Regular paper http://dx.doi.org/10.13067/JKIECS.2014.9.6.703

임베디드 시스템에서의 공유 메모리 컨트롤러 디바이스 드라이버 설계

문지훈* · 오재철**

Design of Shared Memory Controller Device Driver in Embedded System

Ji-Hoon Moon* · Jae-Chul Oh**

요 약

단일 시스템에 코어별 운영체제를 사용하는 AMP(Asymmetric Multiprocessing) 기반 듀얼 코어에서 프로세서 간 데이터를 전달하기 위해서 공유 메모리 기법을 사용한다. 서로 다른 운영체제에서 공유 메모리를 사용하기 위해서는 두 운영체제 사이의 메시지 통신 및 동기화 문제를 해결해 주어야 하는 문제점이 발생한다. 본 논문에서는 듀얼 코어 환경에서 서로 다른 프로세서 코어 사이에서 데이터 공유를 위해서 별도의 메모리 컨트롤러를 이용하였다. 이 컨트롤러는 두 프로세서에서 동시에 접근이 가능 하도록 두 개의 슬레이브 포트를 지정할 수 있으며, 두 프로세서에 의해서 동시에 데이터 처리를 수행할 경우 메모리 중재자에 의해서 슬레이브 포트의 우선 순위를 결정하게 된다. A에서 B 프로세서로 데이터를 전달 시, SRAM 영역을 논리적으로 8개의 페이지로 분리하였다. 여러 프로세스에서 메모리 영역을 사용 하도록 하였으며 페이지당 4KByte의 크기를 갖도록 하였으며, 현재 페이지가 사용 가능한지 아닌지를 판별하기 위해서 4바이트 크기의 컨트롤 레지스터를 이용하였다.

ABSTRACT

In the AMP(Asymmetric Multiprocessing) based dual core using core-specific operating system in a single processor system, shared memory method is used to send data between processors in dual core. To used shared memory in different operating systems, there is a problem of needing to solving the issue of message communication and synchronization between the two operations systems. In this paper, separate memory controller was used for data sharing between different processor cores in dual core environment. This controller can designate two slave ports to allow simultaneous access from two processors, and in the case of process data simultaneously by two processors, priority order of slave ports is determined through memory mediator. When sending data from A to B processor, SRAM area was logically separated into 8 pages. It allowed using memory area from multiple processes with the size of 4KByte per page, and control register with the size of 4Byte was used to discern the usability of current page.

키워드

Embedded System, Dual Core, Shared Memory, Device Driver
임베디드 시스템, 듀얼 코어, 공유 메모리, 디바이스 드라이버

* 교신저자(corresponding author) : 이니텍 보안개발2본부 DB보안팀 차장(jihoon.moon@initech.com)
** 순천대학교 컴퓨터과학과 교수(ojc@sunchon.ac.kr)
접수일자 : 2014. 04. 18 심사(수정)일자 : 2014. 05. 23 게재확정일자 : 2014. 06. 16

