J · BOSN RTOS 개발하기–마이크로 커널③

시스템 서버(System Server)

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. 지난 2월호 시간서버(Time Server)에 이어, 이달에는 운영체제의 자원관리(할당/회수) 및 시스템 전반의 감시기능을 목적으로 하는 시스템 서버를 소개하겠다.

자료제공 : 아이보슨시스템즈 www.jbosn.com

연재 차례

개요

JBOSN RTOS의 구조는 그림 1에 나타나 있다. 구조적인 측면에서 보면 **그림 1**에서 보여지듯이 계층적이고 모듈화되어 있는 각 모듈들 중에 JBOSN RTOS 운영체제의 자원(resources) 관리 및 시스템 감시와 관련된 기능을 제공하는 목적으로 제작된 것이 시스템서버(System Server)이다. 운영체제의 측면에서 보면, 시스템 서버(System Server)는 운영체제의 자원관리(할당/회수) 및 시스템 전반의 감시기능을 목적으로 하여, 전체 JBOSN RTOS의 안정성과 시스템의 현재 상태를 사용자에게 전달하여주며, 시스템 전체를 무정지 시스템으로 구현하는데 중요한 역할을 담당하는 운영체제의 필수적인 부분이다.

자원은 하드웨어적인 메모리를 포함하여, 소프트웨어적인 커널의 각종구조체 및 데이터들을 지칭한다. 이번호는 이러한 특성을 구현하기 위한 근본적인 개념을 담고 있는 시스템서버의 설계와 구성 원리를 살펴보겠다.

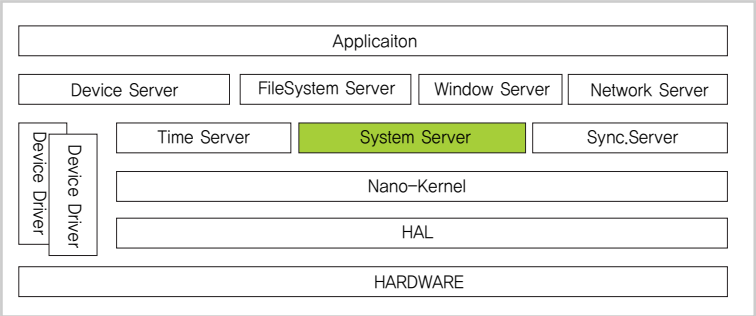

그림 1. JBOSN RTOS 구조

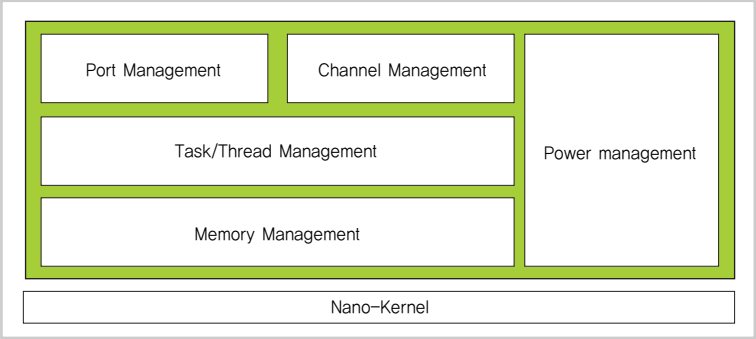

그림 2. 시스템서버 구성도

시스템 서버(System Server)

시스템 서버는 시간서버에서 관리하는 시간 자원을 제외하고, 운영체제에서 사용되는 모든 자원을 관리하는 기능과 JBOSN RTOS만의 독특한 기능을 제공하여 주는 JBOSN RTOS의 마이크로커널 서버중 하나이다. 자원과 관련된 기능 중 나노커널에서 제공하는 것은 하드웨어와 연결된 메모리를 제어할 수 있은 메모리 관리(Memory management)를 위한 기본 라이브러리와 태스크/쓰레드 관리(Task/Thread management)를 위한 라이브러리를 시스템 서버에 제공하는 것이다. 운영체제 자원 중에서 메모리는 임베디드 시스템에서 매우 중요한 자원이다. 메모리를 제어하는 방법은 나노커널에서 제공하는 라이브러리를 호출하여 제어한다. 물론 HAL 레이어에서 메모리와 관련된 정보를 나노커널에 제공하여 준다.

