제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

메모리 복사를 최소화 하는 효율적인 시스템 호출 인터페이스에 관한 연구

송창용 ° 김은기
한밭대학교 정보통신전문대학원
e-mail : netking33@netian.com, egkim@hanbat.ac.kr

A study on the efficient system call interface supporting minimum memory copy

ChangYong Song° EunGi Kim
Graduate School of Information & Communications, HANBAT National University

요 약

UNIX/LINUX 시스템에서 로컬 파일 시스템의 파일 데이터가 네트워크를 통해서 원격지 시스템에 전송되는 경우, 사용자와 커널(Kernel) 공간 사이에서의 메모리 복사가 적어도 2 회에 걸쳐 수행된다. 이러한 사용자와 리눅스(Linux) 커널 공간 사이에서 이루어지는 메모리 복사는 데이터 전송에 소요되는 시간을 증가시키고, 잦은 시스템 호출의 호출은 응용 프로세스와 리눅스 커널 간 문맥 교환(context switching)의 발생을 빈번하게 야기한다. 본 연구에서는 이러한 문제점들을 해결하기 위하여 필요한 경우 사용자와 리눅스 커널 사이에서의 메모리 복사를 수행하지 않고, 커널 공간 내에서의 메모리 복사를 최대한 제한할 수 있는 새로운 알고리즘을 제시한다.

1. 서론

최근 인터넷이 대중화되면서 사람들에 의해 교환되는 데이터의 양은 점점 더 증가하고 있다. 이러한 상호간의 정보 교환은 주로 파일 전송을 통해서 이루어진다. 그러나 현재 파일 전송을 위해서 사용되는 대부분의 응용 프로그램들은 송·수신되는 데이터의 일부분 또는 패킷(packet)의 헤더만을 처리하고, 전송되는 거의 모든 데이터들이 그대로 파일 시스템에 저장되거나 네트워크를 통해서 원격지 시스템으로 전송된다. 그 대표적인 예로서 FTP 와 같은 응용 프로그램을 들 수 있다.[7]

파일 전송을 수행하는 응용 프로세스는 크게 다음 두 가지로 파일 송·수신 동작을 수행한다.

1) 로컬 시스템 내의 파일을 원격 시스템으로 전송하는 경우; 먼저 로컬 파일 시스템으로부터 파일의 일부 또는 전체 데이터를 읽고, 네트워크를 통해서 원격 시스템으로 전달한다.

2) 원격 시스템과 연결이 설정된 네트워크로부터 파일을 수신하는 경우; 수신된 파일의 일부 또는 전체 데이터를 네트워크로부터 읽고, 로컬 파일 시스템에 저장한다.[2][3][4]

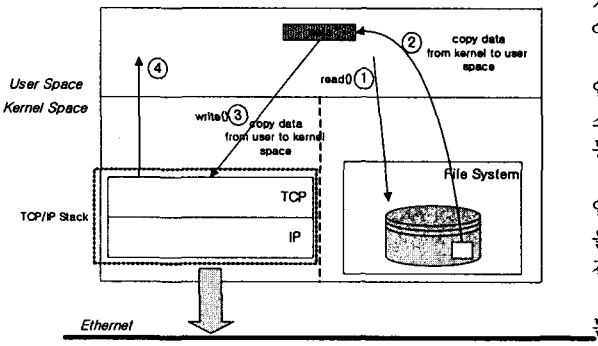
이 두 가지 동작이 이루어지는 동안 송·수신되는 파일 데이터는 응용 프로그램에서 전혀 이용되지 않는다. 단지 송신 또는 수신되는 단순한 의미를 지닌 데이터에 불과한 것이다. 그러나 이러한 단순한 의미를 지고 있는 데이터를 전송하기 위해서는 응용 프로세스와 커널(kernel)간에 적어도 2 번의 메모리 복사와 4 번의 문맥 교환(context switching)이 이루어지게 된다. 크기가 작은 파일을 송·수신하거나 파일 전송이 많지 않은 경우에는 응용 프로세스와 커널 간의 메모리 복사와 문맥 교환이 큰 문제점으로 여겨지지 않을 수 있다. 그러나 파일의 크기가 크고, 파일 전송이 빈번하게 이뤄지는 경우에는 잦은 메모리 복사와 문맥 교환이 시스템의 성능을 저하시키는 아주 큰 문제점으로 대두될 수 있다.[6]

