듀얼코어 임베디드 리눅스 시스템에서 코어간 세마포어 인터페이스 기능 설계

정지성*, 이재기*
*한국전자통신연구원
e-mail : jungjs@etri.re.kr, jklee@etri.re.kr

Design for the Semaphore Interface Function of the Dual Core Embedded Linux System

Ji-Sung Jung*, Jae-Gi Lee*
* Electronics and Telecommunications Research Institute

요 약

세마포어 처리문제는 프로세스간 메시지를 전송하거나 공유메모리를 통해 특정 데이터를 공유할 때 발생되는 문제로 공유된 자원에 여러 개의 프로세스가 동시에 접근하면 안되며, 단지 한번에 하나의 프로세스만 접근 가능하도록 하여야 한다. 세마포어 기본정책은 호출되는 코어에 생성되며, 다른 코어에서는 IPI 를 통해 존재 여부를 확인한다. 동일 코어에서 접근 시 기존방식으로 사용한다. 본 논문에서는 서로 다른 코어 및 운영체제에서 다른 코어에서 접근할 때에는 IPI 를 통해 존재 여부를 확인한 후 더미 세마포어 구조체를 생성하여 관련 정보를 유지하고 해당 요청을 처리해 주는 세마포어 인터페이스 기능 설계 방법을 제안한다. 제안하는 세마포어 인페이스 기능 설계 방안은 멀티 태스킹 기술 구현으로 기존 코어가 가지고 있는 성능상의 문제를 해결해 준다.

1. 서론

복잡한 프로그래밍 환경에서 다수의 프로세스들은 서로 협력하기 위하여 상호 통신하고 자원과 정보를 공유한다. 커널(Kernel)에서는 이것이 가능한 방법을 제공하는데 이를 프로세스간 통신(IPC: Inter-Process Communication)라 부른다. IPC 는 프로그램들간의 데이터를 공유하고 동기화하기 위해 사용되는 방법을 말한다.[1]

IPC 를 처리할 수 있게 해주는 일반적인 도구는 세마포어(Semaphore), 공유 메모리(Shared memory), 메시지 큐(Massage queue) 등이 있다. 이러한 방법들이 갖는 이점은 디스크를 통한 파일공유와 관련 I/O 를 통해 공유하는 방법으로 오버헤드를 줄일 수 있다는 것이다.[2] 프로세스간 통신을 하는 목적은 데이터 공유, 데이터 전송, 자원 공유, 사건 전송, 프로세스 제어를 하기 위함이다.

세마포어는 IPC 설비중의 하나로 분류되긴 하지만, 다른 파이프, 메시지큐, FIFO(First In First Out) 등과는 조금 다르며, 다른 IPC 설비들이 대부분 프로세스간 메시지 전송을 목적으로 하는데 반해서 세마포어는 프로세스간 데이터를 동기화하고 보호하는데 목적이 있다.[3]

프로세스간 메시지를 전송하거나 혹은 공유메모리를 통해서 특정 데이터를 공유하게 될 경우 (무슨 문제가)발생함으로 공유된 자원에 여러 개의 프로세스가 동시에 접근을 하면 안되며, 단지 한번에 하나의 프로세스만 접근 가능하도록 만들어 주어야 한다.[4]

본 논문에서는 서로 다른 코어 및 운영체제를 갖는 듀얼코어 임베디드 리눅스 시스템에서 세마포어 인터페이스 기능 설계 방법을 제시한다.

