J · BOSN RTOS 개발하기④

마이크로 커널-동기화 서버

J · BOSN 실시간 운영체제는 고성능의 멀티-태스킹(Multi-tasking)을 지원하고 매우 강력한 실시간 응용프로그램을 개발할 수 있는 바탕을 제공하여 준다. 고성능의 실시간 커널(나노커널)과 다른 서비스모듈(마이크로커널, 매크로커널)이 결합되어 전체시스템이 이루어진다. 모든 모듈은 각자의 독립성을 최대한 보장하고 있으며, 각 모듈은 실시간 시스템의 서비스를 확장시킨다. JBOSN 운영체제에서 기본적으로 제공하는 모듈 등을 통해 실시간 운영 체제에 대해 자세히 소개하고자 한다. 이번호에서는 이러한 특성을 구현하기 위한 근본적인 개념을 담고 있는 동기화서버의 설계와 구성 원리를 살펴보겠다.

글 : 정병오 대표 / 아이보슨시스템즈 www.jbosn.com

연재 차례

JBOSN RTOS의 구조는 **그림 1**에 나타나 있다, 구조적인 측면에서 보면 그림 1에서 보여지듯 계층적이고 모듈화되어 있는 각 모듈들 중에 JBOSN RTOS 운영체제의 동기화와 통신 관련된 기능을 제공하는 목적으로 제작된 것이 동기화서버(Synchronization Server)이다. 운영체제의 측면에서 보면, 동기화 서버(Synchronization Server)는 시스템의 효율을 극대화하기 위하여 멀티쓰레딩(병렬처리)인 JBOSN RTOS에서, 서로간의 작업을 조율하기 위하여 쓰레드간의 동기화와 통신을 구현하는데 중요한 역할을 담당하는 운영체제의 필수적인 부분이다.

동기화는 "자원동기화"와 "동작동기화"로 나눌 수 있는데, "자원동기화"는 공유자원의 지배적인 접근권한을 적절히 배분하여 여러 쓰레드가 공유자원에 접근 할 때, 접근이 안전한지를 검사해 주거나, 안전하게 접근할 수 있는 방법론을 제시하여 주는 것이고, "동작동기화"는 나의 상태를 다른 쓰레드에게 알려주거나, 다른 쓰레드의 상태를 알 수 있거나, 특정상태가 될 때를 알려주는 방법론을 제시하여 준다. 통신은 데이터를 다른 쓰레드/태스크에게 송/수신하는 것이다. 이번호에서는 이러한 특성을 구현하기 위한 근본적인 개념을 담고 있는 동기화서버의 설계와 구성 원리를 살펴보겠다.

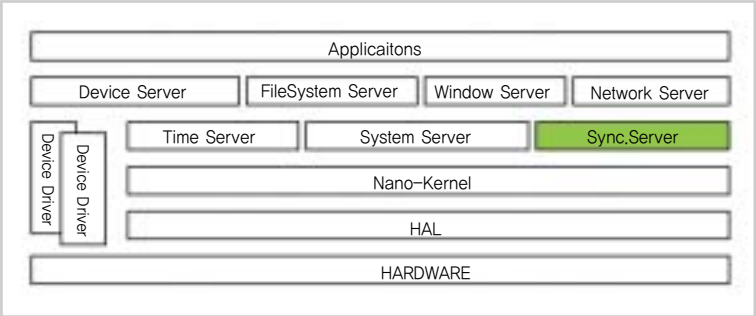

그림 1. JBOSN RTOS 구조

동기화 서버 (Synchronization Server)

실시간 시스템용 소프트웨어는 시스템 효율을 극대화하기 위하여 멀티태스킹을 이용한다. 응용프로그램 설계에는 보통 여러 개의 쓰레드가 병행하여 동작하는데 이들 객체간의 작업들을 조율하기 위해서는 쓰레드간 동기화와 통신이 필요하다.

