TCP/IP 에서의 Zero-copy 매커니즘 연구

채병수* , 차승주**

Study of Zero-copy Mechanism in TCP/IP

Byoung soo Chae*, Seung Ju Tcha**

요 약

인터넷 네트워크를 통한 상호연결망에서 전체시스템의 성능향상을 위해서는 메시지 전송 지연시간을 줄이는 방법으로 수행된 연구는 정보와 데이터 유통경로로서 통신망이 네트워크로 구축된 접속 장치들로 중계기, 브리지, 라우터, 게이트웨이, TCP/IP(Transmission Control Protocol/Internet Protocol)프로토콜에 이르게 되었다. 여기서 TCP/IP 프로토콜 제 3계층인 사용자 데이터 프로그램 프로토콜(UDP)과 전송 제어 프로토콜(TCP)의 전송제어 계층에서 버퍼 캐시를 사용하지 않으므로 통신 속도를 줄일 수 있는 무 복사(zero-copy)로부터 소프트웨어 통신 지연시간을 다루어 통신 성능을 개선하려는 것이다.

ABSTRACT

From the reciprocal connection by this Internet network researchs about the efficiency improvement of the whole system is accomplished with the method which reduces delays in message transmission. From here, we will do a comparative study between the user data program protocol (UDP) and the zero copy which does not use the buffer cache to fine out the valid method to improve the efficiency.

In this thesis, I will change the message copy from execution process of the buffer cache of the TCP/IP on Unix OS with process on Linux OS. The object of conversion is to show you that the zero copy which doesn't use the buffer cache from transfer control class improves the communication efficiency.

키워드 : Zero-copy, Tcp/Ip, IOCP, Buffer

Ⅰ. 서 론

컴퓨터와 데이터 통신의 발달은 근거리 통신망(LAN : Local Area Network)으로 시작해 도시지역 통신망(MAN : Metropolitan Area Network)과 더 넓은 지역 통신망(WAN : Wide Area Network)으로 발전되면서 인터넷 웹(WWW : World Wide Web)에 이르게 된다.

이 과정에서 컴퓨터통신시스템 사이에 송수신되는 정보가 잘못 인식되어 지거나 해석되는 통신 장애를 예방하기 위해서는 하드웨어 및 소프트웨어적으로 여러 가지 기술적 규약을 만들어 통신 개체들 간의 원활한 정보교환을 하도록 프로토콜을 만들게 되었다. 이 과정에서 인터넷 네트워크를 통한 상호연결망에서 전체시스템의 성능향상을 위해서는 메시지 전송 지연시간을 줄이는 방법으로 수행된 연구는 정보와 데이터 유통경로로서 통신망이 네트워크로 구축된 접속 장치들로 중계기, 브리지, 라우터,

* 강원대학교 컴퓨터과학과 박사과정 (bschae@kangwon.ac.kr)
** 관동대학교 전자통신공학과 박사과정

게이트웨이, TCP/IP(Transmission Control Protocol/ internet Protocol)프로토콜에 이르게 되었다.[1] 여기에서 TCP/IP 프로토콜 제 3계층인 사용자 데이터 프로그램 프로토콜(UDP)과 전송 제어 프로토콜(TCP)[2]의 전송제어 계층에서 버퍼 캐시를 사용하지 않으므로 통신 속도를 줄일 수 있는 무 복사(zero-copy)로부터 소프트웨어 통신 지연시간을 다루어 통신 성능을 개선하려는 것이다.

따라서 이 논문에서는 Unix 운영체제를 바탕으로 수행되어지는 TCP/IP의 버퍼 캐시의 실행과정에서 메시지 복사를 Linux 운영체제에서의 과정으로 전이시키어 전송제어 계층에서 버퍼 캐시를 사용하지 않는 무복사(zero-copy)로부터 통신성능의 개선을 목적 한다.론부터는

II. 관련연구

2.1 리눅스 커널

커널이란 운영체제의 핵심을 이루는 요소로서 컴퓨터내의 자원을 사용자 프로그램이 사용할 수 있도록 관리하는 프로그램이며 커널이 제공하는 기능은 프로세스 관리, 파일 시스템, 메모리관리, 네트워크로 구분되며, 사용자 프로그램들이 이러한 기능들을 정해진 규칙에 따라 사용하게 되는 컨텐츠 환경이다.[3], [4]

2.2 무 복사

Data는 한 지점에서 다른 지점으로 이동할 때, 여러 번의 복사(copy) 과정을 거치게 된다. 즉, 송신측에서는 User Process에서 커널 쪽으로, 수신측으로는 커널에서 User Process 방향으로 버퍼 복사(buffer copy)를 하며 이러한 I/O 설계시 복사 과정을 생략하려는 시도를 무 복사 (zero-copy)라 한다.

