Dual Core 시스템에서 Shared Memory 기능 구현

Implementation of the Shared Memory in the Dual Core System

장승주
동의대학교 컴퓨터공학과

Seung-Ju Jang(sjjang@deu.ac.kr)

요약

Linux에서 사용되는 Shared Memory는 동일한 메모리 영역에 여러 개의 프로세스가 접근할 수 있도록 해 주는 기술이다. 본 논문은 Linux 운영체제에서 지원해 주는 System V의 IPC 중 하나인 Shared Memory를 Dual Core 시스템 상에서 동작하도록 구현한다. 본 논문에서는 커널 단계에서 처리되는 SVR(System V Release) 형식의 Shared Memory를 다룬다. 기존의 공유메모리 방식은 단일 처리기를 이용한 방식이다. 본 논문에서는 dual core를 이용하여 공유메모리 처리를 할 수 있는 시스템을 제안한다. 본 논문에서 제안하는 Dual Core 시스템에서 공유 메모리 기능 구현은 기존의 단일 처리기 시스템에서보다 성능을 향상시킬 수 있도록 한다. 공유 메모리를 이용한 프로세스의 동작이 별개의 CPU에서 동작되도록 함으로써 성능 향상을 꾀한다.

■ 중심어 : | 공유 메모리 | 듀얼 코어 시스템 |

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 which is SVR in a kernel step. We design a share memory facility of Linux operating system on the Dual Core System. In this paper the suggesting of share memory facility design plan in Dual Core system is enhance the performance in existing an unity processor system as a dual core practical use. We attemp a performance enhance in each CPU for each process which uses a share memory.

■ keyword : | Shared Memory | Dual Core System |

I. 서 론

Linux에서 사용되는 Shared Memory(공유메모리)는 동일한 메모리 영역에 여러 개의 프로세스가 접근할 수 있도록 해 주는 매력적인 기술이다. 본 논문에서는 SVR IPC에서 사용되는 기술 중 공유 메모리(Shared Memory)를 Dual Core 시스템 상에서 동작하도록 구현한다.

복잡한 프로그래밍 환경에서 다수의 프로세스들은 서로 협력하기 위하여 서로 통신하고 자원과 정보를 공유한다. 운영체제 커널에서는 이러한 기능을 제공하는데, 이를 프로세스간 통신(IPC) 이라 한다. 프로세스간

* 본 연구는 2007년도 한국전자통신연구원 연구과제로 수행되었습니다.

접수번호 : #080502-001
접수일자 : 2008년 05월02일
심사완료일 : 2008년 07월 31일
교신저자 : 장승주, e-mail : sjjang@deu.ac.kr

통신을 사용하는 목적은 데이터 전송, 데이터 공유, 자원 공유, 프로세스 제어를 하기 위해서이다.

Shared Memory 기법은 시스템의 가상 메모리(Virtual Memory) 구조에 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 논문에서는 커널 단계에서 처리되는 SVR(System V Release) 형식의 Shared Memory를 다룬다. 기존의 공유메모리 방식은 단일 처리기를 이용한 방식이다. 본 논문에서는 dual core를 이용하여 공유메모리 처리를 할 수 있는 시스템을 제안한다. 본 논문에서 제안하는 Dual Core 시스템에서 공유 메모리 기능 구현은 기존의 단일 처리기 시스템에서보다 성능을 향상시킬 수 있도록 한다. 공유 메모리를 이용한 프로세스의 동작이 별개의 CPU에서 동작되도록 함으로써 성능 향상을 꾀한다.

본 논문의 2장에서는 관련 연구에 대해서 언급하고, 3장에서 Dual Core 시스템에서 동작하는 Shared Memory 기능을 구현 방법에 대해서 설명한다. 4장에서 Dual Core 시스템에서 동작하는 Shared Memory 기능 구현을 테스트하는 테스트 프로그램에 대해서 설명한다. 5장에서 작성한 테스트 프로그램으로 테스트한 결과에 대해서 설명한다. 마지막으로 6장에서 결론을 맺는다.

II. 관련연구

유닉스 운영체제는 크게 두 개의 버전으로 되어있다. SVR은 초기 유닉스 버전들이 해당한다. BSD는 버클리 대학에서 만든 유닉스 버전이다. 초기 BSD는 대학 내에 설치된 상용 유닉스에 학생들이 필요한 소프트웨어를 만들며 시작된, 일종의 Free Software 모음이었다 [1-3].