시스템 서버는 시스템 전체를 감시하는 기능을 제공하는 특성 때문에, 나노커널에서 가장 많은 도움을 받아야 하며, 제공되는 모든 기능이 나노커널에서 제공하는 정보와 라이브러리를 통하여 구현이 된다.

JBOSN RTOS는 커널이 사용하는 객체와 애플리케이션이나 다른 서버들이 사용하는 객체(Object)의 생성, 소멸 그리고 관리의 모든 것을 서비스해주고 있다. 따라서 특정한 자원을 사용하는 서비스 받기 위해서는 시스템 서버에서 객체를 할당 받아야 사용할 수 있고, 사용이 끝난 객체도 시스템서버를 통하여 되돌려 주어야 한다. 실제로 JBOSN RTOS에 제공하는 다른 서버들도 시스템서버를 통하여 필요한 객체의 생성/할당/소멸을 보장받는다. 특히 시스템의 전원까지 관리하는 서비스 모듈이다. 그림 2는 시스템서버에 에 있는 기능 표시했다.

다음은 시스템 서버에서 제공하는 기능에 대하여 자세히 살펴보겠다.

메모리 관리

메모리 관리는 나노커널의 도움을 받아서 이루어진다. 모든 메모리는 4Kbytes의 크기로 분할되어 있으며, 이것을 메모리 블록 페이지(memory block page)라고 한다. 메모리 관리는 페이지 단위로 구현되어 있으며, 사용자는 페이지 단위로 메모리를 할당/반납할 수 있다.

JBOSN RTOS는 MMU를 사용한 메모리 페이지의 'reallocation'을 제공하지 않는다. 모든 메모리는 하나의 공간에서 관리가 된다. 결론적으로 페이징 또는 스와핑(swapping) 등을 사용하는 매우 동적 메모리 관리(dynamic memory management)를 제공하지는 않는다. 커널을 포함하여 모든 애플리케이션은 단일 메모리 공간(memory space)

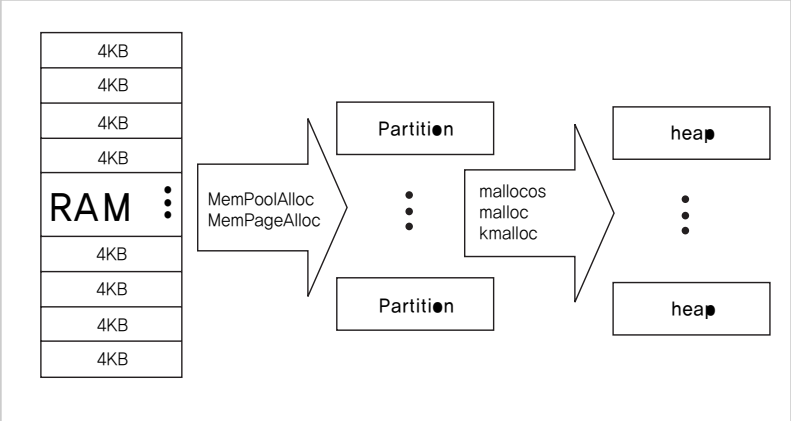

그림 3. memory allocation

을 사용한다.

JBOSN RTOS는 하나의 메모리 페이지(memory page)를 하나의 비트 단위로 변환하여 해당 메모리 페이지의 사용여부를 표시하는 방법과 메모리 페이지를 체인(chain) 형태로 연결하는 방법을 동시에 사용하고 있다. 이 방법은 메모리를 관리하기 위한 최소한의 메모리 요구량과 적은 오버헤드, 일정한 메모리 할당/반환 시간을 충족시키고 있다. 하지만 비트연산으로 인한 메모리 할당/반환에서의 속도의 감소는 존재한다.

