듀얼코어 임베디드 리눅스 시스템에서 공유 메모리 성능 개선 방안 및 성능 분석

Improvement Method and Performance Analysis of Shared Memory in Dual Core Embedded Linux system

정 지 성* 김 창 봉**
Jung, Jisung Kim, Changbong

요 약

최근 복잡한 프로그래밍 환경에서 다수의 프로세스들은 서로 협력하기 위하여 서로 통신하고 자원과 정보를 공유한다. 커널에서는 이것이 가능한 방법으로 프로세스간 통신이라는 IPC(Inter-Process Communication)를 제공한다. 리눅스에서 사용되는 공유 메모리는 동일한 메모리 영역에 여러개의 프로세스가 접근할 수 있도록 해 주는 기술이다. 본 논문에서는 서로 다른 코어에 서로 다른 운영체제를 갖는 듀얼코어 임베디드 리눅스 시스템에서 공유 메모리 성능 개선 방안을 제시하고, MP2530F(ARM926F+ARM946E)의 임베디드 리눅스 시스템을 구축하여 성능을 측정한다. 공유 메모리를 이용한 프로세스의 동작이 별개의 CPU에서 동작되도록 함으로써 성능 향상을 꾀한다.

ABSTRACT

Recently multiple process communicate together. They share resource and information for cooperation in complicated programming environment. Kernel provides IPC (Inter -Process Communication) for communication with each other process. Shared Memory is a technique that many processes can access to identical memory area in the Linux environment. In this paper, we propose a performance improvement method of shared memory in the dual-core embedded linux system which is consist of different core and different operating system. We construct the MPC2530F (ARM926F+ARM946E) linux system and measure the performance therein. We attempt a performance enhancement in each CPU for each process which uses a shared memory.

☞ KeyWords : Dual-Core, Shared Memory, Embedded Linux, 듀얼코어, 공유 메모리, 임베디드 리눅스

1. 서 론

컴퓨팅 기술은 끊임없이 발전해 나가고 있다. 이러한 컴퓨팅 기술의 중심에 컴퓨팅 성능이 있으며, 이는 CPU로 대변된다. 최근 임베디드 시스템은 고성능화로 인해 단순한 주변기기를 벗어나 다양한 사용자가 원하는 복잡한 기능을 충분히 소화해낼 만큼의 뛰어난 능력을 필요로 하고 있다. 듀얼코어는 하나의 CPU 안에 여러 개의 독립적인 실행코어를 둠으로써 프로그램을 병렬적으로 처리할 수 있게 한 CPU이며, 병렬처리 방식으로는 pipelining, multiprocessor, multithreading 등의 방법이 있다.[1] 코어의 모델에는 두 개 이상의 general execution core를 갖고 있는 형태와 하나 이상의 general execution core와 장치특성에 맞는 dedicated DSP들을 갖고 있는 형태가 있다.[2] 듀얼코어는 단일코어 시스템에서의 멀티 테스킹 문제 즉, CPU 사용 병목 현상을 해결할 수 있다.[3] 듀얼코어 아키텍처를 보면 코어들이 공유 메모리를 두고 경쟁해야 하기 때문에 현재의 듀얼코어

* 정 회 원 : 한국전자통신연구원 연구원
jungjs@etri.re.kr(주저자)
** 정 회 원 : 공주대학교 정보통신공학부 교수
aggie@kongju.ac.kr(교신저자)
[2010/01/29 투고 - 2010/03/05 심사(2010/05/24 2차 - 2010/06/23 3차) - 2010/07/02 심사완료]

칩에는 잠재적인 결점이 존재한다. 따라서 각각 메모리에서 데이터를 더욱 오래 기다리게 되며, 이에 따라 병목현상이 발생하고 성능이 저하될 가능성이 있다. 듀얼코어 방식에서는 여러 개의 코어로 프로그램을 분산시켜 병렬적으로 동작하게 해야 하기 때문에 프로그램 혹은 디버깅이 매우 어려워지고 기존의 많은 응용 프로그램들이 듀얼코어 방식에 최적화되어있지 않다. 이에 듀얼코어에서는 잘 만들어진 멀티쓰레드와 주어진 문제들을 여러 쓰레드로 분리하여 수행하는 알고리즘과 데이터 구조가 요구된다. 사용자의 요구가 증대되면서 프로그램이 갖춰야 할 기능 또한 많아지고 당연히 프로그램 크기도 비대해졌다. 또한 하나의 프로그램만으로는 사용자의 욕구를 충족시키지 못해 여러개의 프로그램을 만들어야 하는 상황이 발생하고 있다.[4] 복잡한 프로그래밍 환경에서 다수의 프로세스들은 서로 협력하기 위하여 서로 통신하고 자원과 정보를 공유한다. 커널(Kernel)에서는 이것이 가능한 방법을 제공하는데, 이를 프로세스간 통신(Inter-Process Communication) 또는 IPC라 부른다.[5] IPC는 프로그램들간의 데이터를 공유하고 동기화하기 위해 사용되는 방법을 뜻한다. 이때 프로그램은 보통 프로세스를 의미한다. IPC를 할 수 있게 해주는 일반적인 도구는 세마포어, 공유 메모리, 내부 메시지큐 등이 있다.[6] 이런 방법이 갖는 이점은 디스크를 통한 파일공유와 관련 IO를 통한 공유하는 방법의 오버헤드를 줄일 수 있다는 것이다. 프로세스간 통신을 하는 목적은 데이터 전송, 데이터 공유, 사건 전송, 자원 공유, 프로세스 제어를 하기 위함이다. 이를 위하여 전통적인 Unix에서는 시그널(Signal), 파이프(Pipe), 프로세스 추적(Process Tracing)의 기능을 사용하였다.[7] 그러나 많은 사용자 프로그램들에서는 이것만으로 IPC를 충족할 수가 없었으며, 이것은 SystemV 릴리즈 버전에서 메시지큐, 세마포어, 공유 메모리 기법을 사용하게 된다.[8] 공유 메모리 구현에 있어서는 시스템의 가상 메모리(Virtual memory) 구조에 크게 의존하며, 데이터의 복사나 시스템 요청을 사용하지 않고 많은 양의 데이터를 공유하는 매우 빠르고 융통성 있는 기법을 제공한다.[9] 그러나 이 기법은 동기화 기법을 제공하지 않는다는 단점이 있다. 예를 들면, 두 개의 프로세스가 같은 공유 메모리 영역의 내용을 변경하려고 할 경우, 커널은 이를 순차적으로 처리를 해야 하나, 그렇게 하지 않으면 쓰여진 데이터는 사용할 수 없게 된다. 그래서 공유 메모리를 사용하는 프로세스들은 프로세스들 간에 동기화를 하는 프로토콜을 고안해야 하며, 이것은 공유 메모리 성능에 영향을 주는 요소로 작용할 수 있다.