① 전형적 copy 방법

Data를 보내거나 받을 때는 I/O device 내에 있는 버퍼로부터 게이트 회로(flag register)의 승인 신호(pass)를 필요로 하지만, 대부분의 I/O system에는 사실상 사용자 버퍼(user buffer)와 커널 프로세서 사이에 한번 이상의 복사가 수행하게 된다. 특히, Network I/O의 경우에는 체크 섬(check sum) 연산의 추가로 2번의 복사 필요하게 되고, 복사 활동은 상당한 양의 메모리가 필요해지므로 프로토콜 layer와의 불필요한 상호작용과 Checksum 연산으로 지연 요인이 발생된다.

② Fast buffer

TCP와 같은 대부분의 Network Protocol Stack들은 Data를 받고, 전송할 때, Checksum 연산을 요구하는 구조를 가지고 있으나, Data가 커널의 Access 없이 I/O 장치로부터 사용자 버퍼로 직접 보내지는 Fast buffer[5]를 한다. 그런데, 이 시스템 호출이 될 때는 User의 연산이 쓰여지고 있는 버퍼를 수정하는 것을 막아야 한다. 이유는 Application이 I/O 연산이 계속 이어지면, 입출력 실행이 되자마자 응용프로그램이 계속하여 떠맡아야 하는 불필요한 실행 대기상태가 계속되기 때문이다. 그러므로 전형적인 가상메모리에서의 연산은 무 복사와 밀접한 관계가 된다.

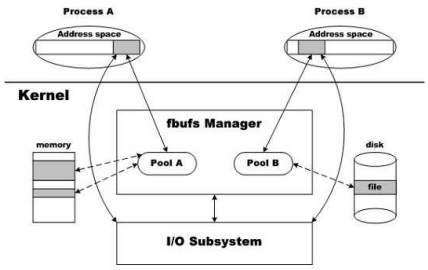

그림 1. 'fbuf'의 버퍼 배치
Fig 1. 'fbuf' of buffer located

③ M Buffer

멀티미디어 데이터와 같이 재사용 가능성이 적고, 데이터 블록의 복사가 시스템 내부적으로 과도하게 발생하여 버퍼 캐시의 실용성이 낮다는 인식에서 Storage의 Memory buffer를 사용하는 방법을 M buffer라 한다. 즉, 그림 1에서와 같은 사용자 버퍼로는 데이터가 복사되지 않고, 커널 내의 'mbuf'에 전송되어야 할 데이터 블록이 연결 리스트로 유지된다.

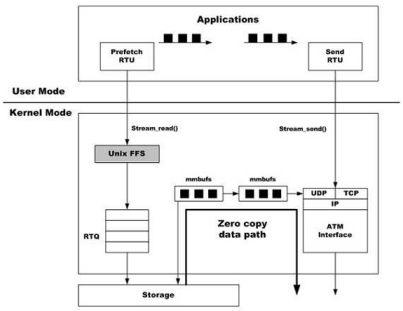

그림 2. 'mbuf'의 이용
Fig 2. use of 'mbuf'

III. Zero-copy의 요소와 구조 설계

3.1 데이터 복사의 구조

데이터의 이동 과정에서 데이터 복사가 발생된다는 것은 근원지(source)와 목적지(destination)가 존재한다는 것으로, 실제 시스템 내의 프로세스들이 데이터 저장(storage) 장치에서 프로세스로 복사하는 것이다. 프로세스간의 복사가 많이 발생은 온라인 네트워크에 의한 오늘의 컴퓨터통신 체제에서는 서로 다른 운영체제(OS)들에 의한 작업이 관련되기 때문이다.

즉, 이 같은 환경에서 여러 모듈이 서로 협동하여 작업을 수행하여야 하는 데, 이들 모듈들의 독립적인 프로세스는 독립적인 주소공간, 그 주소공간에 Mapping된 물리적 메모리, 그 메모리에 있는 코드와 데이터를 처리하는 CPU 명령의 실행시간을 할당하여야 할 등등이 공유되어야 하나, 지금 대부분의 운영체제들은 보호모드를 쓰며, 가상적 주소공간을 만들어놓고, 사용하는 주소공간에서만이 물리적 메모리를 Mapping하기 때문에 직접 호환될 수가 없게 된다.