메모리 관리에서 가장 중요한 점은 시스템의 지속적인 사용으로 인한 메모리 'fragmentation' 의 발생이다. 단편화 현상은 MMU를 사용한다면 쉽게 해결을 할 수 있는 문제이지만, JBOSN RTOS처럼 MMU를 사용하지 않는다면 치명적인 약점이 될 수 있다. 이 문제를 회피하려면 메모리 단편화가 발생하는 근원을 막아야 한다. JBOSN RTOS는 메모리 단편화 방지와 메모리 절약을 위하여 커널 차원에서 'heap' 을 제공하지 않는다.

사용자는 페이지 단위(4KB)의 할당만을 받을 수 있다. 사용자도 임베디드 시스템의 특성상 메모리의 할당/반환을 너무 반복적으로 방법은 피하는 것이 추천한다. 그림 3은 JBOSN RTOS의 메모리 사용법을 설명한 것이다.

JBOSN RTOS는 메모리를 할당받기를 원하는 사용자에게 두 가지의 API를 제공하여 준다. MemPoolAlloc(nPowerOfTwo)은 2의 지수승((4KB x 2nPowerOfTwo)으로 할당받는 경우이다. 이 경우는 내부적인 메모리 단편화가 적게 발생하여 보다 효율적인 메모리 관리가 가능하지만 할당받은 사용자는 필요한 메모리양보다 많은 메모리를 할당받는 경우가 발생하여 메모리를 낭비하게 된다. MemPageAlloc (nPage)은 메모리 페이지의 개수(4KB x nPage)로 할당을 받는 경우이다. 내부적인 단편화는 많이 발생하여 메모리의 효율성이 떨어질 수 있지만, 사용자는 필요한 메모리양만큼만 할당받아서 사용할 수 있으며, 메모리의 낭비를 줄일 수 있다.

애플리케이션에서 메모리를 할당 받아서 사용한 후, 메모리를 반환하기 위하여 할당받은 함수에 대응하여 MemPoolFree, MemPageFree 함수를 사용한다. JBOSN RTOS는 메모리를 할당받은 쓰레드가 메모리를 반환하지 않고 종료가 되었을 때, 자동으로 메모리를 회수하지 못한다. 그러므로 안정적인 메모리 사용을 위해서는 반드시 애플리케이션은 반드시 종료하기 전에 할당받은 메모리를 시스템에 반환하여야 한다.

마지막으로, JBOSN RTOS는 heap을 운영체제 내부에서 지원하지 않고 있으므로, 애플리케이션에서 heap을 사용할 수 없다. 그러나 애플리케이션에서 heap을 사용하려면 위에서 설명한 메모리할당 방법을 이용하여 메모리를 할당 받은 후에 JBOSN RTOS에서 제공되는 heap 라이브러리(user library)를 사용하면 heap으로 사용할 수 있다. 애플리케이션은 자신이 heap으로 사용하고 있는 메모리 용량까지 알 수 있어서 자신이 사용하는 메모리 용량을 정확히 예측할 수 있다. 결과적으로 사용자는 시스템 전체의 메모리 사용현황을 예측하고 메모리 문제로 시스템이 멈추거나 이상 동작하는 현상을 방지할 수 있다.