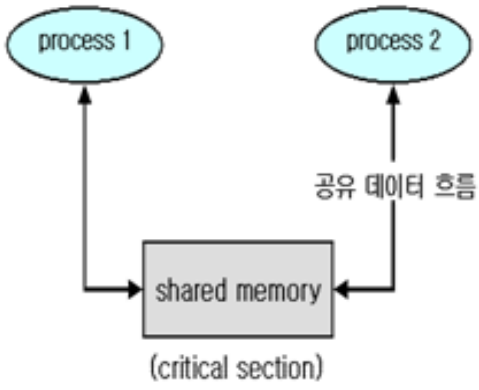
본 논문의 구성은 다음과 같다. 2장에서 리눅스에서 사용되는 공유 메모리 구조와 임베디드 리눅스를 기반으로 하는 듀얼코어 프로세서에서의 IPC 구조에 대하여 알아본다. 3장에서는 듀얼코어 임베디드 리눅스 시스템에서 공유 메모리(Shared Memory)의 성능 개선 방안에 대해 설명한다, 4장에서는 임베디드 리눅스 시스템 기반의 듀얼코어 공유 메모리 부분의 성능 향상을 위한 기능 설계에 대해 설명한다. 5장에서는 듀얼코어 임베디드 리눅스 시스템 테스트베드 환경을 구축하고 공유 메모리의 성능 개선에 따른 성능 향상을 측정하며, 메모리 크기 변화에 따른 IPC 속도를 측정 결과를 보여준다. 그리고 6장에서는 본 논문에 대한 결론과 향후 연구 방향에 대해 설명한다.

2. 관련 연구

2.1 공유 메모리 구조

리눅스에서 사용되는 공유 메모리(shared memory)는 프로세스가 공유 메모리 세그먼트를 통해 공통 데이터 구조 및 데이터에 접근할 수 있게 한다. 공유 메모리는 커널 작업을 수반하지 않으며 프로세스 간의 데이터 복제 작업이 불필요하기 때문에, IPC(Inter-Process Communication)를 위한 빠른 기술이다.[10] 공유 메모리는 유닉스 System V (1983)에서 처음 등장한 세가지 종류의 프로세

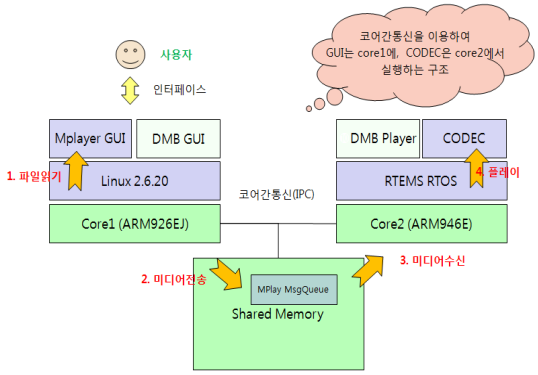
스간 통신 방법 중 하나로 이들이 제공하는 세가지는 메시지 큐(message queue)와 세마포어(semaphore), 그리고 공유 메모리(shared memory)이다.[11] 듀얼코어 IPC는 서로 다른 코어 및 서로 다른 운영체제를 사용하며 공유 메모리를 통해 메시지 공유를 한다. 공유 메모리(shared memory)는 물리적인 메모리의 특정 영역에 대하여 프로세스들에 의해 공유되어지는 영역(region)이라 할 수 있다. 프로세스들은 이 공유 메모리 지역을 자신의 가상 메모리 영역에 부착(attach)하여 사용한다.[12] 영역은 데이터 읽기/쓰기의 시스템 호출을 사용하지 않고 다른 방법으로 접근을 하기 때문에 프로세스간 데이터를 공유하는 기법 중 가장 빠르다고 할 수 있다.[8] 예를 들어, 하나의 프로세스가 공유 메모리의 영역에 기록(write)를 하게 된다면, 이 내용은 이 영역을 공유하는 모든 프로세스들에 의해 바로 보여지게 되어 아주 빠르게 데이터를 공유하게 된다.[13]



(그림 1) 공유 메모리 구조

2.2. 듀얼코어 프로세서의 IPC 구조

(그림 2)는 임베디드 리눅스를 기반으로 하는 MP32530F의 듀얼코어 프로세서에서 IPC 구조를 나타낸다. 서로 다른 프로세서 및 다른 운영체제를 사용하는 IPC 메커니즘 중 공유 메모리 기법으로 공유 메모리를 통해 서로 다른 데이터를 공유한다. (그림 2)와 같이 서로 다른 프로세서를 사용할 경우에 각각의 프로세서에서 공유 메모리를 통하여 데이터를 공유하거나 정보를 주고받을 수 있다. 또한 서로 다른 프로세서에서 동시에 다른 프로세스가 수행될 수 있다. 프로세서간 job의 분배가 가능하나 정보는 하나의 메모리를 통해 공유가 가능하다. 문제는 OS가 서로 다르기 때문에 서로 다른 job 스케줄러가 존재하며, 이들 사이의 메시지 통신 및 동기화 등의 문제가 존재한다.[14]



(그림 2) 듀얼코어 프로세서의 IPC 구조

(그림 3)은 듀얼코어 프로세서의 메모리 주소 인식에 대한 그림으로 프로세서 간 메모리 공유 구조에서 두 프로세서는 임의의 메모리 영역에 접근이 가능하다. 그러나 두 프로세서가 메모리의 주소를 다르게 인식한다. ARM946 프로세서는 PA_ARM940_BASE에 정의된 base address를 0 번지로 인식하며, <linux/include/asm-arm/arch-mmsp2/hardware.h> 에 ARM940 프로세서의 메모리 영역과 공유 메모리 영역에 정의된다.[15]]

