Dual Core 시스템에서 Shared Memory 기능 설계

장승주*, 이광용**, 김재명**

The Design of the Shared Memory in the Dual Core System

Seung-Ju Jang*, Gwang-Yong Lee**, Jae-Myeong Kim**

이 논문은 2007년도 한국전자통신연구원 연구비를 지원받았음.

요 약

본 논문은 대부분의 Linux 운영체제에서 지원해 주는 System V의 IPC 중 하나인 Shared Memory를 Dual Core 시스템 상에서 동작하도록 설계한다. Linux에서 사용되는 Shared Memory는 동일한 메모리 영역에 여러 개의 프로세스가 접근할 수 있도록 해 주는 기술이다. 본 논문에서는 Shared Memory의 큰 두 갈래 중 커널 단계에서 처리 되는 SVR(System V Release) 형식의 Shared Memory를 다룬다. 본 논문에서는 리눅스 운영체제의 공유 메모리 기능을 Dual Core 시스템에서 동작하도록 설계한다. 본 논문에서 제안하는 Dual Core 시스템에서 공유 메모리 기능 설계 방안은 듀얼 코어를 활용하여 기존의 단일 처리기 시스템에서보다 성능을 향상시킬 수 있도록 한다. 공유 메모리를 이용한 프로세스의 동작이 별개의 CPU에서 동작되도록 함으로써 성능 향상을 꾀한다.

ABSTRACT

This paper designs Shared Memory on the Dual Core system so that it operates a general System V IPC on the Linux O.S. Shared Memory is the technique that many processes can access to identical memory area. We treat Shared Memory in this paper among big two branches of Shared Memory which are SVR in a kernel step format. We design a share memory facility of Linux operating system on the Dual Core System. In this paper the suggesting design plan of share memory facility in Dual Core system is enhancing the performance in existing unity processor system as a dual core practical use. We attempt a performance enhance in each CPU for each process which uses a share memory.

키워드

Auto White Balance, CIEL*a*b* color space, Color Sampling, color gain

Ⅰ. 서론

Linux에서 사용되는 Shared Memory(공유메모리)는 동일한 메모리 영역에 여러 개의 프로세스가 접근할 수 있도록 해 주는 매력적인 기술이다. 일반적으로 SVR IPC(System V Release InterProcess Communication)라 불리는 공유메모리, 세마포어, 메시지 큐의 세 가지는 System V 유닉스에서 구현된 기술이다.

본 논문에서는 SVR IPC에서 사용되는 기술 중 공유메모리(Shared Memory)를 Dual Core 시스템 상에서 동작하도록 설계한다.

복잡한 프로그래밍 환경에서 다수의 프로세스들은

* 동의대학교 컴퓨터공학과 교수

접수일자 2008. 03. 20

서로 협력하기 위하여 서로 통신하고 자원과 정보를 공유한다. 커널에서는 이것이 가능한 방법을 제공하는 데, 이를 프로세스간 통신(Inter-Process Communication) 또는 IPC라 부른다 [1-3]. 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. 그러나 많은 사용자 프로그램들에서는 이것만으로 IPC를 충족할 수가 없었으며, 이것은 SystemV 릴리즈 버전에서 메시지 큐, 세마포어, 공유 메모리 기법을 사용하게 된다 [4-8].

Shared Memory 기법은 시스템의 가상 메모리(Virtual Memory) 구조에 크게 의존하며, 데이터의 복사나 시스템 요청을 사용하지 않고 많은 양의 데이터를 공유하는 매우 빠르고 융통성 있는 기법을 제공한다. 그러나 이 기법은 동기화 기법을 제공하지 않는다는 단점이 있다. 두 개의 프로세가 있다고 가정하면, 공유 메모리 영역의 내용을 동시에 변경하려고 할 경우, 커널은 이를 순차적으로 처리를 해야 하나, 그렇게 하지 않으면, 쓰인 데이터는 엉키게 된다. 그래서 공유 메모리를 사용하는 프로세스들은 프로세스들 간에 동기화를 하는 프로토콜을 고안해야 하며, 이것은 공유 메모리 성능에 영향을 주는 요소로 적용할 수 있다.

