메모리 복사를 최소화하는 효율적인 네트워크 시스템 호출 인터페이스

준회원 송 창 용*, 김 은 기**

An Efficient Network System Call Interface supporting minimum memory copy

Chang-Yong Song* *Associate Member*, Eun-Gi Kim**

요 약

본 논문에서는 파일 전송 시 발생하는 메모리 복사(memory copy)와 문맥 교환(context switch)을 최소화하여 시스템의 성능(performance)을 향상시킬 수 있는 네트워크 시스템 호출에 관한 연구를 수행하였다.

기존 파일 전송 기법에서 사용자가 하나의 패킷을 전송할 때, 사용자와 커널(Kernel) 공간 사이에서의 메모리 복사가 2회에 걸쳐 수행된다. 이러한 사용자와 커널 공간 사이에서 이루어지는 메모리 복사는 데이터 전송에 소요되는 시간을 증가시키고, 시스템의 성능에 좋지 않은 영향을 준다. 본 논문은 이러한 문제점들을 해결하기 위하여 필요한 경우 사용자와 커널 사이에서의 메모리 복사를 수행하지 않고, 데이터가 커널 공간 내에서 송수신될 수 있는 새로운 알고리즘을 제시하였다. 또한 실제의 시스템에서 제안된 알고리즘의 성능을 분석하기 위하여 리눅스 커널 버전 2.6.0의 소스 코드를 수정하였고, 새로운 네트워크 시스템 호출을 구현하였다.

성능 측정 결과, 본 연구에서 제안한 파일 전송 방식이 기존의 파일 전송 방식에 비하여 짧은 파일 전송 시간을 보여주었다.

Key Words : System Call; Memory Copy; Linux; File System; Network.

ABSTRACT

In this paper, we have designed and simulated a new file transmission method. This method restricts memory copy and context switching happened in traditional file transmission. This method shows an improved performance than traditional method in network environment.

When the UNIX/LINUX system that uses the existing file transfer technique transmits a packet to the remote system, a memory copy between the user and kernel space occurs over twice at least. Memory copy between the user and kernel space increase a file transmission time and the number of context switching. As a result, the existing file transfer technique has a problem of deteriorating the performance of file transmission.

We propose a new algorithm for solving these problems. It doesn't perform memory copy between the user and kernel space. Hence, the number of memory copy and context switching is limited to the minimum.

We have modified the network related source code of LINUX kernel 2.6.0 to analyzing the performance of proposed algorithm and implement new network system calls.

Ⅰ. 서 론

최근 인터넷이 대중화되면서 사람들에 의해 교환되는 데이터의 양은 점점 더 증가하고 있다. 이러한

* 한밭대학교 정보통신 전문대학원 네트워크 연구실(netking33@netian.com),
**한밭대학교 정보통신컴퓨터공학부 김은기(egkim@hanbat.ac.kr)
논문번호 : 030587-1231, 접수일자 : 2003년 12월 8일
※본 연구는 한밭대학교 정보통신 전문대학원 관리로 수행되었습니다.

상호간의 정보 교환은 주로 파일 전송을 통해서 이루어진다. 그러나 현재 파일 전송을 위해서 사용되는 대부분의 응용 프로그램들은 송수신되는 데이터의 일부분 또는 패킷(packet)의 헤더만을 처리하고, 전송되는 거의 모든 데이터들이 그대로 파일 시스템에 저장되거나 네트워크를 통해서 원격지 시스템으로 전송된다. 그 대표적인 예로서 FTP와 같은 응용 프로그램을 들 수 있다.[1][2]

기존 방식에서 파일이 전송되거나 수신될 때, FTP의 경우를 살펴보면, 파일의 데이터는 이용되지 않는다. 한번의 데이터 송신 또는 수신 시, 사용자 프로세스와 커널(kernel)간에 2번의 메모리 복사와 4번의 문맥 교환이 이루어지게 된다.

크기가 작은 파일을 송수신하거나 파일 전송이 많지 않은 경우에는 응용 프로세스와 커널 간의 메모리 복사와 문맥 교환이 문제점으로 여겨지지 않을 수 있다. 그러나 파일의 크기가 크고, 파일 전송이 빈번하게 이뤄지는 경우에는 잦은 메모리 복사와 문맥 교환의 발생이 시스템 성능을 저하시키는 문제점으로 대두될 수 있다.[3]

본 논문에서는 전송되는 파일의 크기가 크고, 파일 송수신이 빈번히 이루어지는 시스템에서 성능을 저하시키는 문제점인 사용자 프로세스와 커널 간 메모리 복사와 문맥 교환을 최소화하기 위해 데이터 전송이 커널 내부에서 수행되도록 하는 네트워크 파일 송수신 기법을 제안하였다.