동기화서버는 쓰레드들이 같은 자원을 공유할 때 발생하는 "자원동기화"를 해결하기 위한방법, 쓰레드들의 상태를 서로 주고받을 수 있는 "동작동기화"를 해결하는 방법, 쓰레드들간의 데이터를 교환할 수 있는 방법을 제공/관리하는 기능을 제공하는 JBOSN RTOS의 마이크로커널 서버중 하나이다. 동기화와 관련된 기능은 앞에서 설명한 다른 서버들과는 다르게 하드웨어와는 관련된 것이 없기 때문에 하드웨어와 직접적인 연결은 필요없다. 그림 2는 동기화서버에 있는 기능을 표시하였다.

동기화 서버에서 제공하는 도구로는 세마포어(semaphore), Mutex, Event, Conditional Variables, Message Queue 객체를 제공하고 있다. 또한 critical section 기능이 이 서버에서 직접 서비스하지 않지만 연관이 되어 시스템콜을 통하여 제공이 되고 있다. 동기화 서버에서 제공되는 기능을 사용할 때는 실시간 애플리케이션 입장에서 상당한 추가 부담(overhead)을 초래할 수 있으므로 적절히 사용하여 부담을 최소화 할 수 있게 사용하여야 한다. 이 기능을 사용하여 애플리케이션을 작성할 때는 여기서 제공하는 리소스를 사용하여 생기는 모든 문제는 사용자에게 전적으로 책임이 있다. 예를 들어, 이벤트 자원을 기다리는 쓰레드는 다른 쓰레드에서 이벤트 자원을 제공하여야 실행을 계속하게 된다. 만일 영원히 이러한 자원을 다른 쓰레드에서 제공받지 못하면, 이 쓰레드는 영원히 수행이 되지 않게 된다. 또한 여러 개의 쓰레드가 동시에 수행될 때 생길 수 있는 데드락(deadlock)이 발생할 수 있다. 결론적으로 이 서버가 제공하는 서비스를 사용하는 사용자는 상당한 주의가 필요하다.

자원동기화는 여러 개의 쓰레드가 동시에 같은 자원(공유자원)을 접근할 때 자원의 무결성을 위하여 자원의 접근을 동기화 시켜준다. 제공되는 기능 중 세마포어, 뮤텍스, 크리티컬섹션이 이러한 기능을 제공한다. 예를 들면 두개의 쓰레드가 하나의 메모리를 공유하고 있다고 가정해보자. 하나의 쓰레드는 공유 메모리에 데이터를 쓰고, 다른 쓰레드는 공유 메모리에서 데이터를 읽는 동작을 한다. 쓰는 동작을 하는 쓰레드가 데이터를 쓰고 있는 중간에 읽는 쓰레드가 데이터를 읽어 가면 데이터의 완결성이 없고 이전의 데이터와 현재의 데이터가 혼합된다.

동작 동기화는 다중 쓰레드로 작성된 프로그램이 제대로 동작을 하려면 쓰레드의 동작을 다른 쓰레드의 동작의 결과의 조건이나 상태에 맞추어서 동기화를 시켜줘야 한다. 이것이 동작 동기화는 조건 동기화 또는 순차동기화라도 불리기도 하는 이유이다. 동작 동기화는 동기적일 수도 있고 비동기적일 수도 있다. 제공되는 동기화 기능중 이벤트나 메시지 시그널이 대표적이다. 하나의 쓰레드는 여러 개의 쓰레드가 실행한 결과를 입력으로 받아서 작업을 진행하는 경우가 많다. 이러한 동작을 구현하기 위해서 동작 동기화를 사용한다.

통신을 위하여 제공되는 모델은 메시지 버스(message bus)가 존재한다. 통신을 위한 모델을 최소한으로 제한한 것은 JBOSN RTOS는 단일 메모리 공간에서 동작을 하고 있으

며, 쓰레드간의 의존성을 높였고, 태스크간의 의존성을 최대한으로 차단한다는 가정하에서 설계를 하였다. 데이터 통신으로 인한 메모리 내용의 이동을 최대한 제한하여 시스템의 전체 성능을 높이기 위해서이다. 데이터 통신을 필요로 할 때는 동기화 모델을 사용하여 자원(특히 메모리)의 동기화 기법을 사용하는 것을 추천한다.