3.2 데이터 이동의 구조

프로세스간 통신을 할 때 데이터를 복사하여 두는 파이프라인(pipeline)이란 공유영역을 두어서, 이를 복사해서 전달하도록 하며, 파이프는 한번의 복사를 위해 임시적으로 쓰이거나 두 프로세스간에만 형성이 된다. 그러므로 별도의 프로세스간 연결에는 별도의 파이프가 형성된다. 그림 3은 프로세스영역과 커널영역에서의 Pipe 생성을 보여준다.

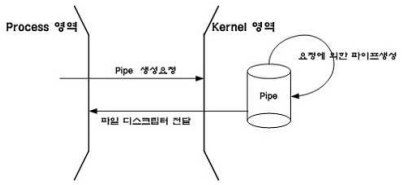

그림 3. 프로세스와 커널영역에서의 Pipe 생성
Fig 3. Process and Pipe generation of cornel zone

3.3 zero-copy의 매커니즘

네트워크상 데이터 이동은 데이터 복사 구조에서 이루어지므로, Zero-copy의 효율적 사용을 위한 필요조건은 효율적 동작 구조, 그리고 저장 장치(storage)로부터 네트워크 인터페이스까지 전달 과정에서 복사를 최소화하기 위한 전반적 데이터 이동 구조가 된다.

따라서 네트워크 송수신에 사용되는 Zero-copy 메커니즘에서 송신의 경우는 데이터를 tcp 스택의 버퍼에 복사하는 것이 아니라 데이터가 저장된 버퍼를 User-process에서 Kernel-process로 포인터 자체를 맡기는 위임이 발생된다.

현재 Storage-network Interface 간의 최소화 복사를 위해선 디스크로부터 직접 Network Interface로 보내는 방식을 사용하는 데, 리눅스의 'sendfile()' 함수를 사용하였으나 이 방식에서 몇몇 하드웨어에서 문제가 발생하였다. 이는 Storage Device가 부하에 취약했으며, 동일한 리소스에 대한 원격요청이 많았을 경우에 동일한 리소스를 반복적으로 여러 번 계속 읽어야만 하는 상황이 발생되었기 때문이다. 실 예로, 파일을 송신할 때 'storage'→'network'로 이해되나, 사실은 'storage'→'memory'....→'network'로 복사가 일어나면서 데이터가 이동되므로 이러한 불필요한 복사과정을 생략할 수 있도록, 중간 메모리를 거치게 하는 것이 Zero-copy의 구조이다.

3.4 zero-copy의 설계

Zero-copy의 설계목적은 시스템 메모리로 복사하는 과정을 생략하거나, 읽고(read) 쓰는(write) 것 자체를 생략하는 것이 불가능하기 때문에 장치에서 시스템 메모리로의 복사가 아니고 중간 메모리를 생략시키는 것이다.

컴퓨터와 컴퓨터간의 통신방식으로 동일한 파일을 10명에게 보낼 때에 그림 4에서와 같은 구조에서 표 1에서의 알고리즘으로 'sendfile()'함수와 같이 Storage에서 Network으로 복사되는 방식에서 m.buffer와 흡사한 Zero-copy로 수정한 그림 5에서와 같은 구조로 표 2 와 같이 할 수 있다.

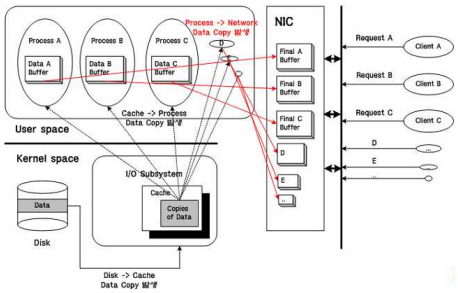

그림 4. Disc 파일 복사 사용자 접속구조
Fig 4. User interface structure of disc file copy

표 1. Disc 파일 복사
Table 1. disc file copy

'disk'→'cache'→'app-buffer0'→'temp buffer0'
→'network-buffer0'→'network-final-buffer0'
→'app-buffer1'→'temp buffer1'
→'network-buffer1'→'network-final-buffer1'
→'app-buffer2'→'temp buffer2'
→'network-buffer2'→'network-final-buffer2'
⋮
→'app-buffer9'→'temp buffer9'
→'network-buffer9'→'network-final-buffer9'