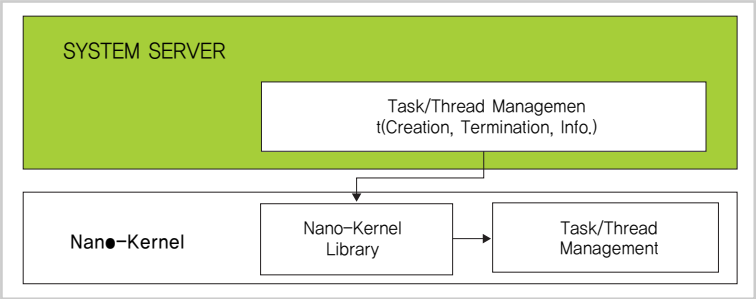

그림 4. 태스크/쓰레드 관리 호출도

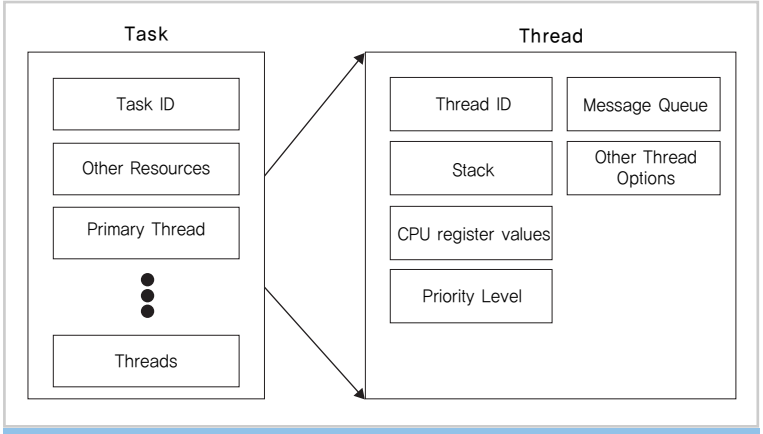

그림 5. 태스크/쓰레드 구성도

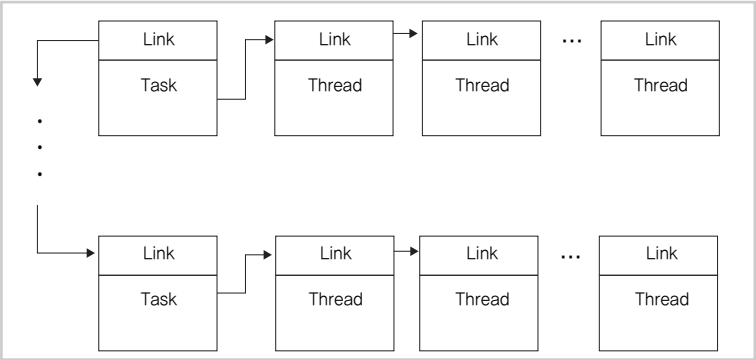

그림 6. 태스크/쓰레드 관계도

태스크/쓰레드 관리

태스크/쓰레드(Task/Thread) 관리는 나노커널과 시간 서버에서도 존재한다. 시스템 서버는 태스크/쓰레드를 실제로 운용하지는 않고, 태스크/쓰레드와 관련된 자원들만 관리하여 준다. 태스크/쓰레드가 생성될 때 필요한 다양한 자원을 할당하거나 종료될 때 자원을 회수하여 야 한다. 그림 4에서처럼 실제 할당된 자원을 사용하는 주체는 나노커널과 시간 서버이다. 즉, 시스템 서버는 관련된 자원의 관리만을 담당하므로 태스크/쓰레드의 동작에 미치는 영향이 거의 없다. 그러나 JBOSN RTOS를 사용하는 사용자 입장에서는 태스크/쓰레드의 생성/소멸/관리는 시스템서버의 서비스를 받는 것으로 충분하다.

태스크는 실제로 사용자의 생성시킨 애플리케이션을 나타내며, 여러 개의 쓰레드로 구성이 된다. 그림 5에 나타나듯이 태스크는 최소한 1개 이상의 쓰레드로 구성되며, 여러 개의 태스크가 생성되어 운영될 수 있다. 쓰레드는 나노커널에서 처리하는 가장 기본적인 처리단위이며, 시스템콜을 통해 운영체제의 서비스를 받을 수 있다.

그림 5는 태스크와 쓰레드의 구성을 표시하고 있다. 태스크는 필요한 자원과 그림 5에서처럼 쓰레드들을 가지고 있다. 쓰레드는 자신의 ID와 사용하는 스택, 우선 순위(priority level) 등등을 포함한 형태를 가진다.