BSD와 SVR 유닉스 버전은 서로 다른 길을 걷게 되었지만 나중엔 다시 합쳐져서 현재의 상용 유닉스들은 대부분 통합된 기능들을 제공하고 있다 [3][4-10].

유닉스 운영체제에서 Shared Memory 기능은 shmget( )으로 시작하는 SVR 형식의 Shared Memory와 shm_open( )으로 시작하는 POSIX 형식구성되어 있다 [5, 7, 9, 14]. 이 기능의 차이점은 SVR 형식은 커널 단계에서 구현되었고, POSIX 형식은 라이브러리로 구현되었다. SVR 버전은 대부분의 유닉스 커널과 리눅스에서 지원하며 커널에 포함되어 있기 때문에 커널 파라미터를 조정해서 기능을 변경하여 사용가능하다. 메모리 영역의 구분은 key와 id를 통해 다른 프로세스가 접근 할 수 있으며 메모리 영역별로 euid라든가 permission 같은 메타 정보 역시 따로 커널에서 관리하게 된다 [5-7].

POSIX 버전은 일종의 메모리 디스크 방식으로 디렉터리에 마운트 한 후, 마운트 된 곳에 파일을 만드는 형태로 접근할 수 있다. 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 [3-6].

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.

III. Dual Core 시스템에서 동작하는 Shared Memory 기능의 구현

공유메모리는 물리적인 메모리의 특정 영역에 대하여 프로세스들에 의해 공유되어지는 영역을 말한다. 이

공유메모리는 프로세스들이 동작하면서 공유하여 사용이 가능하다. [그림 1]은 공유메모리 사용에 대한 프로세스 동작의 일반적인 구조이다.

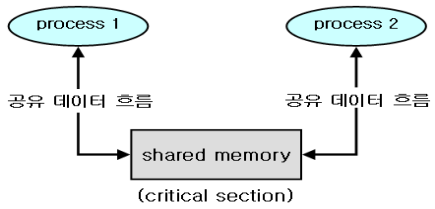

그림 1. 공유메모리 시스템 구조

공유메모리는 물리적인 메모리의 특정 영역에 대하여 프로세스들이 공유하여 사용이 가능하다. [그림 1]에서 공유메모리 영역은 Critical Section(임계 영역)이다. 즉, 여러 프로세스들이 이 구역에 동시에 접근하게 되면 데이터가 깨어지게 된다. 이러한 경우에 프로세스들이 공유메모리에 동시에 접근하지 못하도록 하는 방법을 제공해야 한다. 이러한 기능은 공유메모리 시스템에서 기본적으로 제공이 되고 있다. 이러한 단점에도 불구하고 공유메모리를 사용하는 이유는 데이터 공유를 쉽고 빠르게 할 수 있기 때문이다.

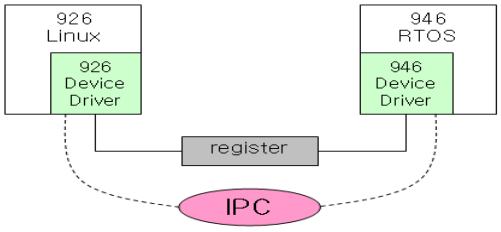

그림 2. Dual Core에서의 공유메모리 시스템 구조

[그림 2]는 Dual Core 시스템에서 공유메모리 시스템 구조를 나타낸다. 서로 다른 CPU에서 동작하는 프로세스들 간에 공유메모리를 통해서 데이터 교환이 가능하다. Dual core 시스템을 이용한 공유메모리는 단일 처리기를 사용하는 공유메모리에 비해서 성능 향상등의 장점을 얻을 수 있다. 프로세스의 작업량이 많을 경우에 단일 처리기 시스템에서는 수행에 시간이 많이 소요된다. 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 Dual Core 시스템을 사용하게 되면, 두 개의 프로세스를 동시에 서로 다른 CPU에서 수행할 수 있다. 이것은 시스템 성능 향상을 도모할 수 있다.

Dual Core 시스템에서 서로 다른 CPU에서 독립적으로 갖고 있는 Main Memory에서 할당되는 Shared Memory 정보를 저장하기 위해 [그림 3]과 같은 새로운 자료구조를 만들었다.