또 다른 측면에서 보면, 멀티태스킹을 지원하는 시스템에서 쓰레드는 다른 쓰레드들과 통신을 사용하여 상호협력을 하면서 하나의 작업을 수행한다. 이 때 통신이란 쓰레드간의 데이터를 주고받는 행위를 말한다. 데이터란 특정 상태를 나타내는 신호나 실제 값을 나타내는 데이터 두 가지가 있다. 통신을 하는 방법은 신호를 전달하는 메시지 시그널이나 이벤트 등의 방법이나 실제 데이터를 전달하는 메시지버스 방법 등이 있을 수 있다. 그러나 실제 구현에서는 제한된 방법을 사용하는 것보다 다양한 동기화 기법을 사용하여 데이터 통신을 구현하는 경우가 많다.

결론적으로 이상의 기능들을 사용하면 데이터 전송, 쓰레드의 현재 상태를 알려주기, 다른 쓰레드 실행의 간접제어, 다른 쓰레드와 동작동기화를 이룰 수 있으며 또한 자원을 공유하기 위하여 여러 방법을 스스로 구현할 수 있다.

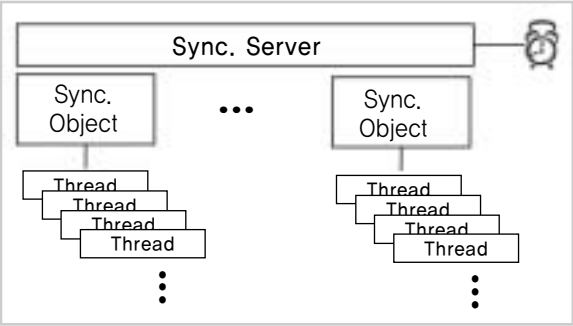

그림 2. Sync server 구성도

JBOSN RTOS에서 동기화나 통신기능을 구현하는 방식은 다른 운영체제와 상당히 다른 구조를 가지고 있다. 리눅스의 경우, 동기화객체에 접근하여 접근권한을 획득하기 위해서는, 다른 쓰레드들과 경쟁을 하여 획득하는 것이 일반적인 구조이다. 예를 들면, 한사람만이 통과할 수 있는 출구에 여러 사람이 동시에 나가려고 한다면, 서로 경쟁하고 경쟁하여 이긴 사람이 나가게 된다. 이런 구조는 상당히 구조를 단순화시킬 수 있다는 장점이 있지만, 일단 하나의 동기화 객체가 획득 가능한 상태로 변하게 되면, 해당하는 동기화 객체를 얻고자 하는 모든 쓰레드들이 경쟁에 참가하여 불필요한 경쟁을 벌이게 되고, CPU의 자원을 낭비하게 되는 단점이 있다.

JBOSN RTOS는 출구를 지키는 문지기를 하나 지정한다고 보면 된다. 나가려는 사람은 문지기에 등록을 하고, 자신의 순서를 기다리기만 하면 된다. 문지기는 등록된 사람들을 보고 나가는 순서를 미리 정하고, 순차적으로 출구로 내보내게 된다. 즉, 나가는 순서를 결정하는 것은 문지지(동기화서버)의 정책에 따라서 달라지게 되고, 소모적인 경쟁에 따른 CPU의 낭비도 줄이게 된다. 또 다른 장점은 쓰레드들이 동기화에 관련된 코드를 직접실행하지 않고 동기화서버에게 요청하고 허가만 받는 것이므로 시스템의 안정성에 크게 기여한다. 또한 동기화 서버만이 동기화객체에 접근할 수 있으므로 CPU의 atomic operation을 보장받지 않더라도 구현이 가능하다. 즉, 동기화 서버는 CPU의 종류에 따라 atomic operator로 동작성을 보장받지 않아도 되므로 하드웨어와 관련성이 전혀 없다.

