2000년 한국정보처리학회 추계 학술논문발표집 제7권 제2호

멀티미디어 패킷 전송에 적합한 I/O 서브시스템 설계 및 구현

남상준, 이병래, 김태윤
고려대학교 컴퓨터학과
e-mail:sjnam@netlab.korea.ac.kr

Design and Implementation of I/O Sub-System based on Multimedia Packet Transfer

Sang-Jun Nam, Byung-Rae Lee, Tai-Yun Kim
Dept. of Computer Science and Engineering, Korea University

요 약

최근 멀티미디어 데이터에 대한 요구는 사용자가 증가함에 따라 데이터 양도 증가하고 있다. 이에 반해 서버시스템이나 네트워크의 대역폭은 이러한 서비스 요구를 충족시키기에 미흡하다. 본 논문에서는 리눅스 환경에서 일반적인 UDP(User Datagram Protocol) 전송 메커니즘을 SIO(Special Input/Output) 시스템 콜을 커널 내부에 추가하였다. UDP전송을 커널 내부에서 수행함으로써, 사용자 모드와 커널 모드 사이의 데이터 복사의 횟수와 문맥 교환을 줄였다. 커널 내부에서 수행하게 SIO 시스템 콜을 설계하고 구현함으로써 일반적인 리눅스 환경보다 약 31%의 성능향상을 보았다. 본 논문에서는 SIO와 같은 효과적인 커널 내부의 전송 시스템 콜을 사용함으로써 멀티미디어 관련 서버에 적용할 수 있도록 하였다.

1. 서론

최근 멀티미디어 수요에 따른 데이터 양이 급속히 증가하면서 서버시스템의 부하가 증가하고 있다. 이러한 요구를 충족시키는 서버시스템은, 지금까지의 한정적이고 일반적인 범위에서 벗어나, 실시간 처리 능력과 대용량의 데이터를 효율적으로 전송하는 새로운 서버 시스템이 필요하다[1].

멀티미디어 데이터와 같은 서비스에서 성능향상에 요구되어지는 것은 대역폭과 문맥 교환이다. 이런 서버시스템은 저장 장치와 네트워크 하부시스템 사이에서 주기적으로 데이터를 주고받는다. 기대되는 작업에 대해 서버시스템은 높은 성능으로 디자인되고 전송되어야 한다. 그러나, 일반적으로 존재하는 운영체제는 이러한 성능개선을 보장하거나, 저장장치와 네트워크 하부시스템 사이의 효과적인 경로를 제어하지는 않는다. 대다수의 애플리케이션들은 사용자 모드와 커널 모드에서 과도한 데이터 복사가 발생한다. 멀티미디어 스트림 전송과 같이 한번 접속된 서비스가 다른 변경 없이 연속적으로 전송된다면, 데이터 복사의 횟수를 줄이고 빠른 데이터 경로와 전송을 이룰 수 있을 것이다[2][3].

본 논문에서는 빠른 전송을 수행하기 위해 UDP 전송 메커니즘을 커널 내부에서 이루어지게 SIO(Special Input/Output) 시스템 콜을 설계하고, 리눅스 환경에서 구현과 성능 평가를 한다.

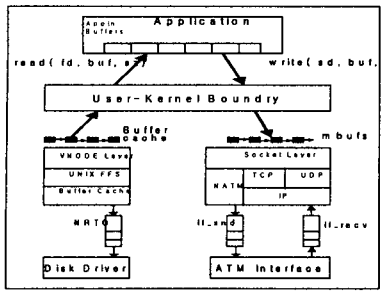
본 논문의 구성은 2장에서 관련 연구를 분석하고, 3장에서 새로운 네트워크 메커니즘인 SIO 시스템 콜을 설계, 구현하고 일반적 서버와의 비교, 분석한다. 4장에서는 본 논문의 결과 및 향후 과제에 대해 기술한다.

2. 관련연구

2.1 일반적인 I/O 메커니즘

<그림1>은 현재 UNIX 운영체제 시스템에서의 네트워크 I/O 시스템의 계층구조를 보여주고 있는, 애플리케이션에서 데이터를 요구하고 그 데이터가 네트워크 인터페이스로 전송되는 일반적인 과정이다. <그림1>의 서버는 사용자 공간에서 네트워크 장치까지는 두 단계의 데이터 복사가 일어난다.