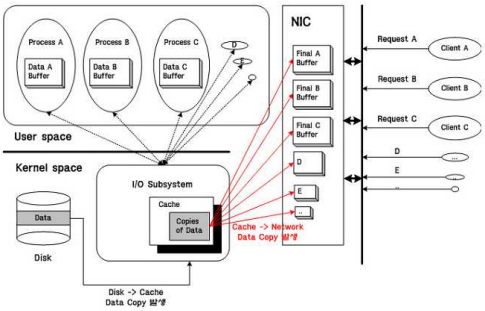

그림 5. m.buffer 방식의 사용자 접속구조
Fig 5. user interface structure of m.buffer

표 2. m.buffer에 의한 Zero-copy
Table 2. m.buffer's Zero-copy

'disk'→'network-final-buffer0'
'disk'→'network-final-buffer1'
'disk'→'network-final-buffer2'
⋮
'disk'→'network-final-buffer9'

표 3. Disc 공유캐시의 사용
Table 3. using the Disc shared cash

'disk'→'cache'→'shared-cache'
→'network-final-buffer0'
→'network-final-buffer1'
.
⋮
→'network-final-buffer9'

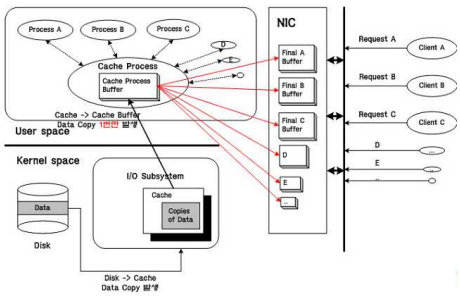

그림 6. Disc 공유캐시 이용 사용자 접속구조
Fig 6. using the Disc shared cash for user interface

이는 프로세스의 가상 주소공간에 특정한 '.dll'을 공유시키고, 해당 '.dll'이 각자의 가상주소공간에 캐시 프로세스의 공유메모리에 물리적으로 Mapping시키는 방법으로 하므로, 캐시 메모리를 복사하지도 않고, 직접 윈도우의 overlapped하는

I/O나, 유닉스의 'fbuf'로 활용하게 되며 그림 6에서와 같은 구조로 파일전송을 표 3 과 같이 할 수 있는 구조로 설계된다.

Ⅳ. 실험검증

4.1 실험 검증

① 시뮬레이션

사용한 시스템은 Pentium 233Mhz, 128Mb에서 Client에서 지속적으로 접속 요청의 상태를 유지하기 위해 fork함수를 이용한 다중 접속 서버로 구현하고, 이에 응답하도록 Linux 사용자 사양서(manual)에 의해서 'epoll'을 Edge Trigger와 Level Trigger의 인터페이스로 'fds'의 동작으로 Window 시스템에서와 Zero-copy를 표 1 을 실행하게 한 Linux 서버에서의 실험을 구분해 실행시키었다.

② 'epoll'의 실험

프로세스 기반의 다중 접속 서버 모델을 아래 그림과 같이 구현하였다.

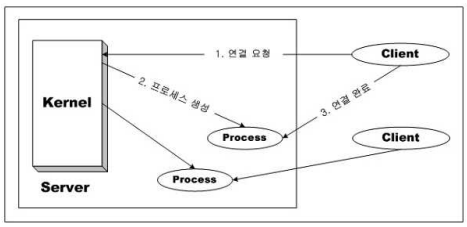

그림 7. 다중 접속 서버 모델
Fig 7. mult-interface Server model

그림 7에서 클라이언트가 연결 요청을 할 때마다, 서버는 새로운 프로세스를 생성해서 클라이언트의 연결 요청을 수락한다. 즉 클라이언트의 수와 프로세스의 수가 같게 된다. 표 4 는 fork 함수 호출을 통한 프로세스 생성방법의 예이다.