본 논문은 전송되는 파일의 크기가 크고, 파일 송·수신이 빈번히 이루어지는 시스템에서 성능을 저하시키는 문제점인 응용 프로세스와 커널 간의 메모리 복

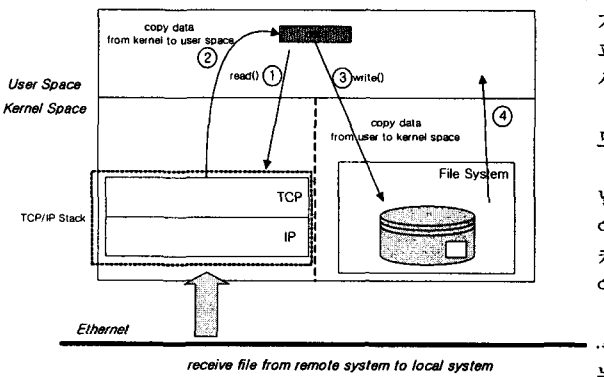
사와 문맥 교환 문제의 해결을 위해 커널 내부에서의 메모리 복사를 이용한 네트워크 파일 송·수신 기법을 제공한다.

2. 관련 연구

현재 파일 데이터의 송·수신을 수행하기 위해 사용되는 응용 프로그램들은 로컬 파일 시스템으로부터 파일 데이터를 사용자 메모리 공간으로 복사한 후 TCP/IP 스택(stack)으로 데이터를 전송하거나, TCP/IP 스택으로부터 데이터를 읽어 사용자 메모리 공간으로 복사하고, 사용자 메모리 공간에 있는 데이터를 로컬 파일 시스템에 저장하는 동작을 수행한다.[1]



(a) 로컬 파일을 원격지로 전송하는 경우



(b) 원격지의 파일을 수신하는 경우

[그림 1] 일반적인 파일 송수신 과정

[그림 1]은 현재 사용되는 파일 송·수신 응용 프로세스의 동작을 나타내고 있다. [그림 1]의 (a)는 로컬 파일 시스템의 파일을 원격지 시스템으로 전송하는 4 단계의 과정으로 이루어진다. 그 동작 과정은 다음과 같다.

-단계 1 : 응용 프로세스가 로컬 파일 시스템의 파일을 원격지에 전송하기 위해 로컬 파일 시스템으로 전송하고자 하는 파일의 데이터를 읽기 요청하는 단계이다.

-단계 2 : 응용 프로세스로부터 읽기 요청된 커널 메모리 공간의 파일 데이터가 사용자 메모리 공간으로 메모리 복사가 수행되어지는 단계이다

-단계 3 : 응용 프로세스가 로컬 파일 시스템으로부터 메모리 복사된 파일 데이터를 원격지 시스템에 전송하기 위해 원격지 시스템과 연결이 설정된 소켓에 쓰기 요청을 하는 단계이다. 이때 사용자 메모리 공간에 있는 파일 데이터가 커널 메모리 공간에 동적으로 할당된 tcp 패킷 구조체(sk_buff)로 메모리 복사된다.

-단계 4 : 응용 프로세스로부터 쓰기 요청된 동작을 모두 완료한 후 그 결과를 통보하는 단계이다.

단계 1 ~ 단계 4 의 모든 단계에서 사용자 모드와 커널 모드간에 동작 모드를 변경하는 문맥 교환이 이루어지고, 단계 2 와 단계 3 에서는 사용자 메모리 공간과 커널 메모리 공간 사이에서 메모리 복사가 각각 이루어진다.