첫 번째 복사는 read() 콜이다. 이것은 커널의 버퍼



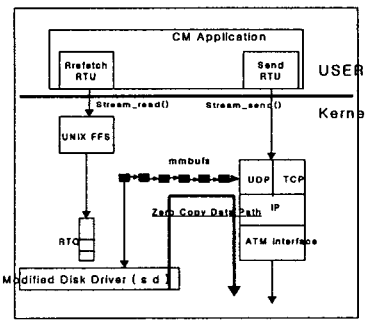
<그림1> 일반적인 서버의 전송 메커니즘

캐쉬에서 사용자버퍼공간까지 데이터를 옮기는 경우이다. 두 번째 복사는 write() 콜이다. 이것은 소켓 계층에서 발생하는 복사로 사용자공간버퍼에서 커널에 있는 mbuf chain까지의 전송을 말한다. 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.

이들은 메모리 공간을 많이 사용한다. 또한, 데이터는 종종 이것이 재 사용될 수 있음에도 불구하고 대체될 수 있다. 성능상의 불이익이 잉여 복사로 인해 이루어지게 되는 것이다.

2.2 MARS 모델

일반적인 서버전송 메커니즘에서와 같이 잦은 데이터 복사를 효율적으로 관리하는 것이 MARS(Massively-parallel And Real-time Storage) 모델[4]이다.



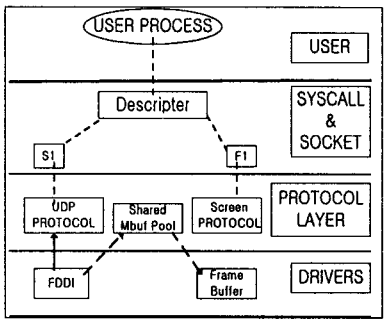
<그림2> MARS의 모델

<그림2>는 디스크에서 네트워크 I/O로의 새로운 데이터 경로와 제어에 대해 보여준다. MARS의 특징을 살펴보면 다음과 같다.

디스크와 네트워크 I/O 사이의 데이터 전달은 Zero Copy를 위해 mmbufs(Multimedia Memory Buffer)라는 새로운 버퍼 관리 시스템 콜을 구현하고 디자인한다. <그림2>에서 보듯이 이 새로운 데이터 경로는 잘 만들어진 read()/write()라는 새로운 인터페이스를 통해 데이터 경로를 액세스하는 오래된 버퍼 캐쉬와도 잘 공존한다.

2.3 스플라이스 메커니즘

스플라이스(Splice) 메커니즘[5]은 I/O 디바이스들 사이의 데이터 경로를 직접적으로 다루는 커널을 설정하기 위해 사용하는 시스템 콜이다. 전통적인 read()와 write() 인터페이스들을 발생시키는 일반적 시스템에서 디바이스들 사이에 흐르는 I/O 데이터는 사용자 프로세스의 주소 공간을 통한다. 스플라이스에 관계하여 어떤 애플리케이션은 메모리 주소를 참조하지 않는 기술자(Descriptors)를 사용하여 운영체제에 관계하여 직접적으로 데이터 근원지와 목적지 사이의 관계를 표현한다. <그림3>를 보면 UNIX에서 구현한 스플라이스 메커니즘의 모습이다.



<그림3> 스플라이스 메커니즘

데이터 근원지와 목적지는 UNIX 파일 기술자에 의해 구체화된다. 소켓 버퍼가 준비가 되었을 때 사용자 프로세스는 소켓버퍼 속에 있는 데이터를 그 자신의 연속적인 사용자 주소 공간까지 복사하면서 그것의 커널 내부에서 수행을 계속하도록 알려준다.

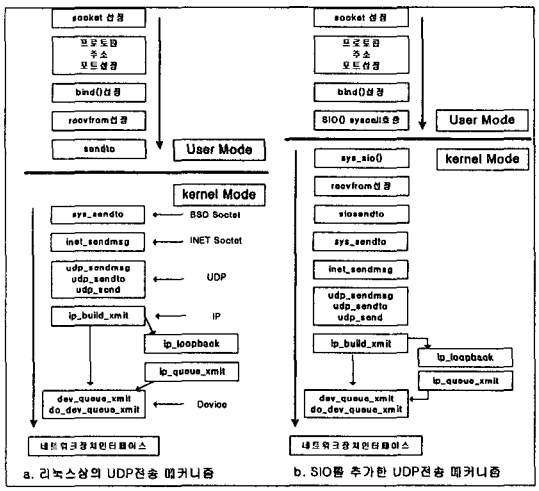
3. SIO 메커니즘 구현 및 성능평가

3.1 SIO 메커니즘 설계

앞장의 MARS 모델과 스플라이스 메커니즘을 살펴보면 멀티미디어 환경의 서비스에 대해 일반적인 서버시스템보다 빠른 전송을 하기 위함이다. 그러나, 일반적인 운영체제는 그렇지 못하다. 일반적인 운영체제의 문제점을 보면 다음과 같다.