Ⅰ. 서 론

IT 환경의 발달로 3D-TV, VOD등과 같은 다양한 영상정보매체 기술은 매우 다양한 분야로 발전을 거듭하고 있다[1]. 이는 주 처리장치인 마이크로프로세서 발전뿐만 아니라 관련 데이터를 처리할 수 있는 임베디드 기기의 발전으로 가능하게 되었다[2]. 임베디드 프로세서는 단순히 장치 내에 내장되는 마이크로프로세서를 지칭한다[3]. 최근 임베디드 시스템의 고성능화로 인하여 복잡한 성능을 가지는 다중 코어 프로세서들이 요구되고 있다. 멀티 코어 프로세서를 이용하여 기존의 임베디드 시스템에서 프로세서의 성능 문제로 인하여 불가능한 작업이 가능하게 되었다. 예를 들어 복잡한 성능을 요구하는 임베디드 시스템인 경우 연산 처리를 다른 프로세서에서 처리 하도록 하여 성능 향상을 할 수 있게 된다. 위의 시스템에서 성능 향상을 위해서 연산 처리 부분을 공유 메모리를 이용하여 다른 프로세서로 전달 처리하여 보다 고성능의 처리가 가능하다. 멀티코어에서 시스템의 성능을 향상시키기 위해서 동일 운영체제를 사용하는 SMP의 경우 어렵지 않게 구현이 가능하나, 서로 다른 운영체제가 탑재되는 AMP의 경우 프로세서에서 사용되는 운영체제의 물리적인 주소가 다르게 되는 문제점이 발생한다. 위의 문제점으로 인하여 기존 논문에서는 멀티코어에서 운영체제가 다른 시스템에서 공유 메모리를 구현하기 위해 운영체제가 가지고 있는 자료구조를 수정하여 두 프로세서 사이의 공유 메모리를 구현하였다[4-6]. 즉, 서로 다른 운영체제를 사용되는 시스템에서 공유 메모리에 데이터를 기록할 경우 각 프로세서에서 사용되는 물리적인 메모리가 다르게 되므로 A 프로세서에서 사용되는 공유 메모리 주소와 B 프로세서가 사용되는 주소는 다르게 되므로, A 운영체제의 공유 메모리 영역으로 데이터가 기록 되었을 때 B 운영체제의 공유 메모리 영역도 동일한 데이터로 갱신해 주어야 하기 때문에 운영체제의 자료구조의 수정이 불가피하다.

본 논문에서는 임베디드 시스템에서 대중적으로 사용되는 ARM 프로세서 멀티 코어에 EMC(: Embedded Memory Controller)를 추가하는 방법을 제안한다. 운영체제에서 사용되는 공유 메모리 영역을 이용하는 것이 아니라 별도의 컨트롤러를 추가하는 방식을 제안한다. 이 컨트롤러는 32KByte의 SRAM으로 구성되어 있으며, 8개의 논리적인 세그먼트로 분할하였다. 따라서 세그먼트의 수는 8이며, 크기는 4KByte이다. 메모리 영역에 접근할 수 있는 디바이스 드라이버를 설계 하였으며, 이를 통하여 세그먼트 아이디 할당 및 해제 그리고 기록 및 읽기가 가능하도록 하였다. 두 프로세서 코어에 리눅스에서 개발된 공유 메모리 컨트롤러 디바이스 드라이버를 컴파일하여, 두 프로세서 사이에 데이터를 기록 및 읽기가 정상적으로 수행 되는지를 테스트 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 설명하고, 3장은 본 논문에서 사용된 듀얼 코어 시스템에서의 공유 메모리 컨트롤러 디바이스 드라이버 설계를 설명한다. 그리고 4장에서는 본 연구에서 제안하는 공유 메모리 컨트롤러 드라이버 실험을 보이고, 마지막 5장에서 결론을 맺는다.

Ⅱ. 관련연구

2.1 듀얼 코어 프로세서

듀얼 코어 프로세서는 2개 이상의 독립적인 프로세서들을 단일 패키지 형태, 즉 단일 칩으로 만드는 것을 말한다[4]. 듀얼 코어 아키텍처는 일반적으로 하나의 운영체제만 실행되는 방식, 서로 다른 프로세서 코어를 탑재하여 한 프로세서는 운영체제를 실행하며 다른 코어는 DSP(: Digital Signal Processors)를 수행하는 방식, 서로 같은 코어를 사용하지만 서로 독립적인 운영체제를 수행하는 방식으로 나누어진다[7].