본 논문은 2장에서 Shared Memory에 관하여 설명하고, 3장에서 Dual Core 시스템에서 동작하는 Shared Memory 기능의 설계 방향에 대해 설명하고, 4장에서 Dual Core 시스템에서 동작하는 Shared Memory 기능 설계를 설명한다. 5장에서 결론을 맺는다.

II. 관련 연구

유닉스의 두 갈래 중 하나인 SVR은 초기 상용 유닉스 버전들이 해당한다고 보면 되며, 다른 한 갈래인 BSD는 버클리 대학에서 독자적으로 구현한 것이다. 초기 BSD는 대학 내에 설치된 상용 유닉스 위에 학생들이 필요한 소프트웨어를 만들며 시작된, 일종의 Free Software 모음이었으나 그 덩치가 커지면서 OS 단계까지 구현하게 되었다 [7-11].

결국 상용 유닉스 업체의 법적 소송으로 인해 완전히 소스를 분리하여 새로운 유닉스 버전을 만들게 되었다.

BSD와 SVR은 서로 다른 길을 걷게 되었지만 나중엔 다시 합쳐져서 현재의 상용 유닉스들은 대부분 통합된 패키지를 제공하고 있다.

그 예로, TCP/IP 관련은 BSD에서 먼저 구현되었으며, SVR은 IPC API들을 구현하였다. 현재 BSD의 후계자인 FreeBSD 등의 유닉스에서는 SVR IPC를 직접적으로 지원하지는 않고 있다 [8, 12-16].

IPC의 Shared Memory는 shmget( )으로 시작하는 SVR 형식의 Shared Memory와 shm_open( )으로 시작하는 POSIX 형식의 Shared Memory로 크게 나뉜다.

두 가지 Shared Memory의 큰 차이점은 SVR 버전은 커널 단계에서 구현되었고, POSIX 버전은 라이브러리로 응용프로그램 단계에서 구현되었다. POSIX 버전의 Shared Memory는 일반 시스템 콜을 활용한 라이브러리이다 [16].

널리 사용되는 SVR 버전은 대부분의 유닉스 커널과 리눅스에서 지원하며 커널에 포함되어 있기 때문에 커널 파라미터를 조정해서 제한 사항을 바꿀 수 있다. API(Application Programming Interface)를 통해 Shared Memory를 할당 받으면 해당되는 메모리 영역의 주소를 받을 수 있다. 메모리 영역의 구분은 key와 id를 통해 다른 프로세스가 접근 할 수 있으며 메모리 영역별로 euid라든가 permission 같은 메타 정보 역시 따로 커널에서 관리하게 된다.

반면 POSIX 버전은 일종의 메모리 디스크 방식으로 디렉터리에 마운트 한 후, 거기에 파일을 만드는 식으로 접근을 한다. 파일은 기본적으로 여러 프로세스에서 접근할 수 있는 개체이고 이 파일을 메모리 상에 만드는 것이므로 우회적으로 Shared Memory의 구현이 가능한 것이다. 그리고 메모리에 만든 이 파일을 mmap( )으로 매핑하면 실제 메모리를 쓰듯이 쓰게 된다. 파일이란 개체에 원래 permission과 euid등의 정보가 존재하므로 SVR 버전처럼 이에 대한 별도의 관리도 필요 없으며 커널에서는 파일로 인식할 뿐이다.

Shared Memory에 대해 SVR 버전은 '특수한 메모리 영역'으로 지정하여 사용자에게 사용하게 해 준다. POSIX 버전은 일반 메모리를 파일시스템처럼 써서 공유가 가능하게 한 후 다시 mmap( )으로 하여 메모리처럼 쓰게 해준다. mmap( )은 실제 파일이라도 파일 오프셋에 가상 주소를 할당해 메모리처럼 쓸 수 있게 해준다.

하지만 POSIX Shared Memory는 느리다. SVR처럼 커

널 단계에서 커널 모듈을 거쳐 직접 메모리에 접근하는 것이 아닌, 라이브러리 단계에서 mmap( )과 파일 시스템 API를 거쳐서 간접적으로 메모리에 접근하기에 느리다.