[그림 1]의 (b)는 원격지 시스템으로부터 전송된 파일 데이터를 TCP/IP 스택으로부터 읽고, 로컬 파일 시스템에 저장하는 4 단계의 과정으로 이루어진다. 그 동작 과정은 다음과 같다.

-단계 1 : 응용 프로세스가 원격지로부터 전송된 파일 데이터를 수신하기 위해 TCP/IP 스택에 읽기 요청을 하는 단계이다. TCP/IP 스택은 원격지 시스템과 연결이 설정된 소켓을 통해서 데이터를 수신하게 된다.

-단계 2 : TCP/IP 스택에 수신된 원격지 파일 데이터는 응용 프로세스의 읽기 요청에 의해서 커널 메모리 공간의 데이터가 사용자 메모리 공간으로 메모리 복사가 수행되어지는 단계이다.

-단계 3 : 응용 프로세스가 TCP/IP 스택에서 수신된 파일 데이터를 로컬 파일 시스템에 저장하기 위해 쓰기 요청을 하는 단계이다. 이때 사용자 메모리 공간의 파일 데이터가 커널 메모리 공간에 매핑된 로컬 파일 시스템으로 메모리 복사가 수행된다.

-단계 4 : 응용 프로세스로부터 쓰기 요청된 동작을 모두 완료한 후 그 결과를 통보하는 단계이다.

단계 1 ~ 단계 4 의 모든 단계에서 사용자 모드와 커널 모드간에 동작 모드를 변경하는 문맥 교환이 이루어지고, 단계 2 와 단계 3 에서는 사용자 메모리 공간과 커널 메모리 공간 사이에서 메모리 복사가 각각 이루어진다.

위의 과정에서 알 수 있듯이 네트워크를 통해서 송·수신되는 파일 데이터의 대부분이 응용 프로세스에서는 이용되지 않는다. 단지 응용 프로세스는 데이터 교환에 필요한 로컬 파일 시스템의 파일을 입출력 동작을 수행하기 위해 열고, 원격지 시스템과의 통신을 위한 소켓 등을 생성하기 위한 네트워크 연결 설정 등 기본적인 초기화 동작만을 수행하게 된다. 파일 송·수신을 담당하는 대부분의 응용 프로세스의 동작은 파일 시스템으로부터 데이터를 읽어 TCP/IP 계층으로 전달하거나 또는 TCP/IP 계층으로부터 수신된 패킷 데이터를 읽어 로컬 파일 시스템에 저장하는 매개체 역할만을 담당한다. 이러한 과정에는 응용 프로 프로세스와 커널 메모리 공간간의 불필요한 파일 데이터에 대한 메모리 복사와 커널과 응용 프로세스간의 문맥 교환이 빈번하게 이루어지게 된다.

[그림 1]을 통해 파일 데이터를 송·수신하는 과정에서 단 한번의 파일 송·수신 동작에서 발생하는 메모리 복사와 문맥 교환의 회수는 다음과 같다. 사용자와 커널 메모리 공간 사이에서 2 번의 메모리 복사가 이루어지고, 사용자 모드와 커널 모드 사이에서의 4 번의 문맥 교환이 이루어진다. 파일 전송의 경우 이러한 방식은 실질적인 데이터 송·수신 동작 이외에 빈번한 메모리 복사 및 문맥교환이 이루어지므로 송 · 수신에 이용되는 파일의 크기가 크거나 파일 전송의 빈도수가 높은 시스템의 경우에는 성능을 크게 저하시키는 단점이 있다.

3. 효율적인 시스템 호출의 설계

3.1 수정된 시스템 호출 모델

기존 모델은 시스템 성능을 저하시키는 사용자 메모리 공간과 커널 메모리 공간 사이에서의 빈번한 메모리 복사와 문맥 교환의 문제점들이 있음을 살펴보았다.

본 논문은 이러한 문제점들을 해결하기 위해 응용 프로세스와 커널 사이에서 이루어지던 동작들을 모두 커널 내부에서 이루어지도록 동작 환경을 제한함으로써 파일 전송 시 시스템의 성능을 향상시키는 기법을 제안한다. 제안된 방식은 다음과 같은 장점들을 갖는다.