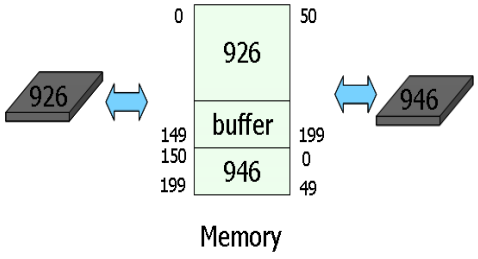
```
#define PA_ARM940_BASE      0x83000000
#define VA_ARM940_BASE      0xf4000000      // 0x03000000
#define ARM940_MEM_SIZE     (16*1024*1024)      // 16M

#define PA_BUFFER_BASE      0x82000000
#define VA_BUFFER_BASE      0xf7000000
#define BUFFER_MEM_SIZE     (16*1024*1024)
```

(그림 3) 공유 메모리 영역 정의

(그림 4)는 리눅스 커널에서 메모리 할당에 대해 정의하는 부분으로 <linux/arch/arm/mach- mmsp2/mdk.c>에서 메모리 할당을 위한 mdk_io_desc 자료구조를 정의하고 iotable_init 통해 할당된다. MP32530F에 포팅된 리눅스 커널 2.4는 ARM940 공유 메모리 영역의 크기 및 위치를 컴파일 타임에 결정을 한다.[16]

```
static struct map_desc mdk_io_desc[] __initdata = {
   /* virtual          physical          length          domain     r  w  c  b */
   {MDK_ETH_VIO_BASE, MDK_ETH_PIO_BASE, 0x10000,        DOMAIN_IO, 0, 1, 0, 0},
   {VA_ARM940_BASE,   PA_ARM940_BASE,   ARM940_MEM_SIZE, DOMAIN_IO, 0, 1, 0, 0},
   {VA_BUFFER_BASE,   PA_BUFFER_BASE,   BUFFER_MEM_SIZE, DOMAIN_IO, 0, 1, 0, 0},
   LAST_DESC
};
static void __init mdk_map_io(void){
    iotable_init(mdk_io_desc);
    ...
}
```

(그림 4) 메모리 할당을 위한 자료구조 정의

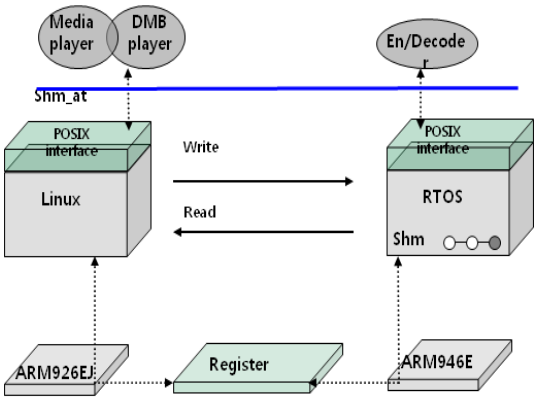


(그림 5) 듀얼코어 프로세서의 메모리 주소 인식

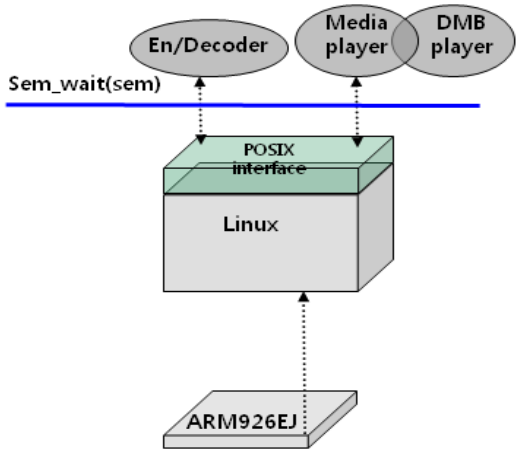
프로세스들의 수행양이 많을 경우 단일 프로세서에서는 프로세스 수행에 시간이 많이 소용된다. 이는 단일 프로세서는 프로세스들을 순차적으로 처리하기 때문이다. 한 프로세스가 CPU 사용의 최대 시간(time slice)을 넘을 정도로 긴 수행을 해야 하는 경우라 가정한다. 단일 프로세서는 CPU 사용의 최대 시간을 사용한 프로세스1에 대해서 CPU 사용을 중지 시키고 프로세스1은 Sleep 상태로 들어서고, 다음 사용자인 프로세스2가 CPU를 할당 받는다. 이럴 경우 프로세스 수행 시간의 거의 2배 가까운 시간이 소요되어서야 프로세스의 수행이 종료 될 수 있다. 하지만 두 개의 CPU를 사용하게 된다면, 혹은 두 개의 CPU를 사용하는 것과 같은 듀얼 코어 프로세서를 사용하게 된다면 두 개의 프로세스를 동시에 수행할 수 있어 성능 향상에 도움이 된다.[17]

3. CPU를 활용한 공유 메모리 성능 향상 방안

듀얼코어 임베디드 리눅스 공유 메모리 성능 향상을 위해서는 다중 처리기를 이용하는 성능 향상, 스케줄링 알고리즘 성능 개선을 통한 성능 향상, 그리고 TLB(Translation Lookaside Buffer)를 통한 성능 개선을 통한 성능 향상 방안들이 있다. 대부분의 기존 연구들은 다중 처리기를 이용한 성능 향상의 방법을 통한 공유 메모리 성능 향상을 도모하고 있다. 다중 처리기를 이용한 성능 향상의 방법을 통한 성능 향상의 경우는 CPU 개수를 늘려서 공유 메모리 성능을 향상시키는 방법이다. 먼저, 성능 향상 정도를 보이기 위하여 하나의 CPU를 동작시킨 상태에서 공유메모리 성능을 측정한다. 그 다음으로 두 개의 코어를 동작시킨 상태에서 공유메모리 성능을 측정한다. 측정한 성능 결과와 하나의 CPU를 동작시킨 상태에서 성능 측정 결과를 비교한다. 비교한 결과를 통해서 성능 향상 정도를 파악할 수 있다. (그림 6)은 CPU를 활용한 공유 메모리 성능 향상 방안에 대한 구조를 보여준다. 스케줄링 알고리즘 성능 개선을 통한 방법의 경우는 CPU 등 시스템 환경에 변화를 주지 않고, 커널 내에서 CPU 스케줄러 부분에서 성능 향상을 꾀하는 방법이다. 스케줄러에서 공유 메모리 성능 향상을 위하여 효과적으로 CPU를 관리할 수 있도록 지원하는 기능이다. TLB(Translation Lookaside Buffer) 방법의 경우는 커널 내의 메모리 처리 부분에서 보다 빠르게 메모리 접근 및 사용이 가능하도록 하여 성능 향상을 이루기 위한 방법이다. 공유 메모리 성능 향상 방안이 수립되면, 실제 성능 향상 정도를 점검해야 한다.