제안된 네트워크 송수신 기법과 기존 네트워크 송수신 기법의 성능 비교를 위해 리눅스 커널(2.6.0 버전)을 수정하고 새로운 네트워크 시스템 호출을 구현하였다.

Ⅱ. 메모리 복사를 최소화한 시스템 호출 설계 및 구현

1. 효율적인 시스템 호출의 설계

기존 방식에서 하나의 파일 데이터 패킷이 전송되는 과정을 살펴보면, 먼저 응용 프로세스는 커널 공간의 하드 디스크에서 파일 데이터를 읽은 후, 읽은 데이터를 커널 공간으로 다시 복사하여 네트워크 계층을 통해 전송되도록 한다. 본 논문에서는 응용 프로세스와 커널 사이에서 이루어지던 파일 입출력과 관련된 모든 동작들을 커널 내부에서 이루어지도록 두 개의 시스템 호출을 제안하였다.

그림 1은 파일 데이터 전송을 수행하는 응용 프로그램의 의사 코드이다.[5]

```
until end of file {
    length = read(file pointer, buffer, bytes);
    write(socket pointer, buffer, length);
}
```

(a) 기존 파일 전송 기법

```
until end of file {
    length = directwrite(file pointer, socket pointer, length);
}
```

(b) 제안된 방식 I의 파일 전송 기법

```
length = directwrite(file pointer, socket pointer);
```

ⓒ 제안된 방식 II의 파일 전송 기법

그림 1. 응용 프로그램의 예

그림 1 (a)는기존 기법을 사용한 응용 프로그램의 의사 코드이다. read() 시스템 호출을 사용하여 파일 데이터를 파일 시스템으로부터 읽고, write() 시스템 호출을 사용하여 하나의 데이터 패킷을 네트워크로 전송한다. 그림 1 (b)는본 논문에서 제안하는 방식 I을 사용한 응용 프로그램의 의사 코드이다. directwrite() 시스템 호출을 사용하여 지정된 크기의 파일 데이터를 파일 시스템으로부터 읽은 후, 읽은 데이터를 하나의 데이터 패킷으로 구성하여 네트워크로 전송한다. 그림 1 (c)는 본 논문에서 제안하는 방식 II를 사용한 응용 프로그램의 의사 코드이다. 사용자는 한 번의 directwrite() 시스템 호출을 사용하여 모든 파일 데이터를 파일 시스템으로부터 읽고, 데이터 패킷을 네트워크로 전송하며, 이러한 동작은 모두 커널 내에서 이루어지게 된다.

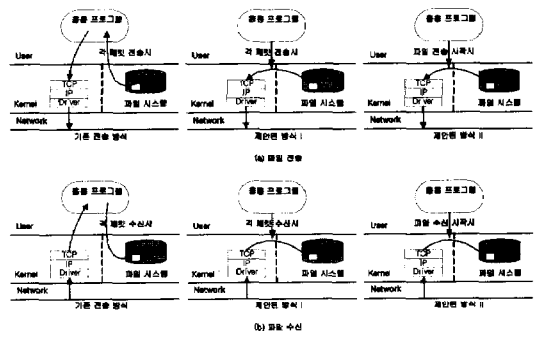


그림 2. 기존 방식과 제안된 방식에서의 파일 송수신 비교

그림 2는 기존 파일 전송 방식과 본 논문에서 제안된 파일 전송 방식의 파일 송수신 과정을 나타낸

것이다.[1] 그림 2의 (a)는 파일 데이터의 송신 동작 과정이다. 그림 2 (a)의 기존 전송 방식에서는 파일 데이터를 네트워크로 전송하기 위해 먼저 파일 데이터를 커널 내의 파일 시스템으로부터 사용자 공간으로 복사한다. 사용자 공간으로 복사된 파일 데이터는 다시 전송을 위해 커널 공간으로 복사된다. 하나의 데이터 패킷 전송시, 사용자와 커널 공간간 메모리 복사가 2회 발생되고, 사용자와 커널 모드의 변경으로 인한 문맥 교환은 4회 발생한다. 사용자 공간의 데이터를 네트워크로 전송한 이후 커널 모드에서 사용자 모드로 반환된다.