첫째, 사용자와 커널 메모리 사이에서의 메모리 복사를 커널 메모리 공간내에서 이루어지도록 제한함으로써 전송에 불필요한 시간을 줄일 수 있다. 로컬 시스템의 파일을 원격지 시스템으로 전송할 때, 기존 방식의 경우에는 두 번의 메모리 복사가 이루어졌다. 그러나 이러한 메모리 공간간의 데이터 복사를 커널 메모리 공간 내에서 처리하도록 함으로써, 그 회수를 1 회로 단축할 수 있다.

둘째, 새로운 시스템 호출을 통해 사용자 모드와 커널 모드간의 문맥 교환을 줄일 수 있다. 기존 방식의 경우, 한번의 데이터 전송을 위해 4 번의 문맥 교환이 발생되었다. 따라서 기존 모델에서는 전송되는 파일의 크기가 큰 경우, 문맥 교환은 파일의 크기에 비례하여 회수가 증가하게 된다. 그러나 본 논문에서 새롭게 제한된 시스템 호출을 통해서 하나의 파일을 전송 시 2 번의 문맥 교환만으로 데이터 전송이 완료된다.

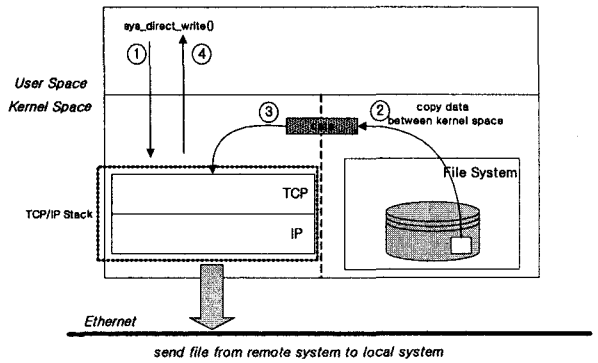
3.2 시스템 호출을 위한 커널 구조

본 논문에서 제안된 구조는 새로운 시스템 호출을 통해서 사용자와 커널 메모리 공간 사이에서의 메모리 복사와 문맥 교환의 회수를 줄이게 된다. 기존 방식에서 사용자 영역과 커널 영역을 넘나들면서 수행되었던 동작들을 커널 영역 내부에서 이루어지도록 제한하여 수행하도록 함으로써 파일 전송 시 소요되던 불필요한 시간을 줄이고 시스템의 성능을 향상시킬 수 있다. 이러한 전송 소요시간을 줄이고 시스템 성능의 향상을 위해 본 논문에서는 2 개의 새로운 시스템 호출을 제안한다.

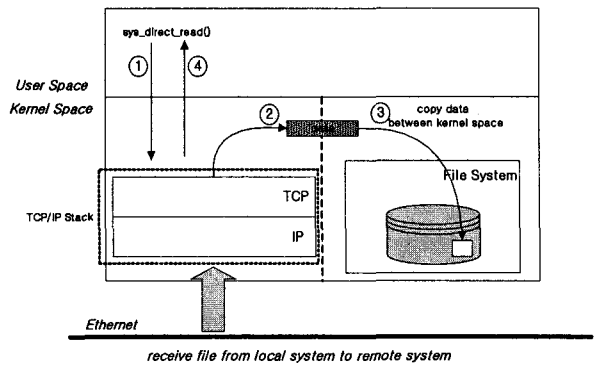
본 논문에서 제안된 두 개의 시스템 호출은 리눅스 시스템에서 구현되었으며, 리눅스 시스템의 커널 버전은 2.6.0 이다. [8]

[그림 2]는 본 논문에서 제안하는 새로운 시스템 호출을 적용하여 로컬 시스템과 원격 시스템간의 파일 송·수신을 수행하는 과정을 나타내었다.