프로세서의 부하 균등화 기술은 어떻게 업무를 코어에 적절히 분배해 줌으로써 듀얼코어의 성능을 높이는가에 초점을 둔 기법으로 세 가지 멀티프로세싱 모델이 있다[4]. AMP 아키텍처는 각 프로세서 코어에 운영체제를 독립적으로 수행 시키는 기술을 의미한다. 위의 경우 크게 이종과 동종 구조로 나누어진다. 이종은 프로세서 코어마다 각기 다른 운영체제를 탑재하는 방식을 말하며, 동종은 각 프로세서 코어에 동일한 운영체제를 탑재하는 것을 의미한다. SMP(: Symmetric Multiprocessing) 기술은 하나의 운영체제에 모든 프로세서 코어들을 사용하여 성능에 초점을 맞추는 것을 말한다. 위의 모델은 두 개 이상의 동일

프로세서들이 공유 메모리를 통해 서로 연결되어 있다. 단일 프로세서에 비해 가격대비 성능이 좋은 소규모 SMP 시스템을 클러스터의 노드로 많이 사용한다[8]. 각 프로세서는 공유 메모리를 통하여 데이터 공유를 할 수 있다. BMP(: Bound Multiprocessing) 기술은 하나의 운영체제가 모든 프로세서 코어를 동시에 관리하나 특정 응용의 경우 수행할 코어가 정해져 있는 경우를 말한다. 예를 들어 임베디드 시스템에서 일반적인 경우 코어1, 2를 사용 중 실수 연산 등이 필요한 경우 VFP(: Vector Floating-Pointer)가 존재하는 프로세서 코어에 연산을 할당하여 시스템의 성능을 향상 시키는 경우이다.

2.2 데이터 공유 기법

공유 메모리는 물리적인 메모리의 특정 메모리 구역을 서로 다른 프로세스들이 데이터를 공유할 수 있는 영역으로 구성한다. 프로세스들은 이 공유 메모리 영역을 자신의 가상 메모리 영역에 부착하여 사용하게 된다. 공유 메모리는 프로세간 통신 기법 중 시스템 호출을 이용하지 않으므로 빠른 방법을 제시한다. 즉, 다른 기법들은 시스템 호출을 통해서 운영체제에 의해서 데이터 처리를 수행하게 되므로 공유 메모리 방법이 성능상 이점을 가지고 있다. 예를 들어 프로세스A에서 공유 메모리 함수를 이용하여 데이터를 복사하게 되면, 프로세서 B는 데이터 읽기 및 변경이 가능하다.

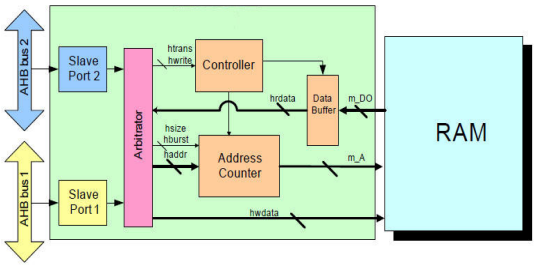

그림 1. 임베디드 메모리 컨트롤러 블록 다이어그램
Fig. 1 Embedded memoryc controller block diagram

그림 1은 EMC 블록 다이어그램을 나타낸다. 이 컨트롤러는 서로 다른 AHB를 연결할 수 있는 슬레이브 포트를 지원한다. AHB를 지원하므로 버스를 이용하고자 하는 자원에 대한 동시성을 제공해줄 뿐 아니라, 우선순위를 제공해 준다. 서로 다른 AHB에서 EMC에 연결 되어 있는 동일 메모리 주소를 접근하고자 하는 경우 데이터 동기화 문제점을 해결할 수 있다.