그림 2 (a)의 제안된 방식 I은 본 논문에서 추가된 시스템 호출을 이용한 파일 데이터 전송 과정이다. 제안 방식 I은 파일 데이터를 전송하기 위해 파일 시스템으로 직접 접근하여 커널 영역 내에서 복사를 수행하고, 복사된 데이터를 네트워크로 전송하게 된다. 하나의 데이터 패킷 전송시 (a)의 기존 전송 방식에 비해 메모리 복사와 문맥 교환이 50% 감소한다. 그림 2 (a)의 제안된 방식 II는 본 논문에서 추가된 시스템 호출을 이용한 또 다른 파일 데이터 전송 과정이다. 이 방식은 제안된 방식 I과 유사하나, 응용 프로그램에서 한 번의 시스템 호출을 수행함으로써 전체 파일이 전송될 수 있도록 하는 방식이다. 그림 2의 (b)는 파일 데이터의 수신 과정으로, 그림 2의 (a)와 거의 유사한 동작을 갖는다.

2. 추가된 시스템 호출의 구현

본 논문에서 제안된 알고리즘의 구현을 위해 2.6.0 버전의 리눅스 커널을 수정하였으며, 파일 전송에 사용되는 directwrite()와 수신된 파일을 저장할 때 사용되는 directread() 두 시스템 호출을 추가하였다.[4] 그림 3은 응용 프로세스가 본 논문에서 추가한 directwirte() 시스템 호출을 수행하였을 때, 커널 내에서 수행되는 동작을 보여주고 있다. sys_directwrite() 시스템 호출 핸들러는 direct_transmit 리스트의 비어있는 엔트리를 할당하고, 데이터 전송에 필요한 값(파일 기술자(file descriptor), 소켓 번호(socket number), 파일 크기, 파일 오프셋(file offset))들을 설정한다. 파일 전송을 위해 응용 프로세스가 요청한 전송 단위인 count와 파일 크기, 파일 오프셋을 비교하여 실제로 전달 가능한 데이터의 크기를 알아내고, 알아낸 크기와 동일한 크기의 sk_buff를 할당한다. 할당된 sk_buff에 전송 요청된 파일 데이터를 복사하여 하위 계층으로 전달한다. sk_buff를 전달 받은 하위 계층은 이를 네트워크를 통해 전달한다. 그림 4에서는 사용자의 directread() 시스템 호출에 따른 커널의 동작을 보여주고 있다.

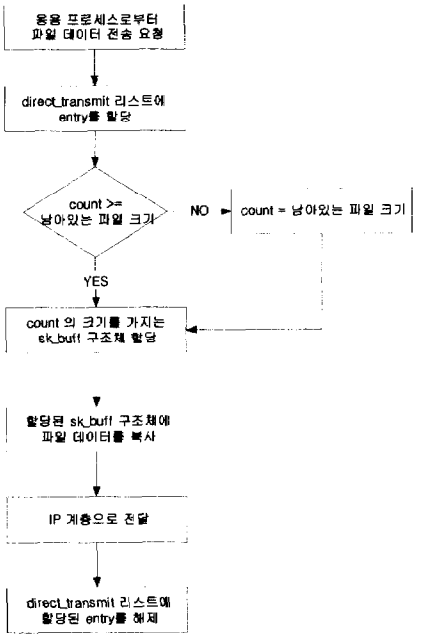


그림 3. directwrite() 시스템 호출의 동작

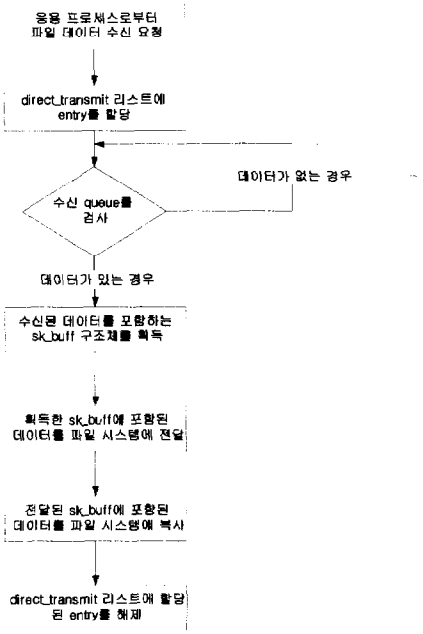


그림 4. directread() 시스템 호출의 동작

Ⅲ. 성능 분석

1. 측정 방식[1)]

본 논문에서 구현된 새로운 시스템 호출과 기존 시스템 호출을 이용한 데이터 전송 소요 시간을 측정하기 위해 그림 5와 같이 구성하였다. 성능 측정을 위해 방법으로 blocking 소켓 모드를 이용하였다. 본 연구의 측정을 위한 컴퓨터는 인텔 CPU P-III 650 MHz와 256 Mbyte의 RAM을 갖고 있으며, 리눅스 2.6.0 운영 체제로 동작한다.