POSIX Shared Memory가 단점만 있는 것은 아니다. 장점은 파일 형태로 구현되어 있기 때문에 사용 하는 만큼만 메모리가 늘어난다는 점이다. SVR의 경우 미리 쓸 만큼의 메모리를 할당 받고 사용하며, 할당량 보다 크게는 사용 할 수 없다.

Ⅲ. Dual Core 시스템에서 동작하는 Shared Memory 기능의 설계 방법

Shared Memory는 물리적인 메모리의 특정 영역에 대하여 프로세스들에 의해 공유되어지는 공통의 영역을 구성한다. Shared Memory는 공통의 영역을 프로세스들이 사용할 수 있게 해준다.

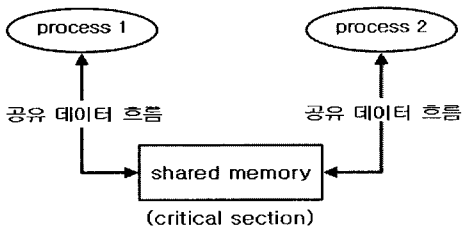

그림 1. Shared Memory 구조
Fig 1. Shared Memory Structure

그림 1.은 Shared Memory의 구조를 나타낸다. Shared Memory는 물리적인 메모리의 특정 영역에 대하여 프로세스들에 의해 공유되어지는 그림 1.의 Critical Section (임계 영역)이라 할 수 있다. 프로세스들은 이 Shared Memory 영역을 자신의 가상 메모리 영역에 부착(Attach)하여 사용한다. Critical Section은 데이터 읽기/쓰기의 시스템 호출을 사용하지 않고 다른 방법으로 접근하기 때문에 프로세스 간 데이터를 공유하는 기법 중 가장 빠르다 할 수 있다. 예를 들어, 하나의 프로세스가 공유 메모리의 영역에 기록(Write)을 하게 된다면, 이 내용은 이 영역을 공유하는 모든 프로세스들에 의해 바로 보여 지게 되어 아주 빠르게 데이터를 공유하게 된다.

그림 2.는 2 CPU에서의 IPC(본 논문에서는 Shared Memory)의 구조이다. 서로 다른 CPU에 서로 다른 운영체제가 IPC 메카니즘 중 Shared Memory 기법으로 서로의 데이터를 공유하는 모습이다.

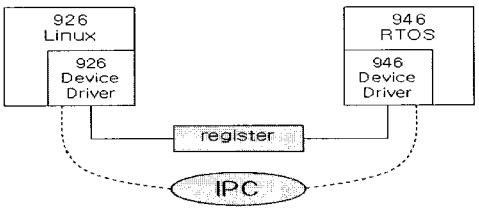

그림 2. 2 CPU에서의 IPC 구조
Fig. 2. IPC Structure in 2 CPUs

그림 2.와 같이 서로 다른 CPU를 사용할 경우에 각각의 CPU에서 Shared Memory를 통하여 데이터를 공유하거나 정보를 주고받을 수 있다. 또한 서로 다른 CPU에서 동시에 다른 프로세스가 수행될 수 있다. 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 그림 2.와 같이 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.

Ⅳ. Dual Core 시스템에서 동작하는 Shared Memory 기능의 설계

그림 3.은 사용자 공간에서 사용한 shmget( )의 동작 과정을 보여준다. 그림 3.의 ①은 shmget( )의 첫 번째 인자로 들어온 값이 IPC_PRIVATE일 경우 newseg( )를 호출하여 새로운 세그먼트를 생성한다.

그림 3.의 ②는 shmget( )의 첫 번째 인자로 들어온 값

이 IPC_PRIVATE가 아닐 경우 ipc_findkey()를 이용하여 id를 찾는다. 만약 id를 찾지 못한다면 그림 3.의 ③과 같이 newseg( )를 호출하여 새로운 세그먼트를 생성하게 된다.

그림 3.의 ④는 shmget( )의 첫 번째 인자로 들어온 값을 갖는 공유메모리 공간을 생성한다. 그림 3.의 shmget( )의 반환 값은 실제 공유메모리의 id를 반환한다.