III. 공유 메모리 컨트롤러 디바이스 드라이버 설계

3.1 시스템 구조

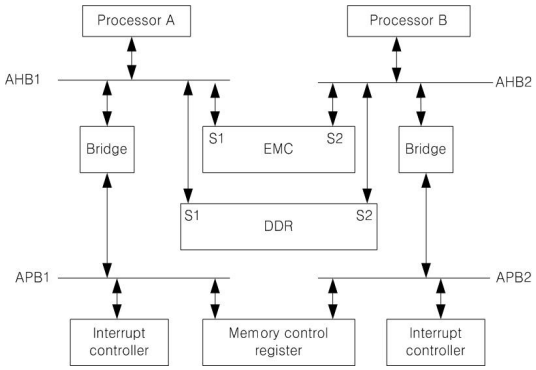

그림 2. 듀얼 코어 공유 메모리 시스템 구조도
Fig. 2 Dual core shared memory system architecture

그림 2는 듀얼 코어 공유 메모리 시스템 구조도를 나타낸다. 프로세서 코어마다 운영체제가 존재하는 AMP의 경우 일반적으로 동일한 공유 메모리 영역을 사용할 수 없다. 따라서 두 프로세서에서 공유 가능한 메모리 영역인 EMC 블록을 이용하여 메모리 접근이 가능하다. 이 컨트롤러에는 32KByte의 SRAM을 이용하며, 논리적으로 8개의 세그먼트로 분리 하였다. 그림 2의 Memory control register는 프로세서가 데이터를 공유하기 위해서 EMC에 데이터를 기록할 경우 이용 가능한 세그먼트 정보 확인 및 상대편 프로세서의 공유 메모리 컨트롤러에 인터럽트를 활성화 시키는 역할을 한다. 예를 들어 프로세서 A에서 새롭게 공유 메모리 구간을 생성하고자 하는 경우, 전체 공유 메모리 영역 중에서 이용 가능한 영역을 이 레지스터를 통해서 알 수 있다.

3.2 공유 메모리 데이터 읽기/쓰기

일반적으로 공유 메모리를 이용하는 프로세스는 자

신의 프로세스에 공유 메모리 세그먼트를 추가하여 덧붙여 사용하여 마치 자신의 프로세스 메모리처럼 이용이 가능하다. 본 논문에서는 기존의 운영체제에서 제공해주는 공유 메모리를 이용하는 방식이 아닌 프로세서 내부에 별도의 메모리 컨트롤러를 추가하는 방식을 이용하였다.

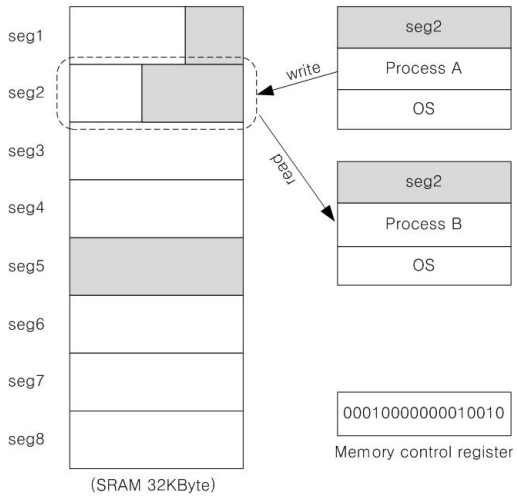

그림 3. 메모리 컨트롤러 데이터 공유 방법
Fig. 3 Method of data shared in memory controller

그림 3은 메모리 컨트롤러를 이용하여 프로세스에 공유 메모리 영역을 추가하는 방법을 나타낸다. 사용되는 메모리 컨트롤러의 SRAM 용량은 32KByte이며, 논리적으로 8개의 세그먼트로 분할하였다. 세그먼트에서 색상이 있는 부분이 현재 데이터가 기록되어 있는 부분을 나타낸다. 하나의 세그먼트당 4KByte 공간을 사용하며, 전체 용량을 모두 이용할 수도 있고 그렇지 않을 수도 있음을 알 수 있다. 그림 3의 오른쪽은 Process A에서 공유 메모리를 생성하여 데이터를 기록한 후, Process B의 프로세스가 공유 메모리를 통하여 데이터를 읽어가는 흐름을 나타낸다. 새로운 프로세스가 공유 메모리 세그먼트를 사용하고자 할 경우 현재 이용 가능한 세그먼트를 할당 받아야 할 것이다. 위의 설정 값들을 기록하는 부분이 그림 3의 Memory control register 항목이다. 이 레지스터는 4바이트 크기로 구성되어 있으며, 이용 중인 세그먼트 값들을 저장하는 용도로 사용된다.