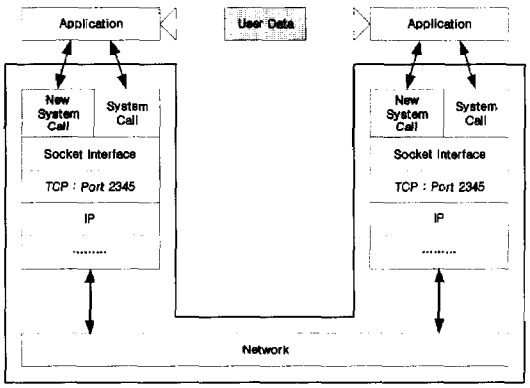


그림 5. 전송 시간 측정을 위한 구성

표 1. 항목 별 시스템 호출

항목	설 명	
	호스트 A	호스트 B
항목 1	read() 사용/ 데이터 수신	write() 사용/ 데이터 송신
항목 2	write() 사용/ 데이터 송신	read() 사용/ 데이터 수신
항목 3	directread() 사용/ 데이터 수신	directwrite() 사용/ 데이터 송신
항목 4	directwrite() 사용/ 데이터 송신	directread() 사용/ 데이터 수신

표 1은 성능 분석을 위한 측정 항목을 보여주고 있다. 항목 1과 항목 2는 기존 전송 기법에 사용되는 시스템 호출을 사용한 측정 항목이고, 항목 3과 항목 4는 본 논문에서 제안된 전송 기법을 적용하기 위해 추가된 시스템 호출을 사용한 측정 항목이다.

표 2는 성능 비교를 위한 측정 방식 별 세부적인 측정 환경을 나타낸다.

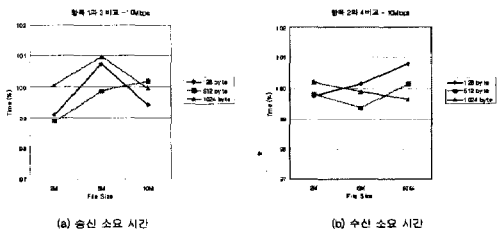
표 2. 성능 분석을 위한 측정 환경

측정 방식	측정 항목	파일 크기 (Mbyte)	대역폭 (Mbps)	패킷 크기 (byte)	항목당 측정 횟수
기존 전송 방식	항목 1 항목 2	2, 5, 10	10, 100, 직접 연결	128, 512, 1024	50 회
제안된 방식 I	항목 3 항목 4	2, 5, 10	10, 100, 직접 연결	128, 512, 1024	50 회
제안된 방식 II	항목 3 항목 4	2, 5, 10	10, 100, 직접 연결	128, 512, 1024	50 회

표 2의 대역폭에서 직접 연결은 UTP 선을 이용하여 두 개의 컴퓨터를 직접 연결한 것을 의미한다.

2. 파일 송수신 시간

2.1 기존 전송 방식과 제안된 방식 I의 파일 송수신 시간



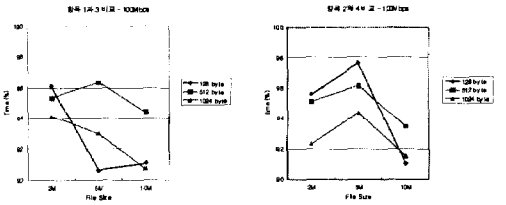
(a) 송신 소요 시간 (b) 수신 소요 시간

그림 6. 10Mbps의 대역폭에서 기존 전송 방식과 제안 방식 I의 파일 송수신 시간

그림 6에서는 10 Mbps의 대역폭을 갖는 네트워크 환경에서 기존 전송 방식과 제안 방식 I의 파일 송수신 소요 시간의 비교 값을 보여주고 있다. 그림

1) 성능 측정을 위한 방식으로는 blocking 소켓 모드와 non blocking 소켓 모드 두 가지 방식이 있다. blocking과 non-blocking 소켓 모드를 모두 테스트한 결과 성능면에서 보면 약간의 차이를 보였으나 기존 전송 방식과 비교하여 향상된 성능을 보였다. 본 논문에서는 두 방식 중 한가지인 blocking 소켓 모드에 대해 성능을 측정하였으며, 그 결과만을 사용하였다.

