulzrinne and S. Casner, "RTP : the Real-time Transport Protocol," Audio-Video Transport Working Group, RFC 1889, January, 1996.

[4] Jose Carlos Brustoloni, "Effects of Data Passing Semantics and Operating System Structure on Network I/O Performance," Ph.D. Dissertation, Technical Report CMU-CS-97-176, School of Computer Science, Carnegie Mellon University, September, 1997.

[5] Moti N. Thadani and Yousef A. Khalidi, "An Efficient Zero-Copy I/O Framework for UNIX," Technical Report, SMLI TR-95-39, Sun Microsystems Lab, Inc., May, 1995.

[6] Rosenberg, J. and Schulzrinne, H., "Timer Reconsideraion for Enhanced RTP Scahability," INFOCOM '98, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings IEEE, pp.233-241, Vol.1, 1998.

[7] Puneet Sharma, Deborah Estrin, Sally Floyd and Van Jacobson, "Scalable Timers for Soft State Protocols," INFOCOM '97, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Driving the Information Revolution, Proceedings IEEE, pp.222-229, Vol.1, 1997.

[8] Jose Carlos Brustoloni and Peter Steenkiste, "Evaluation of Data Passing and Scheduling Avoidance," in Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97), IEEE, pp.101-111, May, 1997.

[9] Buddhikot, M. and Chen, X., "Project MARS : WWW based Scalable, Interactive Multimedia Recording and Playback Services," Winner of the first prize at the Research Poster Competition, ACM SIGCSE/SAC'97, San Jose, CA, Feb-Mar, 1997.

[10] P. Druschel and L. L. Peterson, "Fbufs : A highbandwidth cross-domain transfer facility," In Proceedings of the Fourteenth ACM Symposium on Operating System Principles, pp.189-202, Dec. 1993.

[11] Kevin Fall and Joseph Pasquale, "Improving Continuous-Media Playback Performance with In-kernel Data Paths," Proceedings of the International Conference on Multimedia Computing and Systems, 14-19, Boston, Massachusetts. IEEE-CS, pp.100-109, May, 1994.

[12] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers and S. Eggers, "Extensibility safety and performance in the SPIN operating system," Proceedings of the fifteenth ACM symposium on Operating systems principles, pp.267-283, December, 1995.

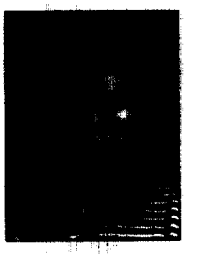
[13] Milind M. B., Dakang W., Guru M.P. and Xin J. C., "Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS," Multimedia Computing and Systems, Proceedings IEEE International Conference on, pp.326-337, 1998.

[14] M. Beck, H. Bohme, M. Dziadzka, U. Junitz, R. Magnus and D. Verworner, "Linux Kernel Internals," 2nd Edition, Addison-Wesley, 1999.

[15] Richard M. S., Roland M. and Andrew O., "The GNU C Library Reference Manual," Edition 0.05, DRAFT last updated, 3 1993 for version 1.07 Beta.

[16] W. Richard Stevens, "TCP/IP Illustrated," Vol.3, Addison Wesley, April, 1996.

[17] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel," O'Reilly, January, 2001.



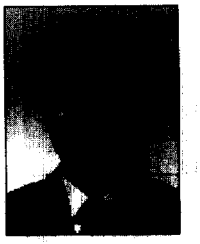
남 상 준

e-mail : sjnam@netlab.korea.ac.kr

2000년 한성대학교 컴퓨터공학과 공학사

2000년~현재 고려대학교 컴퓨터학과 석사과정 재학중

관심분야 : 컴퓨터 네트워크, 실시간 시스템, 서버 QoS, 리눅스, 무선 네트워크 등.



이 병 래

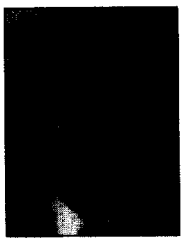
e-mail : brlee@netlab.korea.ac.kr

1998년 고려대학교 컴퓨터학과 학사

2000년 고려대학교 컴퓨터학과 석사

2000년~현재 고려대학교 컴퓨터학과 박사과정 재학

관심분야 : 암호, 네트워크 보안, 이동통신.



박 남 섭

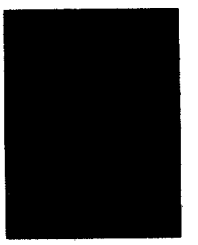
e-mail : nspark@korea.ac.kr

1998년 부산외국어대학교 컴퓨터공학과 학사

2000년 부산외국어대학교 컴퓨터공학과 석사

2000년~현재 고려대학교 컴퓨터학과 박사과정 재학

관심분야 : 네트워크 보안, 결함 허용 시스템, 네트워크 모니터링.



이 윤 정

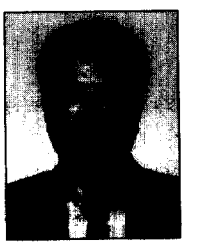
e-mail : genuine@netlab.korea.ac.kr

1993년 숙명여자대학교 전산학과 학사

1998년 숙명여자대학교 전산학과 석사

2000년~현재 고려대학교 컴퓨터학과 박사과정 재학

관심분야 : 컴퓨터 네트워크, 네트워크 보안, 이동통신.



김 태 윤

e-mail : tykim@netlab.korea.ac.kr

1981년 고려대학교 산업공학과 학사

1983년 Wayne state University 전산과학과 석사

1887년 Auburn University 전산과학과 박사

1998년~현재 고려대학교 컴퓨터학과 교수

관심분야 : 전자상거래, 컴퓨터 네트워크, EDI, 이동통신 등.