[그림 2]의 (a)는 제안된 sys_direct_write(file_fd, socket_fd, length) 시스템 호출을 이용하여 로컬 시스템의 파일을 원격지 시스템으로 전송하는 4 단계의 과정으로 이루어진다. 그 동작 과정은 다음과 같다.



(a) 파일 송신 과정



(b) 파일 수신 과정

[그림 2] 새로운 시스템 호출의 과정

-단계 1 : 응용 프로세스로부터 호출된 sys_direct_write() 시스템 호출은 매개 변수로써 전송하고자 하는 파일의 파일 기술자(file descriptor), 원격지 시스템과 연결이 설정된 소켓 번호(socket number), 전송을 희망하는 데이터의 크기 등을 전달 받게 된다.

-단계 2 : 메모리 복사의 회수를 줄이기 위해 TCP 스택에서 할당하는 패킷 자료 구조(sk_buff)를 동적으로 할당한 후, 동적으로 할당된 sk_buff 의 주소를 파일 시스템에 전송한다. 파일 시스템의 파일 데이터를 전달된 주소에 메모리 복사를 수행함으로써 사용자 영역을 거치지 않고, 파일 데이터가 TCP 스택으로 직접 전달되게 된다.

-단계 3 : TCP 계층에서는 메모리 복사된 데이터를 포함하는 패킷의 헤더 값을 설정하여 IP 계층으로 전

달한다.

-단계 4 : 전송 요청된 크기 또는 파일의 크기 만큼의 데이터가 원격지 시스템으로 전송 완료되지 않은 경우에는 단계 2 와 단계 3 을 반복하여 수행한다. 요청된 크기 또는 파일의 크기만큼 모든 전송이 완료된 경우에는 실제로 전송된 크기를 응용 프로세스로 반환한다.

기존 파일 전송에서는 로컬 파일 시스템의 파일 데이터를 응용 프로세스에서 요청한 크기만큼 사용자 영역의 메모리로 복사하고, 이를 다시 원격지 시스템으로 전송하기 위해 동일한 파일 데이터를 TCP/IP 스택에 전달하기 위해 커널 영역의 메모리로 복사한다. 그러나 제안된 기법은 기존 파일 전송과는 달리 응용 프로세스 메모리 공간으로 로컬 파일 시스템의 파일 데이터를 메모리 복사하지 않고, 직접 커널 메모리 공간 내에서 한 번의 메모리 복사로 동작이 이루어지게 되고, 기존 파일 전송에서와 같이 한번의 전송 후 응용 프로세스로 동작 모드를 변경하지 않고, 요청된 모든 파일 전송이 완료된 경우 응용 프로세스 동작 모드로 변경하게 된다. 이와 같은 기법을 사용함으로써 최소한 1 번의 메모리 복사와 2 번의 문맥 교환을 줄일 수 있다.

[그림 2]의 (b)는 제안된 sys_direct_read(file_fd, socket_fd, length) 시스템 호출을 이용하여, 원격지 시스템으로부터 전송된 파일을 로컬 파일 시스템에 저장하는 것으로 그 동작 과정은 다음과 같다.

-단계 1 : 응용 프로세스로부터 호출된 sys_direct_read() 시스템 호출은 매개 변수로써 수신하고자 하는 파일의 파일 기술자, 원격지 시스템과 연결이 설정된 소켓 번호, 수신을 희망하는 데이터의 크기 등을 전달받게 된다.

-단계 2 : TCP/IP 계층으로부터 수신된 데이터를 포함하는 tcp 패킷 구조체(sk_buff)의 주소를 로컬 파일 시스템으로 전달한다.

-단계 3 : TCP/IP 계층으로부터 전달된 데이터의 주소를 이용하여 네트워크로부터 수신된 파일 데이터를 파일 시스템으로 메모리 복사가 이루어진다.

-단계 4 : 수신된 데이터를 로컬 파일 시스템에 저장 후, 저장된 데이터의 크기를 반환한다.