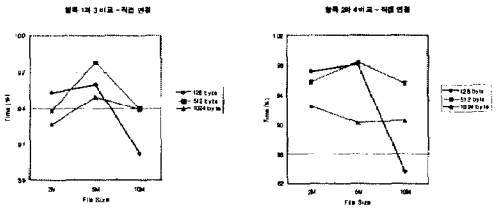
에서는 기존 전송 방식의 소요 시간을 100%로 하여 제안 방식 I의 상대적인 소요 시간을 백분율로 표현하였다. 그림 6을 살펴보면, 10 Mbps의 대역폭을 갖는 네트워크 환경에서는 네트워크의 처리 속도가 시스템의 속도에 미치지 못하여 기존 방식과 제안된 방식 I과의 송수신 시간차가 크지 않은 것을 알 수 있다.



(a) 송신 소요 시간 (b) 수신 소요 시간

그림 7. 100Mpbs의 대역폭에서 기존 파일 전송 방식과 제안 방식 I의 파일 송수신 시간

그림 7에서는 100 Mbps의 대역폭을 갖는 네트워크 환경에서 기존 전송 방식과 제안 방식 I의 파일 송수신 소요 시간의 비교 값을 보여주고 있다. 그림 7은 그림 6과는 달리 네트워크의 대역폭이 증가함에 따라서 파일 송수신 소요 시간이 개선된 결과를 보여주고 있다. 그림에서 제안 방식 I은 기존 전송 방식과 비교하여 파일 송수신 시간이 최대 10% 정도 개선되었다.



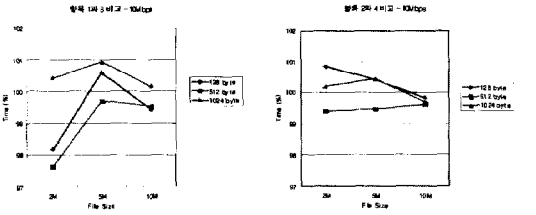
(a) 송신 소요 시간 (b) 수신 소요 시간

그림 8. 직접 연결된 환경에서 기존 파일 전송 방식과 제안 방식 I의 파일 송수신 시간

그림 8은 UTP 케이블을 이용하여 두 대의 컴퓨터를 직접 연결한 네트워크 환경에서 기존 전송 방식과 제안 방식 I의 파일 송수신 소요 시간 비교 값을 보여주고 있다. 제안 방식 I은 기존 전송 방식과 비교하여 송수신 소요 시간이 최대 15% 정도 개선되었다.

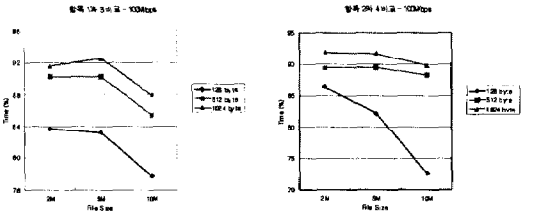
2.2 기존 전송 방식과 제안된 방식 II의 파일 송수신 시간

그림 9에서는 10 Mbps의 대역폭을 갖는 네트워크 환경에서 기존 전송 방식과 제안 방식 II의 파일 송수신 소요 시간의 비교 값을 보여주고 있다. 그림 9은 네트워크를 통해서 전송되는 패킷의 크기에 관계없이 기존 전송 방식과 제안 방식 II에서의 전송 시간차가 크지 않은 결과를 나타낸다. 이것은 그림 6과 유사한 이유로써 위와 같은 결과가 나타난 것을 볼 수 있다.