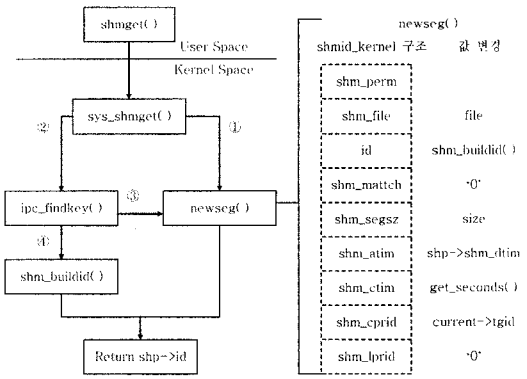

그림 3. shmget( ) 동작과정
Fig. 3. shmget( ) Procedure

그림 4.는 Dual CPU에서 서로 다른 운영체제, ARM 926은 Linux 운영체제를 ARM 946은 RTEMS 운영체제를 사용할 경우 Shared Memory를 구현하기 위한 구상도이다.

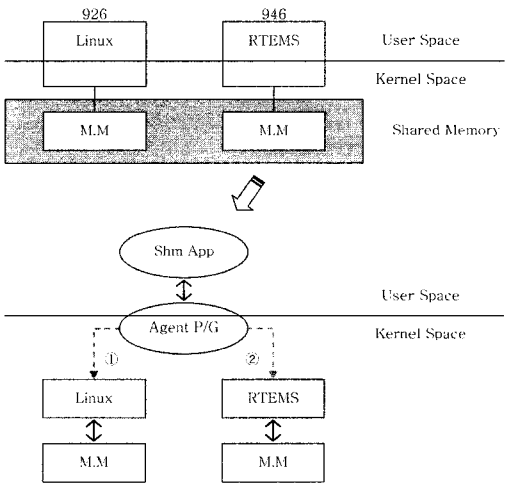

그림 4. Dual Core에서 Shared Memory 설계 구상도
Fig. 4. Shared Memory Design Architecture in Dual Core

운영체제가 서로 다른 경우 사용하는 Shared Memory의 구조가 다를 수 밖에 없다. 이러한 문제점을 보완하고자 그림 4.의 "Agent Program"을 도입하였다. "Agent Program"은 그림 4.의 ①, ②와 같이 서로 다른 운영체제에서 생성한 Shared Memory로의 링크를 행해준다.

그림 4.의 ①은 Linux 운영체제에서Shared Memory를 생성하였다. Linux 운영체제의 관리 아래에서 동작하고 있는 Shared Memory를 RTEMS 운영체제에서 필요 하게 될 경우 Agent Program에서 그림 4.의 ①과 같은 링크를 생성하여 Linux 상의 Shared Memory를 사용할 수 있게 해준다.

그림 4.의 ②는 그림 4.의 ①과 반대로 RTEMS 운영체제에서 생성한 Shared Memory에 Linux 시스템에서 동작하는 프로그램이 접근하고자 할 경우 그림 4.의 ②와 같은 링크가 생성되어 RTEMS 운영체제 상의 Shared Memory를 사용할 수 있게 해준다.

설계 방법은 두 개의 코어 중에서 master와 slave를 지정한다. 공유 메모리 기능의 구현은 agent를 통해서 이루어진다. Agent내 master node의 기능을 이용해서 slave의 기능을 이용한다. Master와 slave간에 데이터 전송 방식은 RPC 메커니즘을 이용한다.

Dual Core의 서로 다른 CPU에서 독립적으로 갖고 있는 Main Memory에서 할당되는 Shared Memory 정보를 저장하기 위해 새로운 자료구조를 만들었다.

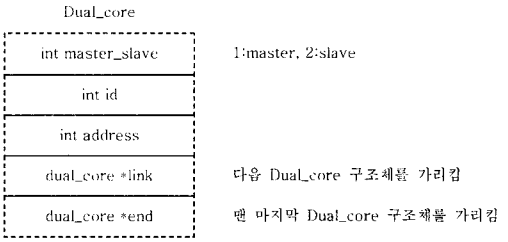

그림 5. 새로운 자료구조
Fig. 5. New Data Structure

그림 5.는 Shared Memory 정보를 저장하기 위한 새로운 자료구조이다. 자료구조의 중요부분은 master와 slave를 구분할 수 있는 변수인 master_slave 변수와 link, end 변수이다.

master_slave 변수의 경우 그 값이 '1'이면 master, 즉 원래의 운영체제인 Linux 운영체제에서 생성되고 관리

되어 진다는 의미이다. master_slave 변수의 값이 '2'이면 slave, 즉 원래의 운영체제가 아닌 다른 운영체제인 RTEMS 운영체제에서 생성되고 관리되어 진다는 의미이다.