2. 듀얼코어에서 세마포어 인터페이스 기능 설계

다른 코어상에서 동작하는 프로세스 사이의 통신이 이루어지기 위해서는 세마포어와 같은 동기화 기법이 요구된다. 이에 대해 두 코어의 프로세스 사이에 사용 가능한 세마포어 기법이 필요하다. 기본정책은 세마포어는 호출되는 코어에 생성되며, 다른 코어에서는 IPI(Inter-Process-Interface)를 통해 존재 여부를 확인한다. 동일 코어에서 접근 시 기존방식으로 사용되지만 다른 코어에서 접근될 때에는 IPI 를 통해 존재 여부를 확인한 후 더미 세마포어 구조체를 생성하여 관련 정보를 유지하고 해당 요청을 처리한다. 다른 코어의 세마포어를 사용하기 위해서 sem_open 함수를 호출할 때 IPI 통해 전달된 세마포어 정보를 바탕으로 더미 자료구조를 생성하고 sem_wait 과 sem_post 등의 연산을 통하여 처리하도록 하는 것이다.

POSIX 세마포어는 기본적으로 sem_open, sem_wait, sem_trywait, sem_post, sem_close 등의 함수를 명시한다. 본 논문에서는 듀얼코어 MMSP 2+ 프로세서 환경에서 각 함수가 어떻게 구현되었는지를 설명한다.

2.1 sem_open

sem_open 은 매개변수로 전달되는 이름의 세마포어를 오픈 한다. 우선 926 코어의 프로세서가 946 코어

의 세마포어를 오픈하는 경우이다. 리눅스의 POSIX (Portable Operating System Interface) 세마포어는 glibc 에서 구현되며 커널의 futex 를 이용한다. Sem_open 은 _file_name_lookup 을 통해 해당 세마포어의 존재 여부를 확인하는데, 만약 존재하지 않는다는 결과를 반환하면 IPI 를 통해 946 코어에게 해당 이름의 세마포어 검색을 요청한다.

946 코어의 RTEMS(Rea-time executive for multiprocessor system)는 IPI 를 통해 전달된 이름의 세마포어 존재 여부 확인과 sem_t 타임의 세마포어 ID 를 얻기 위해 _POSIX_Semaphore_Name_to_id 함수를 호출한다. 그리고 _POSIX_Semaphore_Get 함수를 호출하여 대당 세마포어에 접근한다.

POSIX_Semaphore_Get 은 다시 Object_Get 을 호출하는데, 이는 세마포어 ID 로부터 POSIX _Semaphore_Control 을 얻는다. 여기서 세마포어 ID 는 세마포어가 저장되어 있는 테이블의 인덱스를 의미하며, 세마포어 테이블의 참조는 _POSIX_Se maphore_Info rmation 의 local_table 을 통해 얻는다.

세마포어의 카운터는 POSIX_Semaphore_ Control.semapho re.count 이며 IPI 를 통해 카운터의 주소와 세마포어 ID 를 926 코어의 리눅스에게 전달한다.

RTEMS 와 달리 리눅스는 sem_open 을 통해 반환되는 sem_t 타임의 데이터가 카운터 자체이다. 따라서 946 코어로부터 받은 세마포어 카운터의 주소를 타입 변환 연산 후 반환하면 된다. 하지만 추후 sem_post 연산에서 RTEMS 에게 대기중인 스레드를 깨우게 할 때 RTEMS 에서 사용되는 세마포어 ID 를 전달해야 하므로 futex_q 에 필드를 추가하여 이 값을 기록한다.

946 코어에서 926 코어의 세마포어를 오픈하는 경우에는 sem_open 에서 _POSI X_Semaphore_Name_to_id 가 해당 이름의 세마포어가 존재하지 않는다는 결과를 반환하면 IPI 를 통해 926 의 리눅스에게 해당 이름의 세마포어 정보를 요청한다. 926 코어의 리눅스는 전달받은 이름에 해당하는 세마포어 ID, 즉 카운터를 찾아 주소를 946 코어의 RTEMS 에게 전달한다.

RTEMS 는 926 으로부터 받은 세마포어의 메모리 주소를 이용하여 더미 posix_sema phore_control 구조체를 생성한다. 이때 리눅스로부터 받은 세마포어를 매핑시키기 위해 POSIX_Semaphore _Control.semaphore.count 필드에 카운터의 주소를 할당한다. 이때 기존의 카운터 필드는 포인터 타입이 아니므로 구조체 변경이 요구된다.