사용자모드와 커널 모드 사이에서의 잦은 데이터 복사가 일어난다는 것이고, 둘째는 애플리케이션 수행시 발생하는 문맥 교환은 커널의 작업보다 많은 오버헤더를 발생시킨다.

멀티미디어 데이터를 전송하는 서버에서 오디오나 비디오 데이터의 경우는 변경이 많이 일어나지 않는 연속적인 스트림의 전송이 요구된다. 이와 같은 전송은 사용자 모드에서 네트워크를 연결한 뒤 커널 모드에서 연속적인 전송을 하게 된다면 전송시간을 단축시킬 수 있을 것이다. 리눅스의 일반적인 서버 시스템의 UDP 전송 메커니즘은 <그림4>의 a 와 같고, 이 논문에서 제시하는 서버시스템의 커널 모드에서의 전송 방법은 <그림4>의 b 와 같다.



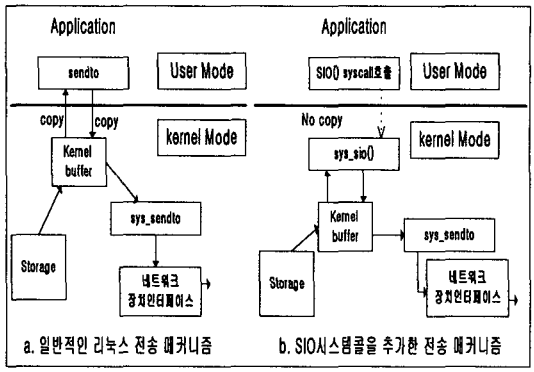
<그림4> SIO시스템 콜 추가 전과 후의 전송 메커니즘의 흐름

<그림4>의 b 와 같이 사용자 애플리케이션에서 소켓 설정과 bind를 행한 후, 사용자 애플리케이션은 SIO 시스템 콜을 호출한다. 커널 내부에 삽입된 sys_sio()함수가 다음 전송 단계인 recvfrom()과 sendto()를 수행하므로써 UDP 전송이 커널 내부에서 이루어지게 된다.

<그림4>의 사용자모드와 커널모드의 경계부분에서의 데이터 복사의 경로를 좀더 자세히 보면 <그림5>와 같다. <그림5>의 a 그림은 일반적인 리눅스의 전송 과정으로 send()를 호출하면서 커널 모드와 사용자모드의 경계에서 데이터 복사가 발생한다.

그러나, <그림5>의 b 와 같이 SIO 시스템 콜을 사용한 전송 메커니즘에서는 send()를 커널 내부에 추가하므로써 데이터 복사를 줄이는 과정을 볼 수 있다.

이 논문에서 제안한 <그림5>의 b 와 같은 전송 메커니즘을 구현은 3.2 절과 같다.



<그림5> SIO 시스템 콜 추가 전과 후의 커널과 사용자모드 경계지점 전송 메커니즘

3.2 SIO 구현

<그림4>의 b 와 같은 모델을 가진 운영체제는 존재하고 있지 않다. 리눅스 시스템이 커널의 소스를 공개방식으로 지향하는 운영체제이므로 새로운 UDP 전송 메커니즘을 구현해 보았다[6][7].

커널 내부에 UDP 패킷 전송을 담당하는 SIO 시스템 콜을 등록한다.

등록한 SIO 시스템 콜 내부에 UDP 패킷전송 메커니즘의 알고리즘은 다음과 같다.

```
asmlinkage int sys_sio()
{
        ...
        sioread(fd, buf, len);
        server_len = siosendto(buf);
        ...
    return 1;
}
```

위와 같이 SIO 함수는 커널 내부에서 버퍼를 읽어와서 siosendto함수의 파라미터로 넘긴다.

커널을 컴파일하고 사용자 애플리케이션에서 커널 내부에 새로 만든 sio() 시스템 콜을 호출한다.