3.3 공유 메모리 관리

사용자 프로세스에서 공유 메모리를 이용하여 데이터 읽기 및 쓰기를 수행하기 위해서는 공유 메모리의 세그먼트 아이디를 이용한다. 본 논문에서 설계한 공유 메모리 컨트롤러는 새로운 세그먼트를 할당 받게 되면 새로운 공유 메모리 세그먼트 아이디를 획득하게 된다. 현재 세그먼트가 8개 이므로 1부터 8까지의 아이디를 얻게 되며, 이를 관리하기 위한 공유 메모리 관리 테이블을 생성하였다.

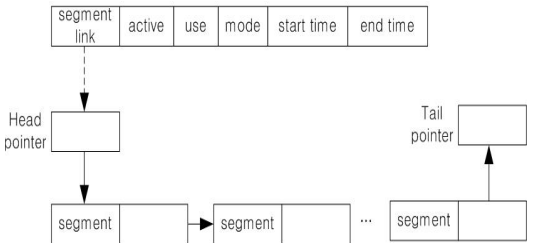

그림 4. 세그먼트 정보 저장 자료구조
Fig. 4 Data structure of saved segment information

그림 4는 공유 메모리 컨트롤러 세그먼트 정보를 저장하는 자료구조를 나타낸다. 한 프로세서가 새로운 공유 메모리 세그먼트를 할당 받고자 하는 경우 그림 3의 Memory control register를 이용하여 자료구조에 변경 사항을 기록한다. 예를 들어 다른 프로세스에 의해서 공유 메모리를 할당 하거나, 해제할 경우 이 레지스터에 기록 되므로 변경된 사항을 이용하여 위의 세그먼트 정보 저장 자료구조를 업데이트 한다. 그림 4의 노드는 세그먼트 저장 정보를 나타낸다. segment id는 1부터 8까지의 세그먼트 아이디, active는 현재 세그먼트 사용 여부를 의미한다. 그리고 use 항목은 현재 세그먼트 노드를 다른 프로세스에 의해서 사용중인지의 여부를 나타내며, mode는 해당 공유 메모리 세그먼트를 생성한 프로세서를 나타내는 것으로 0이면 프로세서 A를 의미하며, 1이면 프로세서 B를 의미한다. 만일 5개의 세그먼트를 할당 하였다면 노드의 수는 5가 될 것이다. 현재 테스트를 위해서 세그먼트를 8개, 메모리 크기를 각 4KByte로 할당 하였지만, 향후 유연성을 위해서 링크드 리스트를 이용하여 자료구조를 설계 하였다.

3.4 공유 메모리 컨트롤러

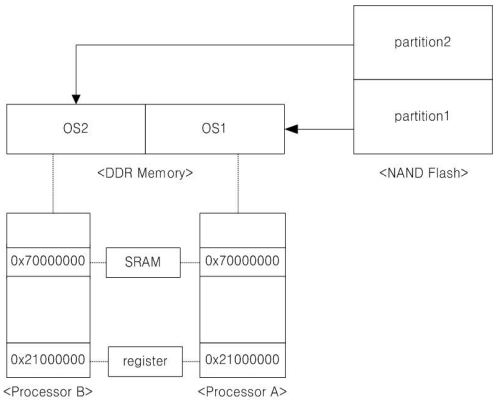

그림 5. 멀티 코어를 위한 임베디드 시스템 운영 방법
Fig. 5 Operating method of embedded system for multi core

그림 5는 프로세서 코어에 다른 운영체제가 탑재되는 시스템에서의 운영 방법을 나타낸다. 프로세서 코어마다 운영체제가 탑재되므로 플래시 메모리를 2개의 파티션으로 분할하여 첫 번째 파티션에는 프로세서 A에 대한 부트로더 및 운영체제 이미지가 저장되며, 두 번째 파티션에는 프로세서 B의 이미지가 저장된다. 사용된 DDR 메모리는 256MByte이며, 두 프로세서에 운영체제를 수행시키기 위해서 논리적으로 메모리 영역으로 절반씩 이용 하도록 하였다. 그리고 메모리 컨트롤러 주소는 0x70000000이며, register는 0x21000000를 사용한다. 즉, 0x70000000 부분이 SRAM의 시작 주소이며, 0x70007FFF가 종료 주소를 나타낸다.