제공되는 모든 동기화 및 통신 모델은 빠른 응답시간과 동작의 에러를 최소화하기 위하여 간단명료한 모델들로 작성이 되어 서로간의 통합이나 의존성을 제거하였다. 결론적으로 다른 운용체제에서 발견할 수 있는 매우 복잡하고 서로간의 의존성이 큰 동기화는 찾아볼 수 없지만, 주어진 기능만으로도 개발자들에게 충분하다고 판단이 된다. 오히려 복잡하고 의존적인 기능들을 모두 이해하여 정상적으로 사용하려할 때, 많은 오류가 발생할 수 있을 것이다.

다음은 동기화 서버에서 제공하는 기능에 대하여 자세히 살펴보겠다.

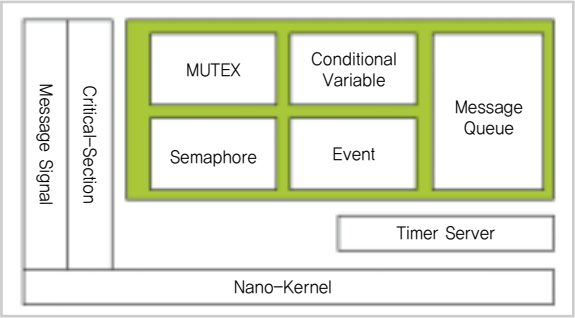

그림 3. 동기화 서버 구성도

크리티컬 섹션(CRITICAL-SECTION)

크리티컬섹션은 여러 개의 쓰레드들에서 공유되지만, 여러 개의 쓰레드들이 그 영역안에서 동시에 실행되지 않아야 되는 하나의 실행코드의 블록을 말한다. 이 실행 코드들의 대부분은 서로 공유되고 있는 자원을 접근하여 사용하는 것이 일반적이다. 이 경우에는 해당하는 부분의 실행 코드외에 다른 실행코드 부분들에서도 똑같은 공유된 자원을 접근하여 사용할 수도 있다. 따라서, 보다 광의적으로 크리티컬섹션을 해석하면, 지정된 하나의 공유자원을 접근하는 모든 실행코드 블록은 하나의 크리티컬섹션이라고 말할 수 있다.

크리티컬 섹션에 오직 하나의 쓰레드만이 접근하는 것을 보장하기 위하여 쓰레드들이 해당하는 크리티컬섹션에 진입하거나 빠져나오는 것을 감시하여야 한다.

서로 다른 쓰레드에서 공유자원을 접근할 수 있는데, 다른 동기화 방법과는 달리 실행되는 코드 집합에 진입하는 것을 제한하는 방법이다. 제일 먼저 크리티컬섹션에 진입을 하는 쓰레드는 다른 쓰레드의 방해를 받지 않고 공유자원을 독점적으로 사용하는 코드집합을 수행한 후에 크리티컬섹션에서 나오게 된다. 다른 쓰레드는 독점적인 사용이 끝날 때까지 대기한다. 하지만 이 경우 우선순위 역전현상과 크리티컬 섹션의 크기와 진입한 쓰레드의 수행 속도에 따라서 시간적으로 매우 중요한 쓰레드가 실행이 되지 못하는 경우도 발생한다. 왜냐하면 크리티컬섹션을 수행하지 않는 다른 쓰레드와는 경쟁관계에 놓이지 않기 때문에 다른 쓰레드는 수행에 방해를 받지 않아 CPU 자원을 빼앗아 갈 수 있다. 기본적으로 JBOSN RTOS는 크리티컬섹션을 제어하는 기능을 제공하고 있지만 시간적으로 매우 민감한 코드는 좀더 확실한 방법을 사용하기를 권장한다.