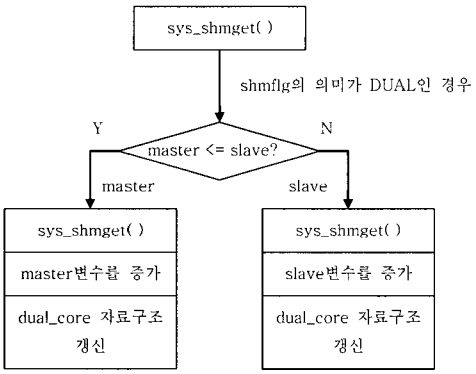

그림 6. 새로 작성된 sys_shmget() 함수 구조
Fig. 6. sys_shmget() Function Structure

그림 6.은 Shared Memory 정보를 저장하기 위한 새로운 자료구조와 그 자료구조를 활용하여 Dual CPU에서 shmget( )을 사용할 수 있게 만들 sys_shmget( ) 함수의 구조도이다.

sys_shmget( )함수의 마지막인자인shmflg 변수의 값으로 Dual CPU에서 동작할 응용프로그램인지를 구분하게 된다. 본 문서에서는 shmflg 변수의 자세한 값보다는 shmflg의 의미가 Dual CPU 응용프로그램이다 라고 말하고 있다.

만약 shmflg의 의미가 Dual CPU 응용프로그램이라고 한다면, master쪽(Linux)인지 slave쪽(RTEMS)인지를 구분하여 master라면 원래 Linux의 sys_shmget( ) 함수 코드가 실행이 되고, slave라면 RTEMS의 shmget( )에 해당하는 함수가 수행 될 수 있도록 환경을 조성해 주면 된다.

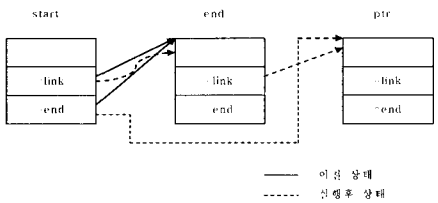

그림 7. 자료구조의 갱신 동작과정
Fig. 7. Data Structure Updating Procedure

그림 7.은 그림 6.의 "dual_core 자료구조 갱신" 부분에서의 동작과정을 나타낸다. 실선으로 표시된 부분이 기존에 유지되고 있던 자료구조가 새로운 정보가 입력이 되면서 자동적으로 점선으로 표시된 부분처럼 변경됨을 보여준다.

'dual_core' 자료구조의 link 변수는 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' 자료구조 이며, 마지막인 '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.

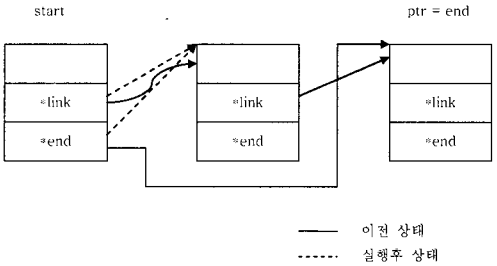

그림 8. shmdt()에서 자료구조의 갱신 동작과정 - a
Fig. 8. Data Structure Updating Procedure in shmdt()-a

그림 8.은 shmdt( ) 함수에서 자료구저의 갱신 동작과정을 보여 주고 있다. 그림 8.의 동작과정은 현재의 자료구조가 'end'일 때의 동작과정이다. shmdt( ) 함수는 shmget( )과 달리 'ptr'의 위치가 임의 적이다. shmget( )함수의 경우 항상 맨 마지막이 'ptr' 이기 때문에 한가지 경우에 대한 동작과정만 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 마지막인 경우(동작과정 - a)는 'start'에서 가리키던 'end'의 주소를 'ptr' 바로 앞 자료구조의 주소로 변경해주어야 한다. 그 이외의 부분은 자료구조들의 연결에 영향을 주지 않는다.