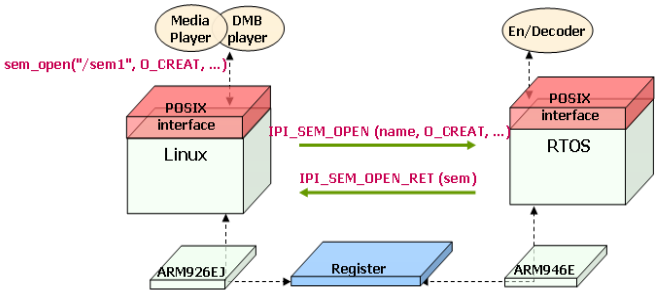


그림 1. 코어간 sem_open 상태도

<표 2> 코어간 sem_open 상태표

Called at	Exist at 926	Exist at 946	Mechanism
926 (Linux)	0	X	기존과 동일
	X	0	IPI_SEM_OPEN, IPI_SEM_OPEN _RET 통해 946 으로 부터 관련 정보(sem_t *) 받음. VFS futex fs 에 더미 file 생성
	X	X	기존과 동일 (926 에 생성)
946 (RTOS)	0	X	IPI_SEM_OPEN, IPI_SEM_OPEN_RET 통해 926 으로 부터 관련정보 (sem_t *) 받음. 더미 file 생성
	X	0	기존과 동일
	X	X	기존과 동일 (946 에 생성)

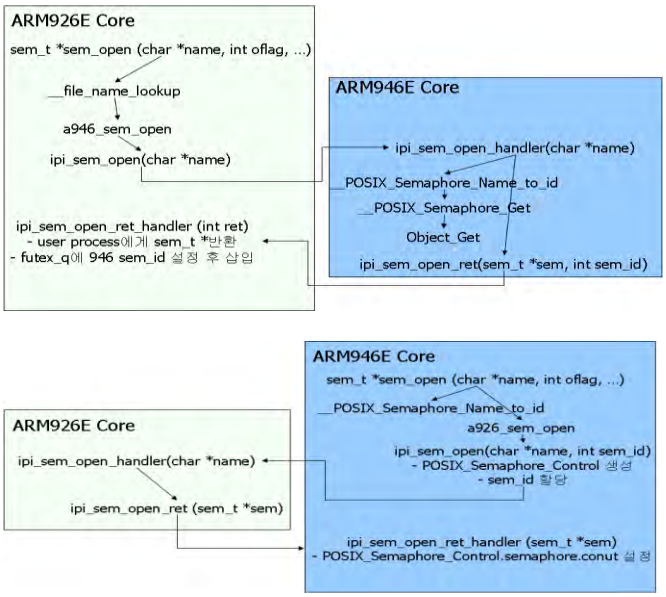


그림 2. 코어간 sem_open 함수 처리 과정

2.2 sem_wait

sem_open 과정에서 생성된 더미 세마포어 자료구조를 이용하여 다른 코어의 세마포어에 접근하는 경우에도 기존기법이 사용된다. 우선 926 코어의 리눅스에서 946 코어의 세마포어에 대해 sem_wait 을 호출하는 경우이다.

리눅스의 유저레벨 프로세스가 sem_wait 을 호출하게 되면 glibc 는 해당 세마포어의 가용 자원을 검사한다. 이때 926 코어에 존재하는 세마포어에 대해 메모리 주소를 통해 접근하여 가용 여부를 판단할 수 있으며, 동기화를 위해 atomic_decr ement_if_positive 함수를 사용한다.

자원이 가용하지 않아 프로세스가 대기해야 하는 경우에는 futex 시스템 콜을 호출한다. 이때 FUTEX _WAIT 이 매개변수로 사용되어 Kernel 내부에서는 futex_wait 함수가 호출된다.