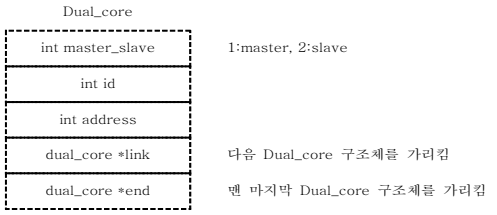

그림 3. 새로운 자료구조

[그림 3]은 Dual Core 시스템에서 공유메모리 정보를 저장하기 위한 자료구조이다. 자료구조에서 master 노드와 slave 노드를 구분할 수 있도록 master_slave, link, end 변수를 두었다.

master_slave 변수의 경우 이 값이 '1'이면 master 노드를 나타낸다. 즉 Linux 운영체제에서 생성되고 관리된다는 것이다. master_slave 변수 값이 '2'이면 slave 노드를 나타낸다. 프로세스가 RTEMS 운영체제에서 생성되고 관리된다는 것을 나타낸다.

[그림 4]는 공유메모리 정보를 저장하기 위한 자료구조와 그 자료구조를 활용하여 Dual Core에서 shmget( )을 사용할 수 있게 만든 sys_shmget( ) 함수이다. 본 논문에서 설계 및 구현된 함수이다.

이 함수에서 shmflg 변수 값으로 Dual Core에서 동

작할 응용프로그램인지를 구분가능하도록 되어있다. 만약 shmflg의 의미가 Dual Core 응용프로그램이라고 한다면, master 노드인지 slave 노드인지를 구분한다. master노드라면 원래 Linux의 sys_shmget( ) 함수 코드가 실행이 되고, slave노드라면 RTEMS의 shmget( )에 해당하는 함수가 수행하게 된다.

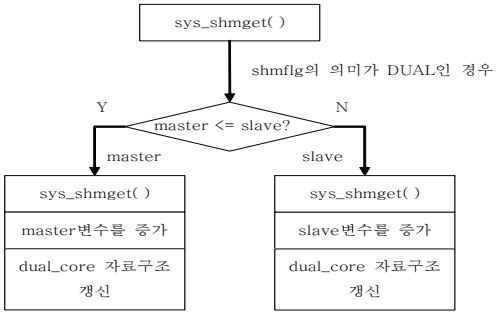

그림 4. 설계된 sys_shmget() 함수

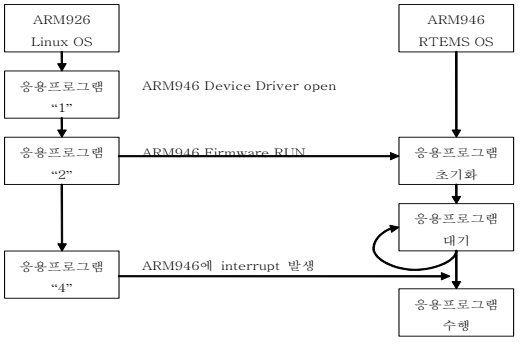

그림 5. 926 CPU와 946 CPU의 데이터 교신 과정

[그림 5]는 ARM926 CPU에서 동작하는 리눅스 운영체제와 ARM946 CPU에서 동작하는 RTEMS 운영체제가 공유메모리를 이용하여 데이터를 주고 받는 과정을 나타낸다. ARM926의 리눅스 운영체제에서 동작하는 응용프로그램에 의해 ARM926의 리눅스 운영체제와 ARM946의 RTEMS 운영체제가 데이터 송수신이 가능하다.

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. 커멘드 "3"은 ARM946의 Firmware를 종료시키는 기능을 갖고 있다. 커멘드 "4"는 ARM926의 리눅스 운영체제 커널 단계 프로그램을 실행시켜 ARM946의 RTEMS운영체제에서 동작하는 프로그램에 interrupt를 발생시킨다. 커멘드 "H"는 제공하는 커멘드 메뉴를 보여준다. 이러한 명령어들을 이용하여 두 운영체제에서 동작하는 프로세스들이 데이터 송수신이 가능하다.