운영체제에서 공유 메모리 컨트롤러를 이용하기 위해서는 물리적 메모리에 접근할 수 있는 커널 모듈 및 디바이스 드라이버가 필요하며, 본 논문에서는 디바이스 드라이버 형태로 설계하였다. 드라이버로 작성된 함수는 아래 표와 같다.

표 1. 드라이버 제공 함수
Table 1. Driver supported function

function	description
open	allocate segment number
read	get the data from memory controller
write	data store at memory controller
ioctl	set register of control register
close	segment recovery

표 1은 응용 프로그램에서 이용 가능한 공유 메모리 컨트롤러 시스템 호출 함수를 나타낸다. open 함수의 경우 이용 가능한 세그먼트를 검사한 후, 이용 가능한 세그먼트가 존재하는 경우 세그먼트 번호 즉, 1부터 8까지의 숫자 값을 리턴하며 그렇지 않는 경우 -1을 리턴한다. close 함수의 경우 사용한 세그먼트 번호를 입력하게 되면, 디바이스에서는 그림 4의 세그먼트 정보 저장 자료구조에서 해당 세그먼트를 초기화 시킨 후, 프로세서의 Memory control register의 해당 세그먼트를 초기화 및 EMC의 해당 SRAM 영역 또한 0으로 초기화를 수행한 후 응용 프로그램에 성공 여부를 리턴하게 된다. 그리고 read 함수는 메모리 컨트롤러부터 해당 세그먼트의 메모리를 읽어 들

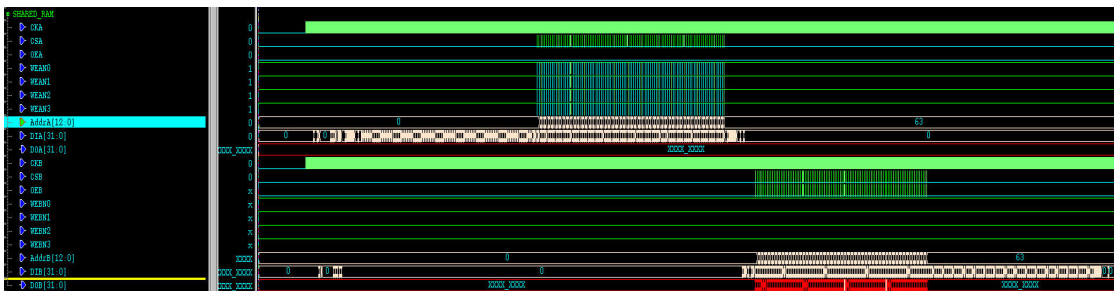

그림 6. 데이터 기록 시뮬레이션 결과
Fig. 6 Simulation result of data storage

이는 함수이며, write 함수는 메모리에 데이터를 기록하는 기능을 제공한다.

Ⅳ. 실험

설계된 공유 메모리 컨트롤러를 테스트하기 위해 각 프로세서에 임베디드 리눅스 운영체제를 포팅 하였으며, 컴파일러는 ARM gcc, 리눅스 커널 버전은 2.6.28, 파일 시스템은 속도가 빠른 cramfs를 이용하였다. 위의 그림 6은 공유 메모리 컨트롤러를 통하여 프로세서 사이의 데이터 공유를 나타낸다. 공유 메모리 컨트롤러는 AHB1, AHB2와 슬레이브 포트로 연결되어 있으므로, 프로세서A와 프로세서B에서 모두 사용이 가능함을 시뮬레이션 결과를 통하여 확인할 수 있다.