그림 6은 JBOSN RTOS에서 'Task Objects'와 'Thread Objects'가 연관을 가지는 관계도를 나타내고 있다. 태스크를 생성시키면 반드시 자동으로 'primary thread'가 하나 생성된다. 즉, 사용자는 쓰레드들을 생성하기 위해서는 반드시 태스크를 먼저 생성하여야 한다. 또 다른 태스크를 생성하게 되면 이전의 태스크와 분리된 또 다른 태스크/쓰레드 그룹이 하나

더 생성되게 된다. 이러한 개념도는 하나의 프로그램은 하나의 태스크로 대변하기 위해서이다. 사용자가 프로그램을 제작할 때, 반드시 태스크를 하나 생성시키고, 쓰레드가 더 필요할 경우는 쓰레드만 생성하여 추가하면 된다. 전체적으로 보면 태스크는 현재 실행되고 있는 태스크의 개수와 같다고 보면 된다. JBOSN RTOS는 하나의 커널 태스크를 생성하여 사용하고 있다. 서비스를 제공하는 모든 커널 모듈들은 커널 태스크에 속해있는 쓰레드로 구성이 되어 있으며, 디바이스 드라이버등도 모두 커널 태스크에 속한 쓰레드이다.

포트/채널 관리

나노커널을 설명할 때, 포트(port)와 채널의 필요성에 대하여 알아보았다. 다른 항목들과 같이 시스템서버는 포트 객체의 할당과 회수만을 담당한다. 그림 9에서처럼 포트 객체는 나노커널의 메시지 버스에서만 사용된다. 채널은 시스템을 포팅하는 개발자가 현재 시스템에 최대 몇 개의 채널을 할당할 것인지 결정하여 제공한다. 아래에 설명하겠지만, 채널은 반드시 포트와 연결이 되어야(Channel Set 참조) 통신에서 사용될 수 있다. 채널은 일정한 구조체를 가지고 있지 않고, 포팅할 때 지정한 채널의 개수를 최대 자원으로 인식하여 시스템서버가 자원을 관리해 준다. 즉, 시스템서버에 의한 메모리의 할당/회수가 일어나지 않고, 나노커널의 메시지 버스에게 정보만을 전달한다.

여기에서 다시 메시지 버스를 이용한 서버/클라이언트 통신에 대하여 살펴보자. 서버는 기본적으로 클라이언트(client)의 요청을 서비스해주는 역할을 하므로, 기본적으로 갖추어야 요건은 서비스 요청을 받을 수 있는 통신방법과 결과를 응답할 수 있는 통신방법을 갖추어야 한다.

또한 여러 개의 클라이언트가 동시에 서비스를 요청할 경우 요청을 관리할 수 있어야 한다. JBOSN RTOS는 이러한 기능을 구현하기 위하여, 메시지버스를 이용한 통신 기능을 제공하고, 메시지 통신의 양단에 포트(port) 라는 개념을 도입하여 해결했다.

동작하는 방법을 비유적으로 설명하면, 컨테이너화물을 배에 선적하여 원하는 목적지로 보내는 것을 가정하여 보자. LA항에서 부산항으로 컨테이너를 보낸다고 가정하면, 보내는 사람은 LA항구에 정박한 화물선중 부산항을 거쳐가는 화물선을 찾아서 컨테이너를 선적할 것이다. 그러나, 해당하는 컨테이너가 어느 항구에서 내려놓아야 하는 것은 화물선 입장에서는 알 수 없다. 따라서 컨테이너가 하역되어야할 항구에 해당하는 일렬번호를 부여받아 적어 놓은 수화물표를 컨테이너에 붙이면 된다. 이 수화물표에 있는 일렬번호를 보고 화물선은 해당하는 컨테이너를 부산항에 내려줄 것이다. 받는 사람은 컨테이너를 부산항에서 인수하여 가져갈 것이다. 즉, LA항이나 부산항이 "port", 일렬 번호가 바로 'channel key", 보내는 사람은 클라이언트, 받는 사람은 서버에 각각 대응된다.