Ⅳ. 테스트 프로그램 작성

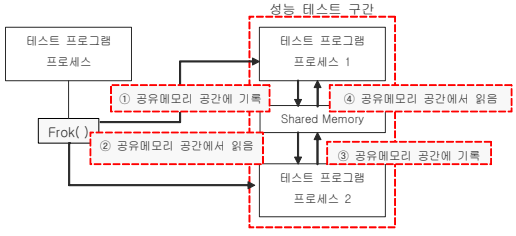

그림 6. Single Core에서 테스트 프로그램 구조

[그림 6]은 Single Core에서 테스트 프로그램의 구조를 나타낸다. Fork( ) 시스템 콜 함수로 나누어진 두 개의 프로세스가 Shared memory 영역을 서로 읽고/쓰기를 반복한다. "프로세스 1(부모)"에서 공유메모리 공간에 값을 기록한다. "프로세스 2(자식)"는 공유메모리공간에서 "프로세스 1"이 기록한 값을 읽어 온다. "프로세스 2"는 읽어온 값을 변경하여 공유메모리 공간에 값을 다시 기록한다. "프로세스 1"은 "프로세스 2"가 공유메모리공간에 기록한변경된 값을 읽어 온다.

테스트 프로그램의 전체적인 동작 순서는 먼저 Shared Memory를 생성하고, 성능 측정을 위한 시간을 설정한다. 시간 단위는 기본 설정이 초로 되어 있다. Fork( ) 시스템 콜 함수로 "프로세스 2"를 생성한다. "프로세스 1"에서 Shared Memory에 데이터를 기록(초기화)한다. "프로세스 2"에서 Shared Memory의 데이터를 읽어와 가공(덧셈 연산) 후 결과를 Shared

Memory에 기록한다. "프로세스 1"은 기록된 결과를 읽어온다.

```
int main()
{
        start = time(0);
        //  ========================
        //  Shared Memory 관련 소스 코드
        //  ========================
        pid = fork();
        if (pid == 0)         {
                자식 프로세스의 작업 코드
        }
        else if(pid > 0)        {
                부모 프로세스의 작업 코드
        }
        printf("Execution Time : %.3f \n",
                        difftime(time(0),start));
        return 0;
}
```

알고리즘 1. Single Core에서 공유 메모리 테스트 소스 코드

[알고리즘 1]은 Single Core에서 동작하는 Shared Memory 성능을 테스트하는 프로그램이다. 변수의 선언이 끝나면 시간함수를 이용하여 현재의 시간을 기록한다. 시간 측정에 포함된 부분은 Shared Memory 영역을 할당하는 부분부터 Fork( ) 시스템 콜 함수, 부모/자식 프로세스에서Shared Memory Data 읽고/쓰기 부분까지 이다.

시간 측정은 변수 선언이 끝난 시점부터 프로그램이 종료되기 전까지 이며, difftime( ) 함수를 사용하여 시작시간과 종료시간의 차이 값을 프로그램의 총 소요시간으로 취하고 있다.

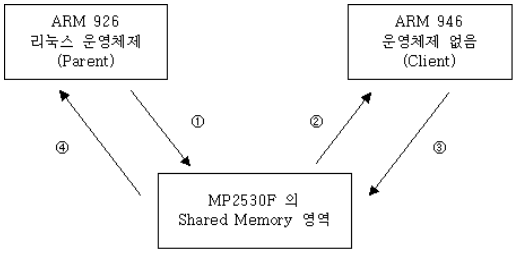

그림 7. Dual Core에서 테스트 프로그램 구조

[그림 7]은 Dual Core에서 테스트 프로그램의 구조를 나타낸다. 물리적으로 나누어진 두 개의 CPU에서 동작하는 프로그램이 Shared memory 영역을 서로 읽고/쓰기를 반복한다.

[그림 7]의 ①은 Shared Memory 영역에 A926이 데이터를 쓴다. 기록된 데이터는 A946에서 읽어 들인다(②). 읽은 내용에 잘 읽었다는 흔적을 표시하여 다시 Shared Memory 영역에 A946이 데이터를 쓴다(③). 기록된 데이터를 A926이 읽어 최종적으로 사용자에게 정상적으로 데이터의 공유가 이루어 졌는지를 출력하여 보여주게 된다(④).

```
do_gettimeofday(start_tmp);
shared_memory = sys_shmat(shmid, (void *)0);
writevalue = (U32)shared_memory;
...
g_pCDualCPU->WriteDataToA946(g_pCDualCPU ,i ,writevalue );
...
printk( "Execution Time : %ld, start : %ld, end : %ld.. \n",
          time_compare_sec(end_tmp, start_tmp),
          start.tv_sec, end.tv_sec);
```