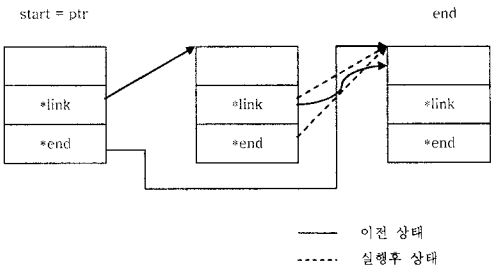

그림 9. shmdt( )에서 자료구조의 갱신 동작과정 - b
Fig. 9. Data Structure Updating Procedure in shmdt()-b

그림 9.는 shmdt( )에서 두 번째에 해당하는 'ptr'이 맨 처음인 경우(동작과정 - b)이다. 'ptr'이 맨 처음일 때, 'start'의 주소값을 'start'다음 자료구조의 주소값으로 바꾸어 주어야한다. 이는 'start'의 값을 이용하여 자료구조들을 따라가면서 'ptr'을 찾는 for loop 문에서 사용하는 중요한 지표이기 때문에 제일 먼저 'start' 값을 수정해 주어야 한다. 그리고 'start'에서 가리키고 있는 맨 마지막 자료구조의 주소를 'start' 다음 자료구조의 'end' 의 값으로 넣어 주면 된다.

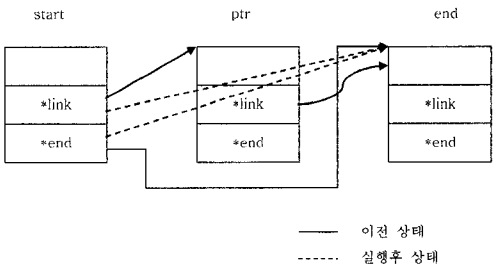

그림 10. shmdt( )에서 자료구조의 갱신 동작과정 - c
Fig. 10. Data Structure Updating Procedure in shmdt()-c

그림 10.은 shmdt( )에서 세 번째에 해당하는 'ptr'이 중간인 경우(동작과정 - c)이다. 이 경우는 'ptr'에 해당하는 자료구조의 바로 앞 자료구조를 수정해 주어야 한다. 먼저 'ptr'의 바로 앞 자료구조를 찾고, 두 번째로 'ptr' 바로 앞 자료구조의 'link' 값을 'ptr' 자료구조의 'link' 값으로 변경해야 한다.

그림 11.은 sys_shmget( ) 함수가 호출되었을 때, Dual Core 용 sys_shmget( )을 수행해야 하는 응용프로그램(Agent Program)에서 왔는지 알 수 있게 하는 shmflg 값을 판별한다.

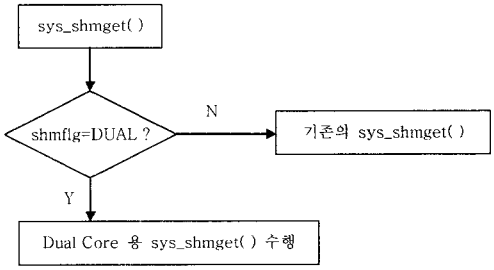

그림 11. Dual Core용 sys_shmget( ) 함수 수행 흐름
Fig. 11. sys_shmget( ) Function Procedure in Dual Core

sys_shmget( ) 함수의 마지막인자로 들어 오는 shmflg 값은 기존 함수에서는 공유 메모리를 찾을 수 없을 때, 공유 메모리 생성 여부를 판별하기 위해 사용하고 있었다. 기존에 사용되던 shmflg 값은 그대로 두고 우리가 원하는 DUAL Core 용의 sys_shmget( ) 함수를 수행하기 위한 판별 값을 '0x12345' 라는 키로 정의 하였다.

sys_shmget( ) 함수가 호출되면 shmflg 값이 '0x0200000' 인가를 판별하여 shmflg 값이 '0x0200000' 이면 우리가 작성한 Dual Core 용 sys_shmget( ) 코드를 수행한다. shmflg 값이 '0x0200000'가 아니면 기존의 sys_shmget( ) 코드를 수행한다.