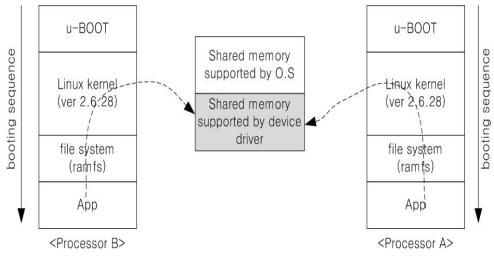

그림 7. 실험 시스템
Fig. 7 Experiment system

그림 7은 설계된 공유 메모리 컨트롤러 실험 시스템을 나타낸다. 시스템이 부팅하기 위해서 부트로더, 리눅스 커널, 파일 시스템, 마지막으로 응용 프로그램을 통해서 디바이스 드라이버 형태로 개발된 공유 메모리 영역으로부터 데이터를 읽어 오거나 저장할 수 있다. 테스트를 위하여 프로세서 A에서 데이터를 기록하며, 프로세서 B를 통하여 읽기를 수행하여 디바이스 드라이버로부터 공유 메모리 세그먼트를 정상적으로 할당 받으며, 프로세서 B를 통하여 공유 메모리로부터 데이터 읽기가 가능하다.

위의 그림 8은 공유 메모리 컨트롤러를 이용한 SRAM 읽기/쓰기를 나타낸다. write 옵션을 통하여 프로그램을 수행하게 되면, 새로운 공유 메모리 세그먼트를 할당 받게 된다. 따라서 open 시스템 콜을 통하여

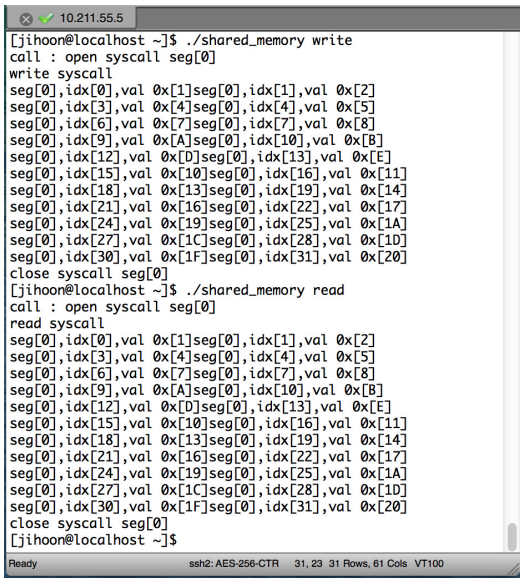

그림 8. 메모리 컨트롤러에 대한 데이터 읽기/쓰기
Fig. 8 Data read/write for memory controller

새로운 세그먼트 번호를 할당 받게 되는데, 현재 세그먼트가 비워 있으므로 첫 번째 요소를 할당받게 됨을 알 수 있다. 그리고 write 시스템 콜을 통하여 데이터를 메모리에 저장하게 된다. 프로그램 시작 시 read 옵션을 입력하는 경우, open 시스템 콜을 호출하지 않는다. 공유 메모리 컨트롤러 디바이스 드라이버는 open 시스템 콜을 통하여 새로운 세그먼트를 할당받기 때문이다. 만일 기존 세그먼트 아이디를 이용할 경우라면 open 시스템 콜 없이 read, write 시스템 콜의 첫 번째 인자인 fd에 세그먼트 아이디를 입력해 주면 해당 세그먼트에 데이터 읽기, 쓰기가 가능하게 된다.

Ⅴ. 결 론

본 논문에서는 서로 다른 운영체제가 탑재되는 멀티 코어 시스템에서 데이터 공유가 가능한 디바이스 드라이버를 설계 하였다. 운영체제가 제공해주는 공유 메모리를 이용하는 방식이 아니라 메모리 컨트롤러 블록을 ARM 프로세서의 AHB에 추가하는 방식을 사용 하였다. 기존 공유 메모리를 이용하는 방식이 아니므로, 임베디드 리눅스 운영체제에서 사용 가능한 디바이스 드라이버를 개발하였다. 드라이버에 메모리 세

그먼트 할당, 해제 및 데이터 기록 및 읽기가 가능함을 시험을 통해서 확인 하였다.