알고리즘 2. ARM926에서 공유 메모리 주소를 ARM946으로 전달하는 소스

[알고리즘 2]는 리눅스커널 레벨에서 동작하는 프로그램 소스 코드이다. ARM926의 리눅스 운영체제와 ARM946의 프로그램이 통신을 이룬다. ARM926의 리눅스 운영체제의 커널 부분에서 ARM946의 RTEMS 응용프로그램과 통신을 이루게 된다. RTEMS 운영체제에서 공유 메모리 방식은 인터럽트 매카니즘에 의해서 동작하고 있다. ARM946의 리눅스 커널 부분에서 ARM946의 RTEMS 응용 프로그램으로 인터럽트 신호를 보내게 되면 RTEMS 응용 프로그램이 인터럽트 신호를 받아 프로그램이 동작하게 되어 있다.

[알고리즘 3]은 ARM946에서 동작하는 프로그램으로 [알고리즘 2]와 연동되어 동작한다.

```
...
value = g_pCDualCPU->ReadDataFromA926( index );
...
g_pCDualCPU->SetInterruptEnable( 0, CTRUE );
g_pCDualCPU->WriteDataToA926( i, value );
g_pCDualCPU->SetInterruptEnable( 0, CFALSE );
...
```

알고리즘 3. ARM946에서 공유 메모리 주소를 받고 다시 ARM926으로 전달하는 소스

V. 실험 및 결과

본 논문에서 Single Core와 Dual Core에서 Shared Memory의 성능을 측정하기 위해서 Dual Core 시스템을 사용하였다.

표 1. 성능 측정에 사용된 시스템 환경

CPU	Intel® Core™ 2 CPU 6400	2.13GHz, 2.13GHz
RAM	1 GB	DDR2 512MB × 2
HDD (총 할당 공간)	30 GB	SATA2 방식
linux Version	Red Hat 9.0 (2.4.20-8smp)	kernel 2.4.20-8
gcc Version	3.2.2	Red Hat linux 3.2.2-5

[표 1]은 Shared Memory의 성능을 측정한 시스템의 환경을 나타낸다. 듀얼 코어로 사용된 시스템은 Red Hat Linux 9.0을 기반으로 하고 있다.

듀얼 코어를 지원하기 위한 smp(Symmetric Multi-Processing) 기능을 갖춘 커널을 사용하였다. 듀얼 코어는 물리적으로 두 개인 프로세서 코어를 하나로 통합, 집적화한 것으로 겉보기에는 하나의 중앙 처리 장치(CPU)로 보이지만, 실제로는 두 개에 해당하는 CPU를 단 것처럼 강력한 연산 능력을 보이는 것으로 향후 프로세서 로드맵에 있어 기반 기술로 기대하고 있다. 듀얼 코어 방식은 한국 IBM사의 파워5, 한국 선 마이크로시스템스사의 스팍4, 한국 휴렛팩커드(HP)사의 PA 8800이 채택하고 있다. 듀얼 코어 프로세서는 물리적으로는 두 개이지만 용도에 따라서는 하나의 프로세서로 볼 수도 있어 프로세서 당으로 적용하는 소프트웨어 라이선스 정책에 혼선으로 줄 수도 있다.

[표 2]는 Single Core에서 Shared Memory Test 프로그램을 수행하여 얻은 결과이다. 위 결과들은 동일한 루프 횟수의 Test를 3회 수행하여 얻은 평균 값을 Test 결과로 나타내었다.

표 2. Single Core에서 공유 메모리 성능 테스트 결과

루프 횟수	1회	2회	3회	평균
100회	100	99	100	99.6
1000회	1009	1008	1008	1008.3
2000회	2018	2019	2019	2018.6
3000회	3029	3030	3029	3029.3
4000회	4038	4039	4038	4038.3

표 3. Dual Core에서 공유 메모리 성능 테스트 결과

루프 횟수	1회	2회	3회	평균
100	7	7	7	7
1000	69	69	69	69
2000	137	137	137	137
3000	203	203	203	203
4000	274	274	274	274

[표 3]은 Dual Core에서 Shared Memory Test 프로그램을 수행하여 얻은 결과이다. 위 결과들은 동일한 루프 횟수의 Test를 3회 수행하여 얻은 평균 값을 Test 결과로 나타내었다.