Futex_wait 은 프로세스를 해당 futex 의 대기 큐에 삽입하고 대기상태로 변경시키는 함수로 add_wait _queue 함수를 통해 해당 프로세스를 futex_q 에 연결하고 __queue_me 함수를 호출하여 futex_q 를 해시 테이블에 삽입한다. 946 코어의 RTEMS 의 경우 POSIX _Semaphore_Get 함수를 통해 POSIX_Semaphore_Control 을 가져오고 _CORE_semaphore_Seize 함수를 이용

하여 통해 자원의 가용 여부를 확인한다.

기존의 _CORE_semaphore_Seize 함수는 카운터의 값을 검사하기 전 _ISR_Disable 함수를 호출하여 946 코어 상에서의 인터럽트 방생과 선점 되는 것을 방지한다. 하지만 이러한 방식으론 926 코어에서 동일 세마포어에 접근하는 것을 막을 수 없으므로 원자화된 연산으로의 변경이 필요하다.

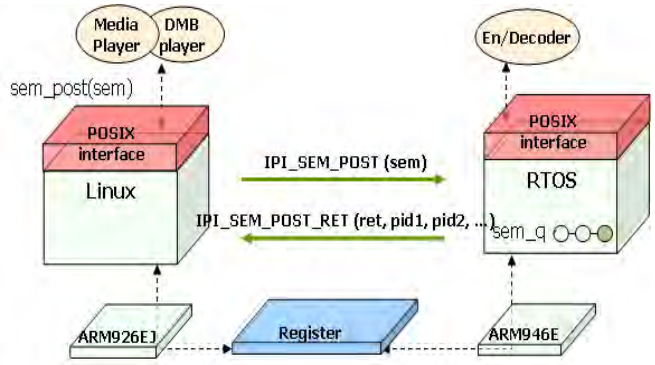
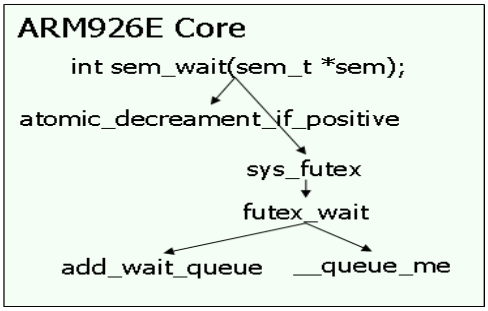


그림 3. 코어간 sem_wait 상태도

<표 2> 코어간 sem_wait 상태표

Called at	Target core		Mechanism
926 (Linux)	926 (Linux)		기존과 동일
926 (Linux)	946 (RTOS)	Value > 0	IPI_SEM_WAIT 통해 semaphore 정보 와 해당 pid 전달. 946 측 sem_q 에 더미 PCB 생성하여 삽입. IPI_SEM_ WAIT_RET 통해 926 의 해당 태스크 sleep 시킴 (이 때 926 에 더미 sem_q (wait_queue) 필요할 듯)
		Value <= 0	Value 값 감소시킴. IPI_SEM_WAIT_RET 통해 926 의 해당 태스크에게 반환 값 전달
946 (RTOS)	946 (RTOS)		기존과 동일
946 (RTOS)	926 (Linux)	Value > 0	IPI_SEM_WAIT 통해 semaphore 정보 와 해당 pid 전달. 926 측 sem_q 에 더미 PCB 생성하여 삽입. IPI_SEM_WAIT_RET 통해 946 의 해당 태스크 sleep 시킴 (이 때 946 에 더미 sem_q 필요할 듯)
		Value <= 0	Value 값 감소시킴 IPI_SEM_WAIT_RET 통해 946 의 해당 태스크에게 반환 값 전달