서버 애플리케이션의 메커니즘은 다음과 같이 구현되었다.

```
/* file name : SioUdpServ.c */
_syscall2(setting parameter)
int     main(void)
{
        socket 설정;
        프로토콜, 주소, 포트 설정;
        ...
        bind 설정;
        start time;
        n = sio(setting parameter);
        end time;
}
```

3.3 SIO 성능평가

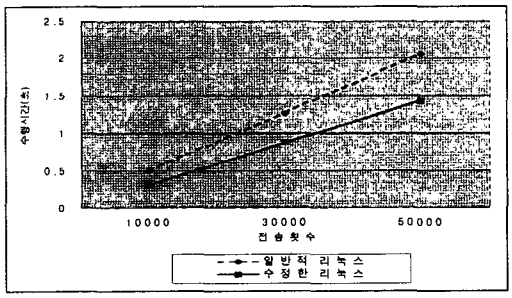
일반적인 UDP 전송과 동일하게 socket() 과 bind() 설정 후 새로운 시스템 콜을 추가하는 방법으로 구현한 후, 수행횟수를 통하여 전송속도를 비교하였다. 구현 환경은 다음과 같다.

① 운영체제 : 레드햇 리눅스6.1
② 리눅스 커널 버전 : 2.2.12
③ 구현에 사용한 언어 : C 언어
④ 서버 주소 : 163.152.39.115
⑤ 클라이언트 주소 : 163.152.39.120
⑥ 수행 시나리오 : 클라이언트의 요청을 받고 서버 호스트는 정해진 횟수만큼 데이터 패킷을 전송한다.

<표1> 전송 횟수별 성능 비교1

	수행하는데 걸린 시간(초)		
전송 횟수(회)	10000	30000	50000
일반적 리눅스	0.50	1.27	2.06
수정한 리눅스	0.31	0.87	1.43

테스트는 각각 100회씩 반복해서 평균값을 구한 값이다.



<그림6> 전송횟수별 성능비교2

<표1>과 <그림6>에서 보는 바와 같이 커널 내부에 시스템 콜을 추가한 수정된 리눅스의 전송횟수를 보면, UDP 전송을 커널 내부에서 수행하는 것이 일반적 리눅스보다 약 31%의 성능향상을 보였다.

본 논문은 리눅스 환경 서버 시스템에서 UDP전송을 커널 내부에 삽입할 경우, 기존의 리눅스 환경의 전송보다 성능향상이 뛰어남을 보였다. 왜냐하면, 사용자 애플리케이션에서 수행하는 시스템 콜이 커널 내부로 들어와 있으므로 적어도 2번 이상 발생하는 데이터 복사가 1번으로 줄어들었고, 문맥 교환도 커널 내부에서 발생하므로 그 수행속도가 애플리케이션보다 빠르게 일어난다. 이로써 인터넷 확산과 다양한 멀티미디어 서버를 요구하는 사용자들에게 빠른 전송을 할 수 있다.

4. 결론 및 향후 연구 과제

본 논문에서는 리눅스 환경 하에서 UDP기반 전송함수를 사용자 애플리케이션에서 수행하지 않고, 커널 내부에 내장함으로써, 기존의 리눅스 환경보다 전송 속도가 향상되었다. 인터넷 상황과 멀티미디어 데이터를 요구하는 각각의 서버 환경(VoIP 서버, 오디오서버, 비디오서버, 화상 회의용 서버 등)에 따라서 적절하게 응용할 수 있다.

서버 시스템에서의 성능향상, 라우터, VoIP 전용 게이트웨이 등 실시간 전송을 보장해야 하는 멀티미디어 전용 서버의 연구가 활발히 진행중이다.

이를 바탕으로 VoIP 서버를 기반으로 이 시스템 콜을 사용한 UDP패킷을 RTP 헤더를 첨부하여 보다 유연하고 패킷손실이 없는 전송을 보장하는 서버 연구가 향후 연구 과제이다.

<참고 문헌>

[1] Guru M.P.: "Multimedia On-Demand Server and services": Washington University, Director of Applied Research Lab.
[2] Milind M.B., Gurudatta M.P., R. Gopalakrishnan: "Scalable Multimedia-On-Demand via World-Wide-Web(WWW) with QOS Guarantees": Washington University, Computer Communications Research Center, Dept. of Computer Sci.
[3] Jose C.B.:, "Effects of Data Passing Semantics and Operating System Structure on Network I/O Performance": Rice University, 1997
[4] Milind M.B., Dakang W., Guru M.P., Xin J.C.: "Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS": Washington University Dept. of Computer Sci.
[5] Kevin Fall and Joseph Pasquale: "Improving Continuous-Media Playback Performance with In-kernel Data Paths": California University, Computer System Lab., Dept. Computer Science and Engineering. 1994
[6] M.Beck, H.Bohme, M.Dziadzka, U.Junitz. R.Magnus, D.Verworner.: "Linux Kernel Internals": 2nd Edition, Addison-Wesley 1999
[7] Richard M.S., Roland M., Andrew O.: "The GNU C Library Reference Manual" Edition 0.05 DRAFT last updated 3 1993 for version 1.07 Beta