기존 모델의 경우 단계 2 와 단계 3 에서 사용자 메모리 영역과 커널 메모리 영역 간 메모리 복사가 이루어졌던 반면에 새로이 제안된 방식에서는 메모리 복사가 이루어지지 않았다. 또한 기존 모델의 경우 단계 1 ~ 단계 4 의 모든 단계에서 사용자 모드와 커널 모드간에 동작 모드를 변경하는 문맥 교환이 이루어졌으나, 새로이 제안된 방식에서는 단계 1 과 단계 4 에서 단 2 회의 문맥 교환이 이루어졌음을 알 수 있다.

기존 파일 전송에서는 원격지 시스템으로부터 수신된 데이터를 사용자 메모리 영역의 메모리로 복사하고, 이를 다시 로컬 파일 시스템으로 전송 전송하기 위해 동일한 파일 데이터를 커널 영역의 메모리로 복사한다. 그러나 제안된 기법은 기존 파일 전송과는 달리 원격지 시스템으로부터 수신된 데이터를 사용자 메모리에 복사하지 않고, 직접 커널 메모리 공간 내에서 한 번의 메모리 복사만으로 동작이 이루어지게 된다. 이와 같은 기법을 사용함으로써 최소한 1 번의 메모리 복사와 2 번의 문맥 교환을 줄일 수 있다.

본 논문에서 설계한 두 개의 시스템 호출을 이용한 파일 전송의 경우, 기존 파일 전송보다 메모리 공간 내에서의 메모리 복사 회수를 줄일 수 있고, 문맥 교환을 최소한으로 억제하여 파일을 전송할 것이다. 따라서 이와 같은 기능을 이용한 파일 전송을 수행할 경우, 시스템에 대한 최대 성능을 기대할 수 있다.

4. 결론 및 향후 계획

본 논문은 인터넷을 통한 파일 송·수신에서 발생할 수 있는 사용자 메모리 영역과 커널 메모리 영역간의 불필요한 메모리 복사와 문맥 교환의 문제점을 지적하고 이를 효과적으로 처리할 수 있는 방법으로 새로운 시스템 호출을 이용한 파일 송·수신 기법을 제안하였다.

본 논문에서 새로이 제안된 기법은 전송되는 파일의 크기가 크고, 파일 전송이 빈번하게 이루어지는 시스템에 보다 빠른 파일 전송 시간과 안정성을 제공함으로써 보다 효과적인 파일 전송을 제공한다.

또한 이전 방식의 파일 송·수신 기법을 그대로 적용하여 사용할 수 있다. 새로이 제안된 기법은 현재 적용되는 인터넷 프로토콜을 그대로 사용하고 있기 때문에 실제 적용에 있어서 무리가 없을 것으로 판단된다.

현재 본 논문에서 제안한 두 개의 시스템 호출을 이용한 파일 전송 기법과 기존의 파일 전송 기법을 비교하는 실험이 진행 중이다.

향후, 응용 프로세스를 이용하지 않고 로컬 시스템의 파일을 네트워크에 연결된 다른 시스템과 공유할 수 있는 파일 분산 저장 장치 등에 적용을 위한 추가적인 연구가 필요하다.[5]

참고문헌

[1] W. Richard Stevens, "UNIX Network Programming", 2sted. PrenticeHall, 1990.

[2] W. Richard Stevens, "TCP/IP Illustrated, Volume 1 The Protocols", Addison-Wesley, 1994.

[3] W. Richard Stevens, "TCP/IP Illustrated, Volume 2 The Implementation", Addison-Wesley, 1995.

[4] W. Richard Stevens. "Advanced Programming in the UNIX Environment", Prentice-Hall, 1992.

[5]Compaq Computer Corp,. Intel Corp., and Microsoft Corp. Virtual Interface Architecture Specification Draft Revision 1.0 http://www.viarch.org/, Dec. 1997

[6] D. D. Clark et al, "An Analysis of TCP Processing Overhead", IEEE Com. Mag., vol. 27, no. 6, Jun 89.

[7] RFC 959, "File Transfer Protocol", http://www.ieft.org

[8] The Linux Kernel Archives 2.6.0, http://www.kernel.org/