(a) 두개의 CPU를 이용한 공유 메모리 동작 과정

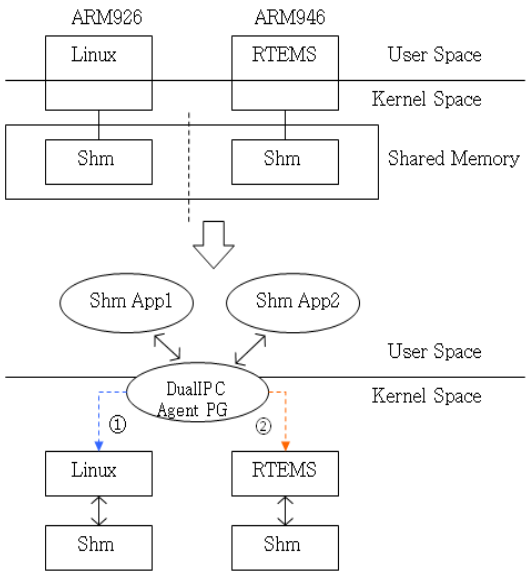


(b) 하나의 CPU를 이용한 공유 메모리 동작 과정

(그림 6) CPU를 활용한 공유 메모리 성능 향상 방안

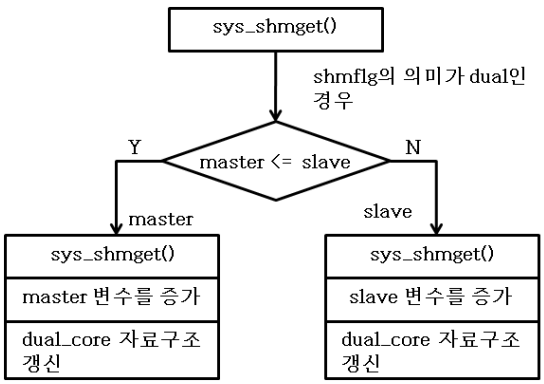
4. 공유 메모리 성능 향상을 위한 기능 설계

이 장에서는 임베디드 리눅스 시스템 기반의 듀얼코어 공유 메모리 부분의 성능 향상을 위한 기능 설계이다. (그림 7)은 듀얼코어에서 서로 다른 운영체제인 ARM 926은 Linux 운영체제를 ARM 946 코어는 RTEMS 운영체제를 사용할 경우 공유 메모리를 구현하기 위한 구성도이다. 운영체제가 서로 다른 경우 사용하는 공유메모리의 구조가 다를 수 있어 이러한 문제를 보완하고자 Agent Program을 도입하였다. Agent Program은 (그림 7)의 ①, ②와 같이 서로 다른 운영체제에서 생성한 공유 메모리로의 링크를 행해준다. (그림 7)의 ①은 리눅스 운영체제에서 공유 메모리를 생성하였다. 리눅스 운영체제의 관리 아래에서 동작하고 있는 공유 메모리를 RTEMS 운영체제에서 필요하게 될 경우 Agent Program에서 (그림 7)의 ①과 같은 링크를 생성하여 리눅스 상의 공유 메모리를 사용할 수 있게 해준다. (그림 7)의 ②는 (그림 7)의 ①과 반대로 RTEMS 운영체제에서 공유 메모리에 리눅스 시스템에서 동작하는 프로그램이 접근하고자 할 경우 (그림 7)의 ②와 같은 링크가 생성되어 RTEMS 운영체제 상의 공유 메모리를 사용할 수 있게 해준다. 설계 방법은 두 개의 코어 중에서 마스터와 슬레이브를 지정한다. 공유 메모리의 기능 구현은 agent를 통해서 이루어진다. agent내 마스터 노드의 기능을 이용해서 슬레이브의 기능을 이용한다. 마스터와 슬레이브 간에 데이터 전송방식은 RPC 메커니즘을 이용한다. 듀얼코어의 서로 다른 CPU에서 독립적으로 갖고 있는 메인 메모리에서 할당되는 공유 메모리 정보를 저장하기 위해 새로운 자료구조를 만들었다.



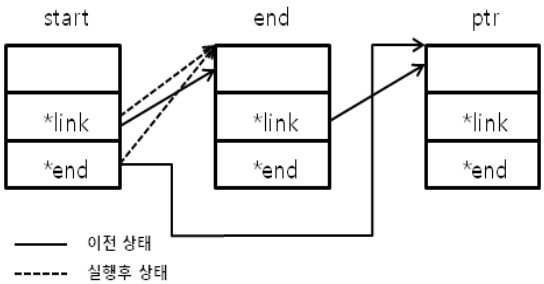
(그림 7) 듀얼코어에서 공유 메모리 설계 구성도

(그림 8)은 공유 메모리 정보를 저장하기 위한 자료 구조와 그 자료 구조를 활용하여 듀얼 CPU에서 shmget()을 사용할 수 있게 만든 sys_shmget( ) 함수의 구조도이다. 본 논문에서 설계 및 구현된 함수이다.



(그림 8) sys_shmget() 함수 구조

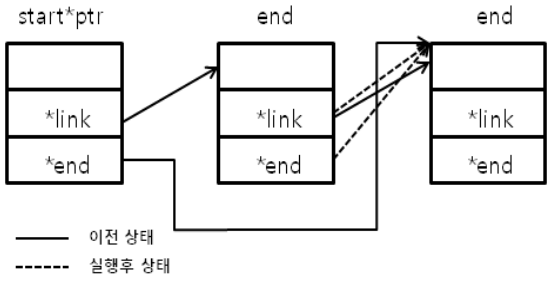
이 함수에서 shmflg 변수 값으로 듀얼 코어에서 동작할 응용프로그램인지를 구분가능 하도록 되어있다. 만약 shmflg의 의미가 듀얼 코어 응용프로그램이라고 한다면, master(Linux) 노드인지 slave(RTEMS) 노드인지를 구분한다. master 노드라면 원래 Linux의 sys_shmget( ) 함수 코드가 실행이 되고, slave 노드라면 RTEMS의 shmget( )에 해당하는 함수가 수행하게 된다.