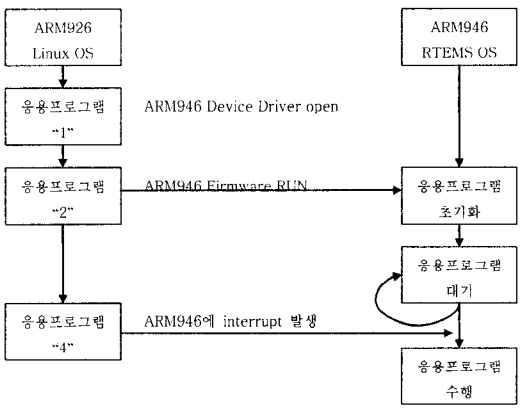

그림 12. 926과 946의 통신과정
Fig. 12. Connection Procedure in 926 and 946

그림 12.는 ARM926의 리눅스 운영체제와 ARM946의 RTEMS 운영체제가 서로 통신하는 과정을 나타낸 그림이다. ARM926의 리눅스 운영체제에서 동작하는 응용프로그램의 커멘드에 의해 ARM926의 리눅스 운영체제와 ARM946의 RTEMS 운영체제가 통신을 한다.

ARM926의 리눅스 운영체제에서 동작하는 응용프로그램의 커멘드는 "1, 2, 3, 4, 5, H, Q"로 나뉜다. 커멘드 "1"의 경우 ARM946의 Device Driver를 열어주는 역할을 한다. 커멘드 "2"는 ARM946의 Firmware를 실행시킨다. 커멘드 "2"에서 ARM946의 RTEMS 운영체제에서 동작하는 프로그램이 초기화를 시작한다. 초기화가 끝난 ARM946의 RTEMS 운영체제에서 동작하는 프로그램은 ARM926에서 interrupt를 발생시킬 때까지 대기한다.

커멘드 "3"은 그림에 나와 있진 않지만 ARM946의 Firmware를 종료시키는 기능을 갖고 있다. 커멘드 "4"는 ARM926의 리눅스 운영체제 커널 단계 프로그램을 실행시켜 ARM946의 RTEMS운영체제에서 동작하는 프로그램에 interrupt를 발생시킨다. 발생된 interrupt에 의해 ARM946의 RTEMS 운영체제에서 동작하는 프로그램이 실행된다.

커멘드 "H"는 제공하는 커멘드 메뉴를 보여준다. 마지막으로 커멘드 "Q"는 ARM926의 리눅스 운영체제에서 동작하는 응용프로그램을 종료한다.

V. 결론

여러 코어에서 여러 프로세스가 동작하고, 그 프로세스들 간의 통신이 중요하게 된 시점에서 본 논문에서 제안하고자 하는 IPC 메카니즘 중 Shared Memory의 기능 설계 중요한 과제이다. 본 논문에서 제안하는 Dual Core 시스템에서 공유 메모리 기능 설계는 듀얼 코어를 활용하여 기존의 단일 처리기 시스템에서보다 성능을 향상시킬 수 있다. 공유 메모리를 이용한 프로세스의 동작이 별개의 CPU에서 동작되도록 함으로써 성능 향상을 꾀한다. 본 논문은 대부분의 유닉스 시스템에서 사용되는 IPC 메카니즘 중 Shared memory의 기능을 Dual Core 시스템에서 동작할 수 있도록 설계해 보았다.

임베디드 시스템에서 성능 향상을 위하여 Dual core CPU를 사용하는 추세로 가고 있다. 나아가 임베디드 시스템은 지금보다 많은 응용 프로그램들이 탑재될 것이다. 이럴 경우에 여러 개의 CPU를 가진 임베디드 시스템 환경을 제공하여 빠른 성능을 보장해 주어야 한다.

본 논문에서 제안하는 Dual Core 시스템에서 Shared Memory 설계 방법으로 Shared Memory를 구현한다면 Single Core 시스템과 Dual Core 시스템에서 빠른 성능을 보장 할 수 있다.