또 다른 예를 들면, 전화를 사용하려면 통신을 할 수 있도록 통신 선로를 연결하고 양 끝단에 전화기를 설치하여야 한다. 그리고 전화를 거는 사람은 받는 상대방의 전화번호를 알고 있어야 한다. 그러나 전화선 양끝단은 항상 송신자와 수신자가 있어야 통신이 이루어진다. 전화기는 "port", 전화번호는 "channel key", 송신자는 클라이언트, 수신자는 서버에 대응된다.

메시지는 클라이언트가 지정한 포트를 출발하여 서버에 직접 전달되지 않고 채널 키(channel key)에 지정된 포트에 전달되면, 서버는 이곳에서 서비스 요청 메시지를 얻어 서비스를 제공하게 된다. 클라이언트 또한 자신의 요청에 대한 서비스 응답 메시지가가 도착할 때까지 메지시를 전송한 포트에서 기다리고 있다. 서버는 수신된 메시지에 들어 있는 송신자와 포트를 찾아서 응답을 기다리는 송신자에게 응답메시지를 보낸다.

서버의 포트는 클라이언트에게 알려져 있지 않아 클라이언트가 서버를 접근할 방법이 없다. 이를 해결하기 위해서 서버는 클라이언트의 접근을 허용하기위한 통신채널 키(channel