(그림 9) shmdt()에서 자료구조의 갱신 과정

(그림 9)는 (그림 8)의 "dual_core 자료구조 갱신" 부분에서의 동작과정을 보여주고 있다. 실선으로 표시된 부분이 기존에 유지되고 있던 자료구조가 새로운 정보가 입력이 되면서 자동적으로 점선으로 표시된 부분처럼 변경됨을 보여주고 있다. 'dual_core'자료 구조의 link변수는 항상 다음 순서에 해당하는 변수의 주소를 가리킨다. 'start'의 end 변수는 항상 마지막 순서에 해당하는 변수의 주소를 가리킨다. 처음 'start'와 'end' 두 개의 변수가 바로 다음 'dual_core' 자료 구조이며, 마지막 'end'를 가리키고 있다. 하지만 새로운 변수인 'ptr'이 입력이 되면서 마지막이었던 변수의 'link' 부분이 'ptr'을 가리키게 되고, 'start'의 'end' 변수가 가리키던 'end'의 주소가 마지막에 추가된 'ptr'을 가리키게 된다.

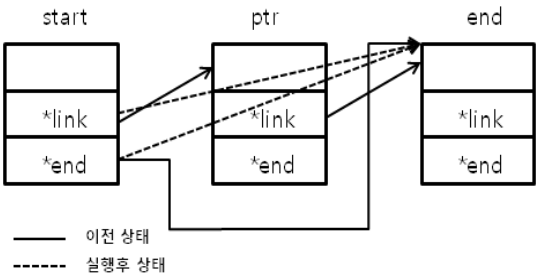
shmdt() 함수는 shmget()과 달리 'ptr'의 위치가 임의적이다. shmget()함수의 경우 항상 맨 마지막이 'ptr'이기 때문에 한 가지 경우에 대한 동작 과정만 있으면 되지만, shmdt() 함수의 경우는 'ptr'의 위치가 맨 마지막인 경우와 맨 처음인 경우, 그리고 중간인 경우 이렇게 3가지의 경우로 나눌 수 있다. 그 첫 번째로 맨 마지막의 경우에서 가리키던 'end'의 주소를 'ptr'바로 앞 자료구조의 주소로 변경해주어야 한다. 그 이외의 부분은 자료 주조들의 연결에 영향을 주지 않는다.



(그림 10) shmdt()에서 자료구조의 갱신 동작 과정-A

(그림 10)에서 'ptr'이 맨 처음일 때, 'start'의 주소 값을 'start' 다음 자료 구조의 주소 값으로 바꾸어 주어야한다. 이는 'start'값을 이용하여 자료구조를 따라가면서 'ptr'을 찾는 for loop 문에서 사용하는 중요한 지표이기 때문에 제일 먼저

'start' 값을 수정해주어야 한다. 그리고 'start'에서 가리키고 있는 맨 마지막 자료 구조의 주소를 'start' 다음 자료 구조의 'end'의 값으로 넣어주어야 한다.



(그림 11) shmdt()에서 자료구조의 갱신 동작 과정-B

(그림 11)의 shmdt()에서 세 번째에 해당하는 'ptr'이 중간인 경우 'ptr'에 해당하는 자료 구조의 바로 앞 자료 구조를 수정해 주어야 한다. 먼저 'ptr'의 바로 앞 자료 구조를 찾고, 두 번째로 'ptr' 바로 앞 자료 구조의 'link'값을 'ptr'자료 구조의 'link'값으로 변경해야 한다.

리눅스 운영체제의 공유 메모리 성능 향상 방안으로 커널 영역에서 사용자 영역으로 복제가 일어나는 부분을 볼 수 있다. 이 부분에 대한 성능 개선 방안으로 이 부분은 shm_at()를 수행하게 되면 할당된 공유 메모리 주소가 parent와 child 프로세스에게 전달되기 때문에 데이터를 읽는 부분에서의 성능 개선 여지가 남아있다. 프로세스 간에 메모리 참조하는 경우에 최대한 접근 횟수를 줄이는 방안으로 shm_at()-> do_mmap()을 수행하는 것과 do_mmap()에서 공유 메모리 영역에 데이터를 기록할 경우, 다른 프로세스에서 이 데이터를 읽는 경우에 다른 프로세스가 공유 메모리에 접근 시에 page_fault() 인터럽트가 발생하게 된다. 그리고 Page_fault()에서 성능 개선을 볼 수 있다. 종합적으로 공유 메모리 영역으로 설정된 부분에서 데이터를 읽고 쓰는 부분과 관련한 곳에서 성능 개선 방안 정립이 필요하다.

5. 실험 및 분석

5.1 실험 환경

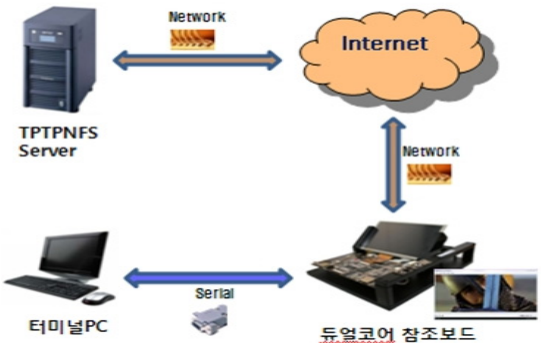
듀얼코어 임베디드 리눅스 시스템 환경을 구축하고 듀얼코어 지원 커널 기반 동영상을 재생하여 다중 환경에서 오디오-비디오(A-V) 프로세스 동기화 처리성능 측정과 메모리 크기에 따른 공유 메모리의 IPC 속도를 측정하였다. 공유 메모리 성능측정을 위해 공유 메모리 소스에 시간을 측정할 수 있는 기능을 추가하였고, 듀얼코어에서 커널의 통신 속도 향상에 따른 오디오-비디오(A-V) 프로세스 동기화 처리성능 비교를 위해서는 (표 1)과 같이 시스템 환경을 구축하였다. (표 1)은 성능 측정을 위한 듀얼코어 임베디드 리눅스 시스템 환경으로 비대칭 듀얼코어(ARM926EJ/ARM946E)용 커널 기술이 적용되어 마스터 코어에는 리눅스 2.6.20.커널이 지원되고 슬레이브 코어에는 알템즈 4.7.1 RTOS가 지원되며, Dual-IPC Device Driver 모듈이 지원된다. 또한 비대칭 멀티프로세싱 환경에서 코어간 고속통신 기술인 POSIX 표준 API 지원과 듀얼코어간 세마포어, 메시지큐, 공유 메모리가 적용되어 있다. 듀얼코어 임베디드 응용개발로 ToolChain, 부트로더 및 RootFS 기반의 기본 개발 환경이 지원된다.