많은 코드를 살펴보면 크리티컬 섹션을 보호하기 위해서 인터럽트 잠금이나 선점 잠금과 같은 상호배재 기능을 사용하는 것을 볼 수 있다. 이 경우 크리티컬섹션을 매우 짧게 작성하지 않으면 시스템의 성능에 치명적인 부담이 될 수 있다. 하지만 시간적으로 매우 중요한 쓰레드는 위에 언급된 것을 희생하더라도 다른 외부사항의 방해없이 시간적인 요구사항을 만족시켜 줄 수 있다. 사용자는 어떤 것을 중시하느냐에 따라 크리티컬 섹션을 구현하는 방법을 선택할 수 있을 것이다.

크리티컬섹션을 동기화서버가 아닌 나노커널의 기능에 포함한 것은 위에서 설명한 보다 빠른 응답 속도를 구현하기 위함이다.

세마포어(SEMAPHORE)

세마포어는 실행중이 여러 쓰레드들이 동기화 또는 상호배재를 목적으로 획득하거나 반환하는 동기화객체이다.

여러 쓰레드들이 동시에 동작하는 멀티태스킹 기능은 필수적으로 쓰레드간의 자원을 접근을 할 때 경쟁을 하게 된다. 이때 하나의 자원을 독점적으로 접근할 권한을 부여하는 방법이 있어야만 쓰레드간의 자원 사용에서 충돌을 피할 수 있다. JBOSN RTOS는 세마포어를 제공하여 이런 기능을 수행할 수 있게 한다.

세마포어는 상호배재가 필요한 어떤 일을 수행할 때나 공유자원을 접근할 때 필요한 지배적인 접근 권한이다. 쓰레드가 세마포어를 획득할 경우 원하는 동작을 수행하거나 공유자원에 접근할 수 있지만, 접근 권한의 소진으로 획득을 실패할 경우 세마포어가 획득될 수 있을 때까지 대기행렬에서 기다린다. 세마포어의 종류는 바이너리 세마포어와 카운팅세마포어로 나눌 수 있는데, JBOSN RTOS는 두 가지를 하나의 방법으로 구현을 할 수 있게 제공한다. 세마포어의 내부 카운

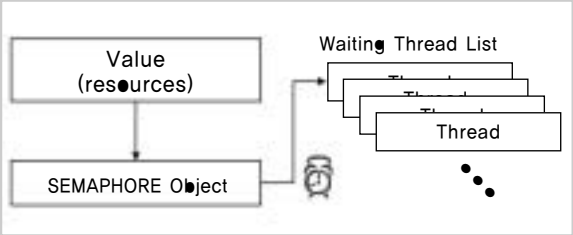

그림 4. Semaphore 구현도

터의 값이 '0' 일 경우 세마포어는 사용불가능하고 이외의 값을 가지고 있을 때는 그 값만큼 사용할 수 있다. 제한조건으로 최대로 증가할 수 있는 값(상한선)을 설정하여 무한히 증가하는 것을 막을 수 있게 되어 있다.

세마포어의 내부에 최대값을 '1' 로 설정을 하면 바이너리 세마포어를 구현할 수 있고, '1' 보다 큰 값을 설정하면 카운팅 세마포어를 구현할 수 있다. 세마포어의 카운터값은 현재 사용가능한 자원의 수를 나타내고 있으므로 세마포어를 설정할 때, 동시에 사용가능한 자원의 수를 최대값으로 설정을 하면 된다. 세마포어 자원을 얻을 때 카운터값을 1씩 감소하고, 세마포어 자원을 반환할 때 1씩 증가한다. 카운터값이 '0' 일때는 더 이상 자원이 없으므로 자원이 반환될 때까지 대기행렬에서 기다린다. 세마포어는 소유권이 없으므로 세마포어 핸들을 알고 있는 모든 쓰레드에서 접근이 가능하다.