참고 문헌

[1] 김선욱, 오재근, 한영선, 최홍욱, 김철우, "임베디드 응용 프로그램 성능 분석", 대한전자공학회:학술대회지, 대한전자공학회 03년도 하계종합학술대회 논문집III, pp. 1355-1358, 2003.

[2] 박재호, "임베디드 리눅스", 한빛미디어, 2002.11. [3] Uresh Vahalia, "UNIX Internals", 홍릉과학출판사, 2001

[3] 박윤미, 성영락, "임베디드 리눅스시스템 설계 및 구현", 한국정보과학회2003 봄 학술발표논문집, pp. 214-216, 2003[4] 한동훈, "시스템V IPC - 공유메모리", 1997

[4] 김용준, "공유메모리를 이용한 채팅방의 원리"

[5] 한동훈, "공유메모리 vs 세마포어를 이용한 chat program", 1997

[6] Sean Walberg, "Share application data with UNIX System V IPC mechanisms", IBM, 2007

[7] 박찬모, "분산 공유 메모리 시스템 설계에 관한 연구", 조선대학교 동력자원연구소 동력자원연구소지 19권 1호 p.129-143, 1997.05

[8] 이병관, "분산 공유 메모리에서 일관성 제어 프로토콜", 관동대학교 부설 산업기술개발연구소 산업기술논문집 12호 p.69-78, 1997.10

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

[10] 박기홍, "Shared Memory를 갖는 멀티프로세서 시스템 구현에 관한 연구", 군산대학 자원과학연구소 3호 p97-103, 1988

[11] 양제현, "Scalable Synchronization in Shared Memory Multiprocessing System", University of Marland, 1994.

[12] Michael Barr, "Programming Embedded Systems in C and C++", O'Reilly, 1999

[13] 박성원, 정기철, "ARM-9을 이용한 임베디드 리눅스 시스템", 복두출판사, 2005

[14] Craig Hollabaugh, "Embedded Linus Hardware,

Software and Installing", Pearson Education, 2002
[15] 조주현, "임베디드 실시간 시스템의 개발환경", 한국정보처리학회지, 제9권 제1호, pp. 120-126, 2002
[16] 홍진기, 문종려, 백승걸, 정선태, "Dual CPU 기반 임베디드 웹 카메라 스트리밍 서버의 설계 및 구현", 대한전자공학회:학술대회지, 대한전자공학회 03 신호처리소사이어티 추계학술대회 논문집, pp. 417-420, 2003.

저자소개

장 승 주(Seung-Ju Jang)

1985년 부산대학교 계산통계학 (전산학) 학사
1991년 부산대학교 계산통계학 (전산학) 석사
1996년 부산대학교 컴퓨터공학과 박사
1987년~1996년 한국전자통신연구원 시스템 S/W연구실
1993년~1996년 부산대학교 시간강사
2000년~2002년 University of Missouri at Kansas City, visiting professor
1996년~현재 동의대학교 컴퓨터공학과 교수
※관심분야 : 운영체제, 임베디드 시스템 운영체제, 자동차용 임베디드 운영체제, 분산시스템, 시스템 보안

이광용(Gwang-Yong Lee)

1991 숭실대학교 전자계산학과 (학사)
1993 숭실대학교대학원 전자계산학과 (석사)
1997 숭실대학교대학원 전자계산학과 (박사)
1997~1998 ETRI 컴퓨터·소프트웨어기술연구소 소프트웨어공학연구부 Post Doc.
1999~현재 ETRI, 융합S/W기술연구본부 임베디드 SW기술연구팀 선임연구원
※관심분야: 임베디드S/W, 멀티코어기술, 임베디드 OS, 센서네트워크, 실시간시스템, 정형기법, 객체지향 개발방법론, 소프트웨어공학

김재명(Jae-Myeong Kim)

1983년 2월: 부산대학교 계산통계학과 이학사
1985년 2월: 한국과학기술원 전산학과 공학석사
2006년 1월~2007년 2월: ETRI 임베디드SW연구단 임베디드OS연구팀장
2008년 3월 ~ 현재 : ETRI 융합SW연구본부 조선융합기술연구팀
※관심분야: 컴퓨터 구조, 시스템 SW, 임베디드 통신