(표 1) 듀얼코어 임베디드 리눅스 시스템 환경

모듈	타입	특징
듀얼코어 참조보드	CPU	ARM926(350Mhz) ARM946(300Mhz)
리눅스 커널(2.6.20) 크로스툴체인 Rootfs, U-boot	커널, 펌웨어	ARM926EJ기반
알템즈(4.7) 크로스툴체인	커널	ARM946E기반
Duel-IPC DD	미들웨어, 커널	RTEMS 기반
단말스킨(메뉴 GUI) 프로그램	미들웨어	윈도우 프로그램
ARM926-946연동	응용	RTEMS 기반

모듈	타입	특징
미디어 플레이어 (Mplayer)		코덱 사용 버전, 360-300Mhz
Duel-IPC POSIX API	미들웨어, 커널	RTEMS 기반
기타 관련 라이브러리 및 모듈들	라이브러리	포팅 및 테스트

(그림 12)는 듀얼코어 임베디드 리눅스 시스템에서 Shared Memory 성능 측정으로 위한 시험 환경 구성도이다. TFTP(Trivial File Transfer Protocol)/NFS(Network File System) 서버는 리눅스 환경으로 TFTP와 NFS 데몬 설치되어 커널 이미지와 파일 시스템이 존재를 하게 된다. TFTP/NFS 서버는 듀얼 코어 참조보드와 10Mbps의 LAN으로 연결되어 TFTP를 통해 서버의 커널 이미지를 듀얼 코어 참조보드의 플래시 메모리에 저장을 한다. 또한 NFS을 통한 파일 시스템을 마운트하여 사용하게 해주는 역할을 한다.



(그림 12) 공유 메모리 성능 측정을 위한 시험 환경 구성도

시험에 사용한 동영상은 MPEG-4 인코딩 영상을 기반으로 하는 코덱인 Xvid 코덱을 사용하였다. Divx보다는 다소 떨어지지만 합법적으로 사용할 수 있으며 오픈 소스로 되어있다는 장점으로 Xvid로 변환된 드라마 영상(미국)을 시험 동영상으로 사용하였다. 시험 동영상 파일을 실행하는 프로그램으로 리눅스 환경 및 다양한 코덱을 지원하는 Mplayer를 사용하였다. 측정을 위한 응용 프로그램은 리눅스에서 사용되는 멀티미디어 Player로 코어간 통신기능을 사용하여 동작이 되며, 가장 안정적이고, 많은 demuxer를 제공하고, system load가 적은 Mplayer를 사용하였다. 측정 방법은 동일한 영상(MPEG-4)을 Mplayer에서 타임 초기화 후 스트림(영상 매체)을 오픈하고, A-V decoder과 out를 세팅하고, A-V decoding을 한 상태에서 1 프레임(Frame) 단위로 A-V sync를 조절하여 측정을 하였으며, 동기화 처리는 밀리세컨드(msec) 단위로 측정을 하였다.

5.2 측정 시나리오

공유 메모리 성능 측정에 앞서 듀얼코어 임베디드 리눅스 시스템에 대한 테스트는 단위 테스트를 통해 단위 기능이 검증된 블록을 통합 시나리오에 따라 통합하면서 블록 간 인터페이스를 중심으로 아래의 기능이 정확히 수행되는가를 종합적으로 검증하였다. 첫째, 부트로더, 루트 파일 시스템 및 리눅스 커널(2.6.x 이상)에 대한 이상 유무를 확인한다. 둘째, 멀티미디어 처리, USB, 유/무선랜, 그래픽처리 등을 포함한 기본 디바이스 드라이버들의 상태 확인을 한다. 셋째, 기본 드라이버 및 커널의 수행여부를 테스팅을 위한 기본 응용 프로그램들(GUI 기반)의 실행 여부를 확인한다. 넷째, 커널은 각 코어에서 독립적으로 프로세스를 실행시킬 수 있어야 한다. 다섯째, 커널은 코어간 통신을 위한 메커니즘인 인터럽트 혹은 공유 메모리 기법을 제공해야 한다. 여섯째, IPC는 Zero-copy, address mapping 등과 같은 효율적인 기법을 사용하여 처리가 될 수 있어야 한다. 그밖에 참조보드용 듀얼코어 기반에서 코어들의 응용 프로세스들을 관리할 수 있는 스케줄링 기법이 있어야 되며, 멀티미디어 플레이어가 코어간 통신기능을 사용하여 동작하여야 한다. 한쪽 코어의 모듈이 다운이 되어도 다른 쪽 코어의 기능에는 영향 미치지 않도록 해야 한다. 이에 대한 테스트는 슬레이브 코어에서 멀티 태스크 수행시 하나의 태스크를 강제로 정지 시킨다. 코어간 공

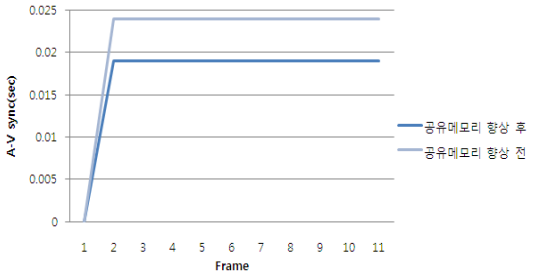
유 메모리 사용시 허용된 메모리에만 액세스가 되도록 하여야한다. 이에 대해 메모리 접근범위를 확인 한다. 공유 메모리 성능 측정은 (표 2)와 같이 테스트 케이스를 통해 진행을 하였다.