뮤텍스(MUTEX)

뮤텍스는 재귀적 잠금과 소유권기능이라는 아주 특수한 제한조건을 가지고 있는 바이너리 세마포어의 일종이다. 일반적인 세마포어는 자원의 사용가능 개수를 나타내는 것과는 달리, 뮤텍스는 자원의 잠금과 풀림을 나타낸다. 뮤텍스는 자원의 잠금을 나타내기 때문에 획득을 하는 즉시 잠김 상태로 천이를 하게 되고, 반환을 하면 바로 풀림상태로 천이를 한다.

뮤텍스를 획득하면 세마포어와는 달리 소유권을 동시에 획득하기 때문에 다른 쓰레드에서 뮤텍스에 접근하는 것 자체가 불가능하게 된다. 따라서 획득한 쓰레드가 직접 뮤텍스를 반환하는 것을 기다려야 한다. 이러한 기능은 다른 쓰레드에 의해서 동기화가 방해를 받지 않고 안정적으로 동작하는 것을 보장하여 준다.

다른 기능으로는 재귀적 잠금을 지원하는 것이다. 재귀적 잠금이란 뮤텍스를 획득한 쓰레드는 반복적으로 해당 뮤텍스를 획득할 수 있다. 뮤텍스는 이 경우 반복적으로 획득된 숫자를 계산하고 있다가 뮤텍스를 반환할 때 숫자를 감소시킨다. 획득된 숫자만큼 반환이 되었을 때 뮤텍스 자원을 완전히 반환이 되어 소유권을 내놓게 된다. 이 기능은 뮤텍스를 획득하는 기능을 가지고 있는 함수가 반복적으로 호출이 될 경우에 매우 유용하게 사용될 수 있다. 세마포어처럼 상한선을 가지고 있지는 않다.

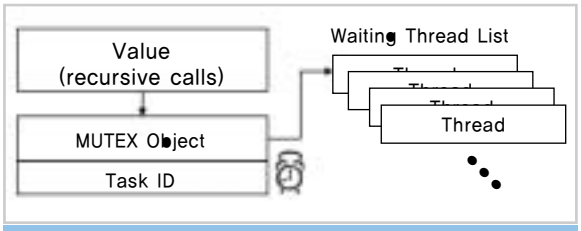

그림 5. MUTEX 구현도

이벤트(EVENT)

이벤트는 쓰레드에서 다른 쓰레드와의 실행 동기화를 맞추기 위해서 사용하는 기능이다. 다른 동기화 기능들은 공유자원의 사용에 대한 것이나 이벤트는 쓰레드가 실행할 때 자신의 실행 상황을 다른 쓰레드에게 알려주는 기능을 한다. 다른 쓰레드는 이벤트 자원으로부터 다른 쓰레드가 이미 알려준 정보를 보고 상황을 추측할 수 있다. 부가적으로 이벤트를 발생시킬 때 데이터를 전송시킬 수도 있는데, 이 데이터는 보호를 받지는 못하고 다른 데이터가 새로 전송이 되면 곧바로 새로운 값이 갱신된다. 결론적으로 가장 최근에 전송된 데이터가 남아 있을 것이다. 쓰레드가 이 값을 읽더라도 지워지지 않고 남아있다.

이벤트는 쓰레드가 실행을 할 조건으로 다른 쓰레드가 정하여진 상태로 진입하는 것을 구현할 때 매우 유용하다. 또한

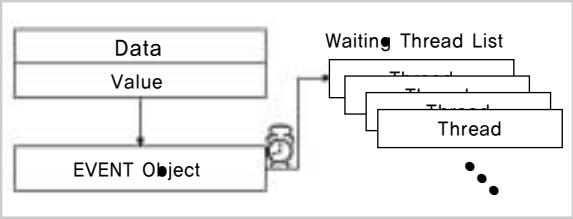

그림 6. EVENT 구현도

데이터를 전송할 수 있어서 결과값을 전송 받을 수도 있다.