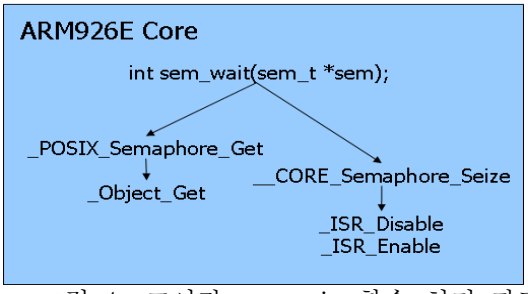


그림 4. 코어간 sem_wait 함수 처리 과정

2.3 sem_post

sem_wait 과 같이 기존의 방법이 사용되며 카운트 값의 변경 시 두 코어 모두에 대한 Locking 이 필요하다. Sem_post 의 경우 세마포어의 카운터의 값을 감소 시킨 후 두 코어의 대기 큐에 존재하는 프로세스들을 깨워야 한다. 이 때 대기 큐가 두 코어 모두에 존재할 수 있으므로 다른 코어에게도 자원이 가용하다는 것을 IPI 통해 알려야 한다.

또한 두 개의 대기 큐 중 어떤 것에 우선권을 주어야 하는지도 고려 대상이 될 수 있다. 926 코어의 리눅스에서 sem_post 가 발생한 경우 atomic_increment _val 함수를 이용하여 카운터의 값을 증가시키고 해시 테이블로부터 해당 futex_q 를 검색하여 대기 큐의 프로세스들을 깨운다. 그리고 946 코어의 RTEMS 에게 IPI 를 발생시키고 futex_q 에 저장된 RTEMS 에서 사용되는 세마포어 ID 를 전달한다.

RTEMS 는 전달받은 세마포어 ID 를 매개변수로 하여 sem_post 를 호출하게 되는데, 이를 통해 결과적으로 _CORE_semaphore_Sur render 가 호출된다. 이 함수는 _Thread_queue_Dequeue 를 통해 대기중인 Tread 를 오픈한다. 이 때 기존의 소스에서는 세마포어 카운터를 증가시키지만 이미 리눅스에 의해 카운터가 증가되었으므로 이 작업은 불필요하다.

946 코어에서 sem_pos 가 발생한 경우 _CORE_semaphore_Surrender 가 호출되어 _Thread_queue_Dequeue 를 통해 대기중이 Thread 를 오픈하고 세마포어 카운터를 증가시킨다. 이어서 IPI 를 통해 926 코어의 리눅스에게 세마포어 ID 를 전달한다.

리눅스는 전달받은 세마포어 ID 를 바탕으로 해당 해시 테이블에서 futex_q 를 검색하고 대기 중인 프로세스들을 오픈하게 된다.

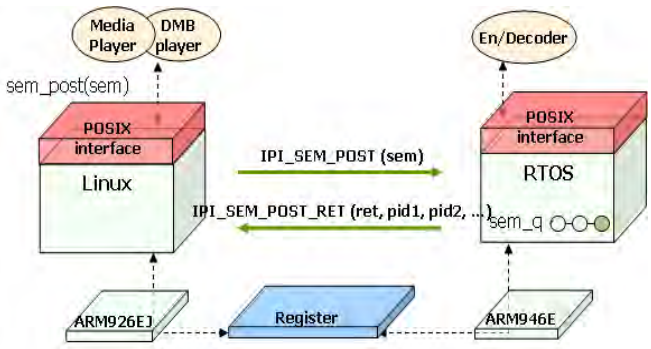


그림 5. 코어간 sem_post 상태도

<표 3> 코어간 sem_post 상태표

Called at	Taget core	Mechanism
926 (Linux)	926 (Linux)	대기중인 태스크들 깨움. 946 의 태스크가 있을 경우 IPI_SEM _WAKE 통해 946 에게 해당 pid 전달 하여 해당 태스크 깨움 (946 의 더미 PCB 삭제)
926 (Linux)	946 (RTOS)	IPI_SEM_POST 통해 946 이 대기중인 태스크 들 깨우게 함.926 태스크가 있을 경우 IPI_SEM_PO ST_RET 통해 926 에게 반환값과 해당 pid 를 전달하고 해당 태스크 깨움 (926 의 더미 PCB 삭제)
946 (RTOS)	946 (RTOS)	대기중인 태스크들 깨움. 926 의 태스크가 있을 경우 IPI_SEM_WAKE 통해 통해 926 에게 해당 pid 전달 하여 해당 태스크 깨움 (926 의 더미 PCB 삭제)
946 (RTOS)	926 (Linux)	IPI_SEM_POST 통해 926 이 대기중인 테스크 들 깨우게 함. 946 태스크가 있을 경우 IP I_SEM_POST_RET 통해 926 에게 반환값과 해당 pid 전달하고 해당 태스크 깨움 (926 의 더미 PCB 삭제)