(표 2) 공유 메모리 성능 측정을 위한 테스트 케이스

	수행 프로그램	테스트 데이터	테스트 결과
시스템 부팅확인	윈도우 하이퍼 터미널 또는 리눅스 미니컴	Port : COM0 또는 COMn Bauud rate : 115200 Parity : 8 Stop bits : 1 Flow control : none	타겟보드(MP2530)에 시리얼에 연결하여 부팅 로그가 보이고 쉘 등에서 입력 가능 확인
	부트로더		시리얼로 연결된 터미널 화면에서 부트로더에서 출력하는 메시지 확인
	리눅스 커널		터미널 화면에서 커널 초기화 메시지 정상 출력 및 시스템 부팅 완료
	Hellow 응용		NFS root에서 간단한 응용 프로그램 실행 여부 확인
디바이스 드라이버	LAN 드라이버	Ping 테스트	RTT 값 출력 확인
	터치 스크린 드라이버	윈도우 화면에서 아이콘 혹은 버튼 클릭	클릭한 위치에 대한 정확한 포인팅 확인 및 해당 아이콘/버튼을 통해 정해진 작업의 정상적 수행 확인
	Video 및 Sound 관련 드라이버	미디어 플레이어 재생	영상 및 음성 출력 확인
	USB 드라이버	USB 장치 연결	파일시스템을 통한 정상적인 사용 가능 확인
응용프로그램	미디어 플레이어 (MPlayer)	동영상 파일 재생	영상 및 음성의 정상적인 출력 확인
	메뉴 프로그램		각종 버튼 및 아이콘 정상 작동 확인
마스터 코어 운영체제 및 슬레이브 코어 운영체제	마스터 코어 운영체제		루트에서 응용 프로세스 생성 및 실행
	슬레이브 코어 운영체제		슬레이브 코어 운영체제 위에서 마스터 코어와는 독립적으로 프로세스 생성 및 실행
코어간 인터페이스	듀얼 테스터 및 슬레이브 쉘 에이전트 프로그램	다음 순서로 컴맨드 입력 : Open;Run;{Test(RW||msg)}*;ShotDown;Stop;Quit	마스터 및 슬레이브 코어간 통신 레지스터 상태를 보여줌
코어간 IPC 메커니즘	커널, 코어간 IPC 테스트 프로그램	공유 메모리 어드레스	테스트 프로그램을 통해 슬레이브 코어로의 데이터 전달 확인(공유 메모리 접근 확인)
스케줄러	테스트 프로그램 (마스터 코어 운영체제에서 슬레이브 운영체제로 특정작업을 요청		슬레이브 코어에서 요청받은 작업을 정상적으로 수행하고 있음을 알리는 메시지 출력
멀티미디어 플레이어	MPlayer	동영상 파일	동영상 재생이 정상적으로 수행되는지 확인
IPC 성능 (Shared Memory)	듀얼코어용 MPlayer 프로그램	동일한 동영상 파일	각각 동영상 재생 속도를 측정하여 비교한다. 대략 10% 이상 성능 향상이 있음을 확인한다.

5.3 측정 결과

(그림 13)은 구축된 듀얼코어 임베디드 리눅스 시스템 환경에서 A-V 동기화 처리 성능 측정을 통해 공유 메모리의 성능 향상에 대한 비교 결과를 보여준다. 듀얼코어 임베디드 리눅스 커널에서 공유 메모리 향상 전과 향상 후 프레임 변화에 따른 오디오-비디오(A-V) 동기화 처리 성능 결과에 대한 그래프를 비교해 보면 공유 메모리 성능 향상 전 리눅스 커널의 오디오-비디오(A-V) 동기화 처리 속도가 0.019sec이고 공유 메모리 성능 향상 후 리눅스 커널의 오디오-비디오(A-V) 동기화 처리 속도가 0.024sec를 나타내고 있다, 차이를 보면 기존 리눅스 커널의 공유 메모리 성능 대비 10% 이상 리눅스 커널에서 공유 메모리 성능 향상이 되었음을 확인할 수 있다. 이는 듀얼코어 임베디드 리눅스 시스템에서 마스터 코어 운영체제의 루트에서 응용 프로세스 생성 및 실행과 슬레이브 코어 운영체제에서 마스터와 독립적으로 프로세스 생성 및 실행이 되어 마스터 및 슬레이브 코어간 IPI(Inter-Processor Interrupt)와 공유 메모리 통신이 사용되었으며, 코어간 통신에서 통신설정 오버헤드가 제거된 결과로 볼 수가 있다.

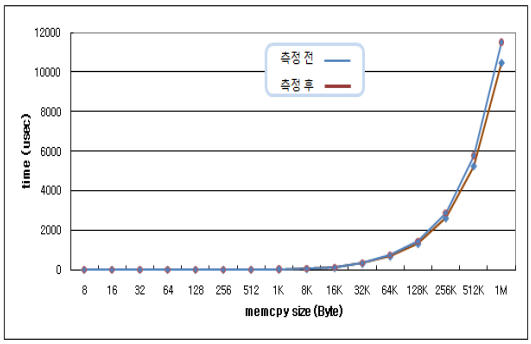


(그림 13) 공유 메모리 성능 향상 전/후 A-V 동기화 처리 결과

듀얼코어 지원 커널에서 마스터 코어와 슬레이브 코어사이의 메시지 전달이 이루어지고 슬레이브 코어를 이용하여 멀티미디어 데이터 디코딩 기능을 지원하고 있다. 중요한 기능은 인터럽트를 통한 슬레이브 코어를 제어하고 슬레이브 코어 지원을 통한 멀티미디어 데이터 처리가 이루어진다는 것이다. 또한 병렬처리율을 높임으로써 기존 리눅스 커널의 속도 대비 수정된 커널의 IPC 속도가 향상됨을 알 수가 있다. (그림 14)는 메모리 크기 변화에 따른 공유 메모리 IPC 속도 측정 결과를 나타낸 것으로 기존 대비 10% 이상 성능이 향상됨 볼 수 있다. 공유 메모리는 시스템의 가상 메모리(Virtual Memory) 구조에 크게 의존하며, 데이터의 복사나 시스템 요청을 사용하지 않고 많은 양의 데이터를 공유하는 매우 빠르고 융통성 있는 기법을 제공되나 동기화 기법을 제공하지 않으므로 공유 메모리를 사용하는 프로세스들 간의 동기화 프로토콜이 공유 메모리 성능에 영향을 주는 요소로 적용될 수 있다.