표 4. 공유메모리 성능 테스트 결과

루프 횟수(회)	1회	2회	3회	Single/Dual
100	100	99	100	Single
100	7	7	7	Dual
1000	1009	1008	1008	Single
1000	69	69	69	Dual
2000	2018	2019	2019	Single
2000	137	137	137	Dual
3000	3029	3030	3029	Single
3000	203	203	203	Dual
4000	4038	4039	4038	Single
4000	274	274	274	Dual

[표 4]는 Single과 Dual Core에서 수행된 공유메모리 성능 테스트 결과를 나타낸다. 본 논문의 4장에서 구현한 알고리즘들을 컴파일하여 생성된 바이러니 파일로 실험한 결과이다. [표 4]의 결과를 보면 Dual Core 시스템이 Single Core 시스템 보다 우수함을 알 수 있다. 이처럼 성능이 우수한 이유는 single core의 경우는 공유메모리의 사용에 대한 부담이 dual core보다 가중되기 때문으로 분석된다.

VI. 결 론

본 논문에서 제안하고자 하는 IPC 메카니즘 중 Shared Memory의 기능은 dual core 시스템에서 중요한 해결 과제이다. 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 동작되

도록 함으로써 성능 향상을 꾀한다. 본 논문은 대부분의 유닉스 시스템에서 사용되는 IPC 메카니즘 중 Shared memory의 기능을 Dual Core 시스템에서 동작할 수 있도록 설계해 보았다.

본 논문은 듀얼 코어 시스템인 임베디드 개발 시스템에서 ARM926에 리눅스 운영체제를 설치하고, ARM946에 RTEMS 운영체제를 설치하였다. 이 환경에서 두개의 서로 다른 운영체제 간에 공유메모리 환경을 사용 가능하도록 설계 및 구현하였다. ARM946에 설치된 RTEMS 운영체제 상에서 공유메모리를 구현하기 위하여 인터럽트 매카니즘을 이용하였다. 본 논문에서 제안한 내용을 구현하여 실험한 결과를 보면 Dual Core 시스템이 Single Core 시스템의 공유 메모리 처리 성능이 뛰어 남을 알 수 있다.

참 고 문 헌

[1] http://eminency.egloos.com/

[2] http://faqs.org/docs/kernel_2_4/

[3] Uresh Vahalia, *UNIX Internals*, 홍릉과학출판사, 2001.

[4] 한동훈, *시스템V IPC - 공유메모리,* 1997.

[5]http://virtual140.emde.inha.ac.kr/khdpmain/read.php3?table=tech&no=101&page=1

[6] 한동훈, *공유메모리 vs 세마포어를 이용한 chat program,* 1997.

[7] W. Sean, Share application data with UNIX System V IPC mechanisms, IBM, 2007.

[8] 박찬모, “분산 공유 메모리 시스템 설계에 관한 연구”, 조선대학교 동력자원연구소 동력자원연구소지, 제19권, 제1호, pp.129-143, 1997(5).

[9] 이병관, “분산 공유 메모리에서 일관성 제어 프로토콜”, 관동대학교 부설 산업기술개발연구소 산업기술논문집 제12호, pp.69-78, 1997(10).

[10] 이상권, “KDSM(DAIST Distributed Shared Memory) 시스템의 설계 및 구현”, 한국정보과학회논문지:시스템및이론, 제29호, pp.257-264, 2002.

[11] 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.

[12] 조주현, “임베디드 실시간 시스템의 개발환경”, 한국정보처리학회지, 제9권, 제1호, pp.120-126, 2002.

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

[14] H. Craig, Embedded Linus Hardware, Software and Installing, Pearson Education, 2002.

저 자 소 개

장 승 주(Seung-Ju Jang) 정회원

- 1985년 2월 : 부산대학교 계산통계학(전산학) 학사
- 1991년 2월 : 부산대학교 계산통계학(전산학) 석사
- 1996년 2월 : 부산대학교 컴퓨터공학과 박사
- 1987년 ~ 1996년 : 한국전자통신연구원 시스템 S/W 연구실
- 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