표 4. 프로세스 생성 알고리즘
Table 4. algorithm of generation Process

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(int argc, char **argv)
{
    pid_t pid;
    pid=fork();
    if(pid == -1)
    printf("fork 실패, 프로세스 id:%d\n", pid);
    if(pid == 0)      /* 자식 프로세스 */
        else          /* 부모 프로세스 */
return 0;
}
```

그림 8 에서 Real은 전통적인 Linux 시스템에서 사용되는 select()방식의 관찰이며, User와 System은 Linux 시스템에서 Zero-copy를 적용한 관찰이다. 이 실험에 이용한 'epoll.c'는 효과적으로 Polling을 구현하도록 Davide Libenzi(Copyright (C) 2001,...,2002: davidel@xmailserver.org)의 Open source를 참조했다.

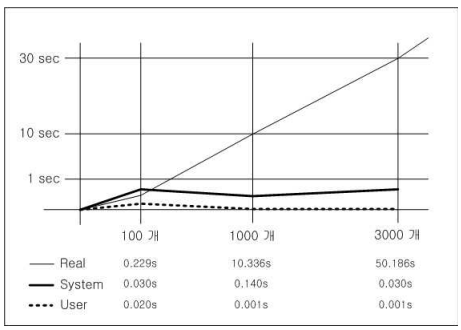

그림 8. 유의 시뮬레이션 결과
Fig 8. result of accept simulation

Ⅴ. 결 론

Network에 대한 읽기 쓰기 모두 Zero-copy가 적용되는 Overlapped I/O 가 적용 가능하며, 파일 읽기 쓰기도 같은 것이어서 Network 인터페이스로부터 디스크 쓰기의 중간에 복사를 최소화하는 것이 가능해 진다. 그러나 User-space의 버퍼를 활용하게 만들면 서버 프로세스가 커널에게 공유캐시

프로세스의 블록과 서버프로세스 자체의 블록을 둘 다 넘길 수 있도록 제작하여야 하기 때문에 매우 복잡해진다.

그리고 이러한 커널에서는 송신이 끝났을 경우에 어디에 통보를 해야 할지, 그리고 통보 받은 프로세스도, 이 버퍼가 누구의 것인지를 확인해야 하므로, 동적으로 생성되는 Command들을 전송해야 한다. 이는 항상 공유버퍼만으로 파일이 전송하는 것이 아니기 때문이다. 그러므로 송신을 할 때, Storage로부터 Network까지의 복사를 최소화하는 것은 수신시 많은 사용자가 복사를 통해 이동하는 대신 Zero-copy 메커니즘에 의해 통보 받은 데이터 자체를 바로 Disc 쓰기 Cache로 활용하도록 하므로, 전담 Thread를 두는 IOCP 방식을 채용해 LINUX에서의 Zero-copy를 구현 했다. 그러나, 이 논문에서는 Window에서와 LINUX에서의 비교를 그림 7 에서와 같이 컴퓨터통신의 네트워크에서의 유의 시뮬레이션으로 제안되었을 뿐 실제 적용으로부터 실측 실험의 평가와 검증 문제를 남기고 있다.

참고문헌

[1] Behrouz A. Forouzan, Sophia Chung Fegan, TCP/IP/ Protocol Suite 2nd Edition; McFraw-Hill Korea 2003.

[2] D. Clark ., V. Jaconbson, J Romkey, and H. Salwen. An analyis of TCP processing voerhead; IEEE Communication Magazine, 27(6):23-29, June 1989.

[3] Beck, m., Bohme, H., Dziadzka, M., Kunitz, U. Magnus, R. and Verworner, D., Linux Kernel Internals, Addison-Wesley, 1996

[4] S. Goel and D. Duchamp. Linux Device Driver Emulation in Mach. In Proc. of the Annual USENIX 1996 Technical Conf., pages 65?73, San Diego, CA, Jan. 1996.

[5] Moti N. Thadani, Yousef A. Khalidi, An Efficient Zero-Copy I/O Framework for UNIX; SMLI TR-95-39 May 1995.

저자약력

채 병 수(Byoung Soo Chae)

2004년 강원대학교 전자계산학과 (이학사)
2006년 강원대학교 대학원 컴퓨터과학과 (이학석사)
2006년~현재 강원대학교 대학원 컴퓨터과학과 박사과정

<관심분야> System Programming

차 승 주(Seung Ju Tcha)

2005년 공군사관학교 전산통계학과 (이학사)
2008년 관동대학교 대학원 전자통신공학과 (공학석사)
2008-현재 관동대학교 대학원 전자통신공학과 박사과정

<관심분야> 영상신호처리시스템