(a) 송신 소요 시간 (b) 수신 소요 시간

그림 9. 10Mbps의 대역폭에서 기존 파일 전송 방식과 제안 방식 II의 파일 송수신 시간

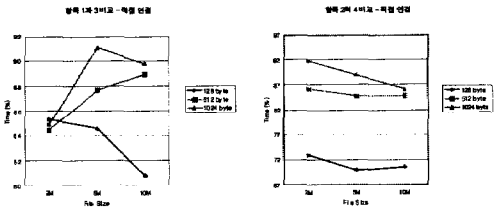


(a) 송신 소요 시간 (b) 수신 소요 시간

그림 10. 100Mbps의 대역폭에서 기존 파일 전송 방식과 제안 방식 II의 파일 송수신 시간

그림 10에서는 100 Mbps의 대역폭을 갖는 네트워크 환경에서 기존 전송 방식과 제안 방식 II의 파일 송수신 소요 시간의 비교 값을 보여주고 있다. 그림 10은 그림 9와는 달리 네트워크의 대역폭이 증가함에 따라서 파일 송수신 소요 시간이 개선된 결과를 보여주고 있다. 제안 방식 II는 기존 전송 방식과 비교하여 파일 송수신 시간이 최대 25% 정도 개선되었다.

그림 11은 UTP 케이블을 이용하여 두 대의 컴퓨터를 직접 연결한 네트워크 환경에서 기존 전송 방식과 제안 방식 II의 파일 송수신 소요 시간 비교 값을 보여주고 있다. 제안 방식 II는 기존 전송 방식과 비교하여 파일 송수신 시간이 최대 30% 정도 개선되었다.



(a) 송신 소요 시간 (b) 수신 소요 시간

그림 11. 직접 연결된 환경에서 기존 파일 전송 방식과 제안 방식 II의 파일 송수신 시간

Ⅳ. 결 론

네트워크 환경에서 기존의 파일 송수신 기법은 사용자와 커널 메모리 영역간의 불필요한 메모리 복사와 문맥 교환을 발생시키는 문제를 갖는다.

본 연구에서는 이러한 문제점을 해결할 수 있는 새로운 알고리즘을 제안하고, 리눅스 커널 2.6.0을 수정하여 새로운 시스템 호출을 구현하였다. 또한, 기존의 파일 전송 기법과 새롭게 제안된 알고리즘의 성능을 비교하기 위하여, 응용 프로세스에서 동일한 파일을 각기 다른 방식으로 전송하여 전송 시간을 측정하였다. 측정 결과, 고속의 네트워크 환경에서는 기존 방식과 제안된 방식의 성능에 있어서 뚜렷한 차이를 볼 수 있었다.

본 연구에서 제안된 새로운 파일 송수신 기법은 불필요한 메모리 복사와 문맥 교환을 최소화 할 수 있도록 함으로써 하나의 프로세스가 파일을 빠르게 전송할 수 있도록 지원할 뿐만 아니라, 파일 전송 서버와 같이 많은 부하를 갖는 시스템의 부하를 줄여 줄 수 있는 장점을 갖는다. 또한, 기존 파일 전송 응용 프로그램의 간단한 수정만으로 본 연구에서 제안된 방식을 적용할 수 있기 때문에 실제 적용에 무리가 없을 것으로 생각된다.

향후, 시스템의 성능 향상을 위한 파일 전송 기법과 네트워크의 성능 향상에 대한 추가적인 연구가 필요할 것으로 생각된다. 또한, 응용 프로세스를 이용하지 않고 로컬 시스템의 파일을 네트워크에 연결된 다른 시스템과 공유할 수 있는 파일 분산 저장 장치 등으로의 적용을 위한 추가적인 연구가 필요하다.[6]

참 고 문 헌

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

[2] 안순신, 김은기, 정보통신 네트워크, 이한 출판사, 1998, pp 174~180

[3] D. D. Clark et al., "An Analysis of TCP Processing Overhead", IEEE Com. Mag., vol. 27, no. 6, Jun 1989

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

[5] W. Richard Stevens, UNIX Network Programming, Volume 1 Networking APIs, Second Edition, PrenticeHall, 1999

[6] Compaq Computer Corp., "Virtual Interface Architecture Specification Draft Revision 1.0", http://www.viarch.org/, Dec. 1997

송 창 용(Chang-Yong Song) 준회원

2002년 2월 : 한밭대학교 정보통신 공학과 졸업

2004년 2월 : 한밭대학교 정보통신 전문대학원 정보통신 공학과 석사

<관심분야> 인터넷프로토콜, 무선랜, WAP, 보안 프로토콜

김 은 기(Eun-Gi Kim)

1987년 2월 : 고려대학교 전자공학과 (공학사)

1989년 2월 : 고려대학교 전자공학과 (공학석사)

1994년 2월 : 고려대학교 전자공학과 (공학박사)

1995년 현재 국립 한밭대학교 정보통신·컴퓨터공학부 부교수

<관심분야> 인터넷프로토콜, 무선랜, 보안 프로토콜, 라우팅 프로토콜 등