(표 3) 공유 메모리 IPC 속도 측정 결과

memory size(byte)	측정 전	측정 후
	time(usec)	time(usec)
8	15.4	14
16	15.4	14
32	15.4	14
64	15.4	14
128	15.4	14
256	16.6	15
512	17.6	16
1K	19.8	18
8K	51.2	46.7
16K	106.3	96.7
32K	348.4	316.7
64K	751.5	683.3
128K	1448.4	1316.7
256K	2874.6	2513.3
512K	5766.7	5242.5
1M	11506	10460



(그림 14) 메모리 크기 변화에 따른 공유메모리 IPC 속도 측정 결과

6. 결론 및 향후 연구

본 논문에서는 대부분의 유닉스 시스템에서 사용되는 IPC 메카니즘 중 공유 메모리의 성능을 듀얼코어 임베디드 리눅스 시스템에서 개선하고자 방안들을 제시하고 그에 대한 성능을 측정하였다. 기존의 듀얼코어 임베디드 시스템에서 커널의 통신 속도 향상이 고성능 X86 시스템과의 성능에 대한 차이를 실험적으로 나타낸 것과는 다르게 본 논문에서는 공유 메모리 영역으로 설정된 부분에서 데이터를 읽고 쓰는 부분으로 프로세스 간에 메모리를 참조하는 경우에 최대한 접근 횟수를 줄이는 방안에 대해 성능 개선 방안을 정립하고, 성능 향상 전과 후의 차이를 체계적인 시나리오를 토대로 측정을 하여 결과를 나타내었다. 본 논문에서 듀얼 코어 시스템인 임베디드 개발 시스템에서 ARM926에 리눅스 운영체제를 설치하고, ARM946에 RTEMS 운영체제를 설치하였다. 이 환경에서 두개의 서로 다른 운영체제 간에 공유 메모리 환경을 사용 가능하도록 설계 및 구현을 하였다. ARM946에 설치된 RTEMS 운영체제 상에서 공유 메모리를 구현하기 위하여 인터럽트 메카니즘을 이용하였다. 또한 운영 체제가 서로 다른 경우 사용하는 공유 메모리 구조가 다를 수 밖에 없는 상태로, 이러한 문제점을 보완하고자 Agent Program 도입 하였고, Agent Program은 서로 다른 운영체제에서 생성한 공유 메모리를 접근해 주는 역할을 한다. 제안한 내용을 구현하여 실험한 결과를 보면 기존 리눅스 커널의 공유 메모리 성능 대비 10% 이상 리눅스 커널에서 공유 메모리 성능 향상이 되었음을 확인할 수 있다. 또한 이러한 결과는 듀얼 코어 임베디드 리눅스 환경에서 공유 메모리 성능 향상이 전력대비 성능 개선 효과 및 생산원가 절감을 기대할 수 있다. 듀얼 코어 기술의 발전은 현재의 듀얼 코어 아키텍처가 가지고 있는 잠재적인 결함을 해결하기 위한 소프트웨어 관련 기술들이 뒷받침 되어야 할 것이다. 이러한 시점에서 여러 코어에서 여러 프로세스가 동작하고, 그 프로세스들 간의 통신이 중요하게 된 시점에서 본 논문에서 제안하고자 하는 IPC 메카니즘 중 공유 메모리의 성능 개선은 중요한 과제이다.

참 고 문 헌

[1] 김종수외 4인, "A Study on shared memory optimization for multi-processor system", 한국정보처리학회(추계) 2001

[2] Hardeep Singh, Rachna Dhand, Sandeep Bassi, "Inter-process communication (IPC) : An interpretive conspectus", IASTED International Conference on Communications, Internet, and Information Technology (CIIT), 2002

[3] Hai Huang, Padmanabhan Pillai, Kang G. Shin, "Improving wait-free algorithms for interprocess communication in embedded real-time systems", USENIX Annual Technical Conference, 2002

[4] 김종수외 4인, "A study on buffer and shared memory optimization for multi-processor system", 정보처리학회논문지, 2002

[5] 이상권 외 3명, "KDSM(DAIST Distributed Shared Memory) 시스템의 설계 및 구현", 한국정보과학회논문지:시스템및이론 29호 p.257-264, 2002

[6] 박성원, 정기철,"ARM 9을 이용한 임베디드 리눅스 시스템", 북두출판사, 2005

[7] A reliable multicast transport protocol suitable for distributed IPC of the TMO model / 안진섭

(한국정보과학회 학술발표논문집 (가을) 2005, Vol.1, v.1 p.532-534)
[8] A design of dynamic networking techniques supporting universal plug and play for real time object TMO distributed inter-process communication / 김성진 (차세대 통신 소프트웨어 학술대회 (NCS) 2005, p.50-53)
[9] Developing scalable signal processing applications using a flexible message based inter processor communication (IPC) framework / Sven Brehmer (Pervasive Signal Processing Conference and Expo (GSPx) 2005, p.2044-1-4)
[10] MPI management of Hermite Collocation computation on a distributed-shared memory system / E. N. MATHIOUDAKIS (WSEAS International Conference on Applied Mathematics (MATH) 2006, 290-295)
[11] Sean Walberg, “Share application data with UNIX System V IPC mechanisms”, IBM, 2007
[12] 장승주, “Dual core 시스템에서 shared memory 기능 구현”, 한국콘텐츠학회논문지 제8권 제9호 (2008년 9월) pp.27-33
[13] Optimization of job super scheduler architecture in computational grid environments / M. Shiraz (International Conference on Grid Computing and Applications (GCA) 2008, Paper 3179)
[14] 장승주외 2인, “Dual Core 시스템에서 Shared Memory 기능 설계”, 한국해양정보통신학회 논문지 제12권 제8호 (2008년 8월)
[15] Performance, area and power trade-offs in mesh-based emulated shared memory CMP architectures / Martti Forsell (International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA) 2008, Paper 4073)
[16] Qiao Xiangzhen, “Design of efficient parallel algorithms on shared memory multiprocessors”, Wuhan University Journal of Natural Sciences Volume 1, 2008
[17] 정지성외 3인, “고속 IPC 기술기반 듀얼코어 임베디드시스템 성능 분석”, 한국통신학회 추계종합학술발표회 2008, p.400-401

◐ 저 자 소 개 ◑

정 지 성
2002년 우송대학교 전자정보통신공학과 졸업(학사)
2008년 공주대학교 대학원 정보통신공학과 수료(석사)
1999~현재 한국전자통신연구원 연구원
관심분야 : 임베디드 리눅스, 네트워크 시스템, etc
E-mail : jungjs@etri.re.kr

김 창 봉
1983년 고려대학교 전자공학과 졸업(학사)
1988년 Florida Tech 대학 전자공학과 졸업(석사)
1992년 Texas A&M 대학 전자공학과 졸업(박사)
1993~현재 공주대학교 정보통신공학부 교수
관심분야 : 광통신망, 광전송, etc.
E-mail : aggie@kongju.ac.kr