일반적인 공유 메모리의 경우 사용자 프로세스에 공유 메모리 세그먼트를 할당하며, 공유 메모리 관리는 운영체제에서 수행하게 된다. 또한 공유 메모리는 읽기/쓰기를 수행할 경우 시스템 콜을 이용하지 않으므로 성능상의 이점을 가질 수 있다. 반면 본 논문에서는 메모리 컨트롤러 블록에서 데이터 읽기/쓰기를 수행하여 프로세스의 DDR 메모리로 복사하여 수행하여야 하는 문제로 좋은 성능을 가져오지 못하는 문제점이 발생한다. 추가한 메모리 블록이 SRAM이므로 큰 메모리 용량을 이용하지 못하는 관계로 세그먼트 할당 및 메모리 크기가 제한되는 문제로 발생함을 확인 하였다.

References

[1] S.-K. Lee and W.-Y. Jeong, "Design of the Entropy Processor using the Memory Stream Allocation for the Image Processing," *J. of The Korea Institute of Electronic Communication Sciences*, vol. 7, no. 5, 2012, pp. 1017-1026.

[2] J.-H. Moon and J.-C. Oh, "Design of the SD Protocol Analyzer," *J. of The Korea Institute of Electronic Communication Sciences*, vol. 8, no. 11, 2013, pp. 1697-1706.

[3] C.-H. Yoon and G.-J. Kim, "Design of Embedded Platform based on Android," *J. of The Korea Institute of Electronic Communication Sciences*, vol. 8, no. 10, 2013, pp. 1545-1552.

[4] J.-S. Jeong, "Improvement Method and Performance Analysis of Shared Memory in Dual Core Embedded Linux System," Master's Thesis, *KongJu University*, Dec. 2010.

[5] J.-S. Jung, K.-Y. Lee, J.-K. Kim, and C.-B. Kim, "Performance Analysis on Dual-core Embedded System Using High Speed IPC Technique," *Proc. of the KICS*, 2008, pp. 1494-1497.

[6] S.-J. Jang, E.-S. Choi, D.-U. Kang, G.-Y. Lee, D.-H. Kim, and J.-M. Kim, "A Study of Performance Enhancement for the Shared Memory in the Linux O.S," *Proc. of the KIISE Korea Computer Congress 2007*, vol. 34, no. 2, 2007, pp. 324-329.

[7] H. Singh, R. Dhand, and S. Bassi, "Inter-process communication(IPC) : An interpretive conspectus," *IASTED Int. Conf. on Communications and Information Technology (CIIT)*, US Virgin Islands, Nov. 2002.

[8] D.-H. Lee, S.-K. Lee, S. Park, and S. Maeng, "Implementation and Performance Evaluation of Software Distributed Shared Memory for SMP Clusters," *J. of KIISE : Computer Systems and Theory*, vol. 30, no. 7, 2003, pp. 331-340.

저자 소개

문지훈(Ji-Hoon Moon)

2002년 동의대학교 컴퓨터공학과 졸업(공학사)
2004년 동의대학교 대학원 컴퓨터공학과 졸업(공학석사)
2009년 순천대학교 대학원 컴퓨터공학과 수료
2013년~현재 이니텍 보안개발2본부 DB보안팀
※ 관심분야 : 정보보안, 임베디드시스템

오재철(Jae-Chul Oh)

1978년 전북대학교 전기공학과 졸업(공학사)
1982년 전북대학교 대학원 컴퓨터공학과 졸업(공학석사)
1988년 전북대학교 대학원 컴퓨터공학과 졸업(공학박사)
1984년~1986년 기전대학교 전자계산학과 전임강사
1986년~현재 순천대학교 컴퓨터공학과 교수
※ 관심분야 : 임베디드시스템, USN, 네트워크 설계 및 분석