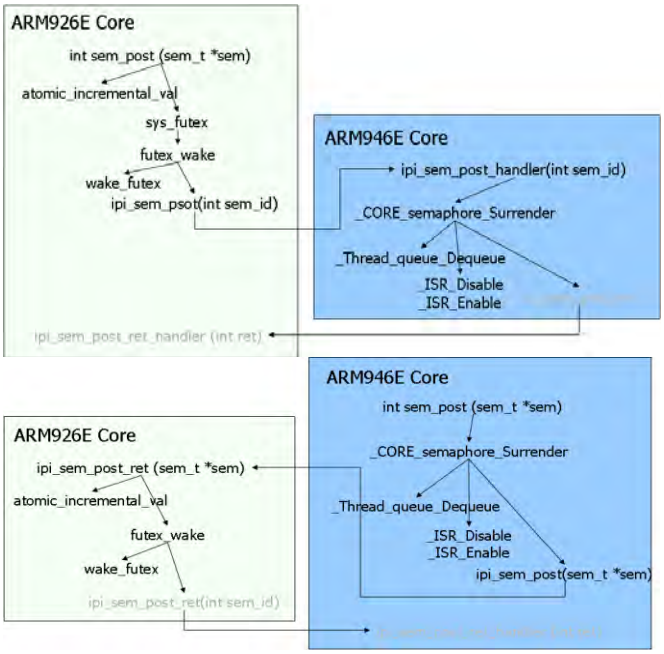


그림 6. 코어간 sem_post 함수 처리 과정

3. 결론

본 논문에서는 듀얼코어 임베디드 리눅스 시스템 환경에서 코어간 통신을 위한 IPC 기술인 세마포어, 메시지큐, 공유메모리에서 세마포어 인터페이스 기능 설계 방법을 제시하였다.

OS 환경 및 서로 다른 코어에서 접근될 때에는 IPI 를 통해 존재 여부를 확인한 후 더미 세마포어 구조체를 생성하여 관련 정보를 유지하고 해당 요청을 처리한다. 다른 코어의 세마포어를 사용하기 위해 sem_open 함수를 호출할 때 IPI 통해 전달된 세마포어 정보를 바탕으로 더미 자료구조를 생성하고 sem_wait 과 sem_post 등의 연산 등은 이를 통하여 처리하도록 설계 되어야 한다.

참고문헌

[1] 한동훈, “공유메모리 vs 세마포어를 이용한 chat program”, 한국콘텐츠학회논문지 1997

[2] Sean Walberg, “Share application data with UNIX System V IPC mechanisms”, IBM, 2007

[3] Allen B. Downey, “The Little Book of Semaphores (Second Edition) Revision data July 2007

[4] 장승주외 2 인, “Dual Core 시스템에서 Shared Memory 기능 설계”, 한국해양정보통신학회논문지 제 12 권 제 8 호 (2008 년 8 월)

[5] 장승주, “Dual core 시스템에서 shared memory 기능 구현”, 한국콘텐츠학회논문지 제 8 권 제 9 호 (2008 년 9 월) pp.27-33

[6] Qiao Xiangzhen, “Design of efficient parallel algorithms on shared memory multiprocessors”, Wuhan University Journal of Natural Sciences Volume 1, 2008

[7] 문지훈, ”AMP 멀티 코어 시스템에서의 공유 메모리 커널 모듈 구현 = Implementation of Kernel Module for Shared Memory in AMP Multi-Core System” 순천대학교 박사논문 (2015 년 2 월)