조건 변수(CONDITIONAL VARIABLE)

조건변수는 쓰레드에서 다른 쓰레드와의 실행 동기화를 맞추기 위해서 사용하는 기능이다. 다른 동기화와 다른 점은 조건변수를 기다리는 쓰레드는 해당하는 조건을 만족하도록 변수가 변할 때까지 기다리는 것이다. 이 때 조건은 BITMAP-AND 연산을 수행하는 경우를 구현했다. 즉, 변수가 변하기를 기다리는 조건으로 여러 비트를 할당을 하고, 만일 여러 해당되는 비트중 하나만이라도 세팅(Set)되면 자원을 획득한 것으로 인식을 하는 것이다. 이러한 기능은 여러 개의 자원 중 최소한 하나를 기다릴 때 매우 유용하게 사용될 수 있다.

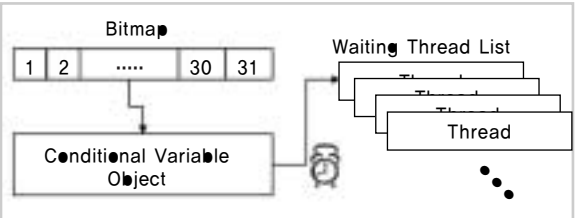

그림 7. Conditional Variable 구현도

메시지큐(MESSAGE QUEUE)

메시지큐는 쓰레드에서 다른 쓰레드로 데이터를 전달하기 위하여 사용하는 통신기능이다. 다른 동기화와 다른 점은 대용량의 데이터를 전달할 수 있다는 것이다. JBOSN RTOS의 메시지큐는 private buffer를 커널의 내부에 생성한다. 즉, 데이터를 보내는 쓰레드에 존재하는 데이터를 커널 내부에 존재하는 private buffer에 복사하고, 데이터 받는 쓰레드는 데이터를 커널 내부의 private buffer에서 복사를 해서 사용하게 된다. 따라서 대용량의 데이터를 보낼 때는 메모리의 복사로 인한 지연을 고려해야 한다. 커널 내부의 private buffer는 페이지 단위(4KB)로 할당이 되어 사용되므로 메시지 큐의 생성시 이 점을 고려하여 메시지 큐를 생성하여야 한다.

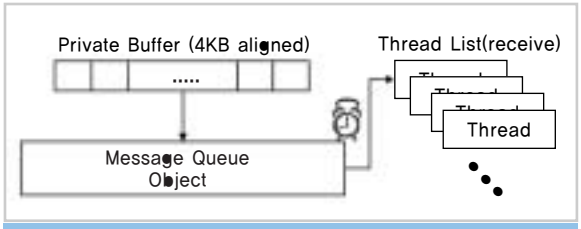

그림 8. Message Queue 구현도

메시지 시그널(MESSAGE SIGNAL)

메시지 시그널은 JBOSN RTOS 메시지버스를 통하여 비트맵 시그널을 전송하는 방법이다. 즉, 동기화 서버에서는 이 기능을 구현되지 않았다. 이 기능은 커널의 다양한 기능을 메시지버스에서 구현하려는 시도에서 탄생했다. 쓰레드들은 이 기능을 통하여 특별한 기능을 요청하거나 자신의 상태를 알려줄 수 있다. 이벤트는 수신측의 상황에 따라 수신을 반드시 하지 않아도 되지만, 메시지 시그널은 수신이 보장이 되므로 확실한 전송을 할 수 있다.

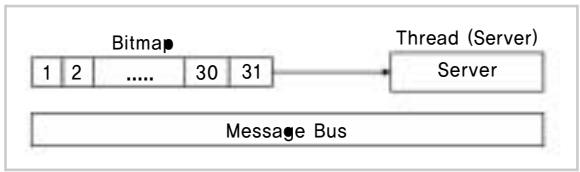

그림 9. Message Signal 구현도

이 기능은 커널에서 실행되는 서비스 서버들이 자주 이용하는 방법으로 사용자는 매우 주의를 하여 사용하여야 한다. 주로 커널이 서비스서버나 드라이버에게 특정한 사건의 발생을 알려주는 수단으로 사용된다.