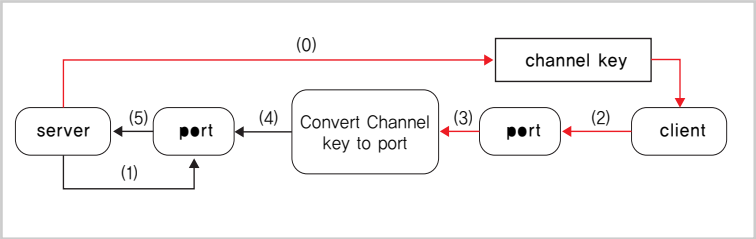

그림 7. 메시지를 이용한 클라이언트의 서비스 요청 구조

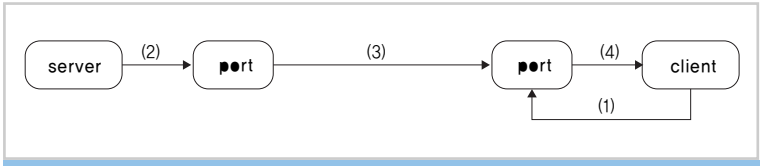

그림 8. 메시지를 이용한 서버의 서비스 응답 구조

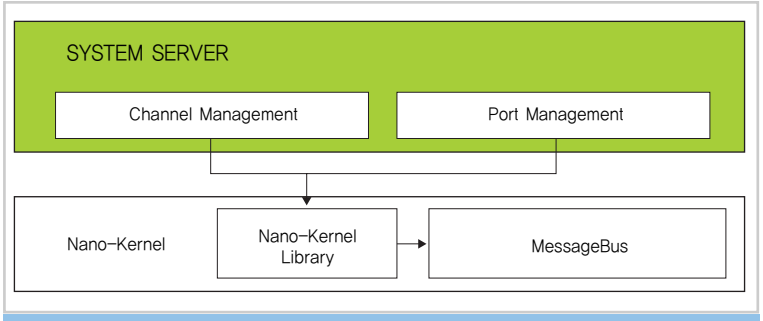

그림 9. 채널/포트 제어 흐름도

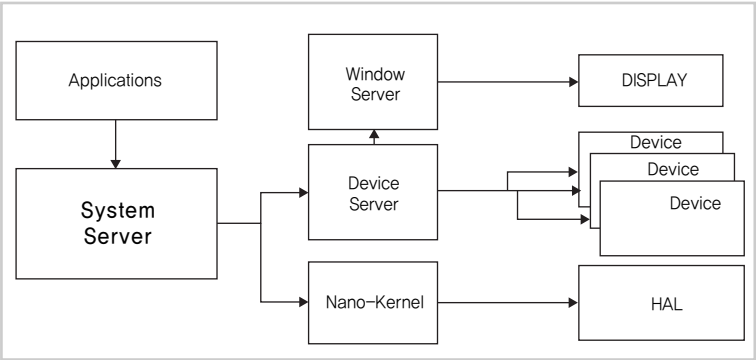

그림 10. 파워 관리 제어 흐름도

key : 수화물표, 전화번호)를 제공하게 된다. 이 채널 키를 이용하여 클라이언트는 서버에 요청을 보내고 메시지 통신 채널(나노커널의 메시지버스)은 이 채널 키를 해석하여 서버가 기다리고 있는 포트에 메시지를 전달을 하여준다. 결론적으로 클라이언트는 이미 공개되어 있거나 자신이 얻은 채널 키(channel key)를 통하여 서버에 접근을 할 수 가 있고, 서버는 자신이 배포한 채널 키 값에 연결된 포트를 통하여 서비스 요청을 기다린다.

이러한 기능은 다음과 같은 형태로 구현이 된다.

그림 7은 클라이언트가 서비스를 요청하는 순서를 나타낸 것이다. 첫 번째로 서버가 서버 포트를 통하여 메시지를 기다리고 둘째로 클라이언트는 공개키(channel key)를 이용하여 자신이 지정한 포트를 통해 서비스 요청 메시지를 전달한다. 셋째로 나노커널의 메시지 버스는 이 공개키를 포트로 변환한 후 서버의 포트에 메시지를 전달한다. 넷째로, 서버는 지정된 포트를 통해 메시지를 수신하게 된다.

그림 8은 서버가 요청받은 서비스에 대한 응답 메시지를 송신하는 순서이다. 첫째로 클라이언트는 서비스 요청을 한 후에 메시지를 보낸 포트에서 응답 메시지를 기다리게 된다. 둘째로 서버는 서비스 메시지를 받은 포트를 통하여 서비스를 요청할 때, 사용된 포트로 응답 메시지를 전송하게 된다. 셋째로 나노커널의 메시지버스를 통하여 응답 메시지는 서버 포트에서 클라이언트가 기다리는 포트로 전송되고, 넷째로, 클라이언트는 기다리던 포트에서 응답 메시지를 수신하게 된다.

서버는 서비스를 요청 받으면 반드시 서비스에 대한 응답을 해주어야 한다. 그러나 예외적으로 두 가지의 서비스 요청

메시지에 대하여는 응답을 하여 서는 안된다. 하드웨어의 인터럽트에 대한 서비스 요청 메시지(MESSAGE_TYPE_INTER RUPT)와 시그널메시지(MESSAGE_TYPE_ SIGNAL)이다. 인터럽트 메시지는 하드웨어의 인터럽트 요청 메시지를 수신하는 것으로 송신자가 존재하지 않는다. MessageSignal(x) API에 의해 생성되는 시그널메시지는 송신자가 응답을 기다리지 않고 진행하기 때문에 응답메시지를 보내서는 안된다.

파워 관리

시스템서버의 중요한 기능 중 파워 관리(power management) 부분이 있다. 시스템 전체의 파워와 관련된 통합적인 관리를 주 목적으로 한다. 나중에 설명할 디바이스 서버(Device server)는 등록되어 있는 디바이스 드라이버의 파워를 제어할 수 있다.

그림 10에서 보듯이 시스템서버는 전원절약모드로 진입하려면 디바이스서버와 나노커널을 이용한다. 디바이스서버에게 등록되어 있는 디바이스들의 전원절약 모드를 명령하고 수행한 후, 나노커널에게 전원절약 모드로 진입할 것을 명령하게 된다. 이때 나노커널은 HAL 레이어의 기능을 이용하여 시스템전체의 전원제어와 CPU의 전원절약 모드로 진입한다. 즉, 제어권을 HAL 레이어에서 멈추게 된다.

전원절약 모드에서 복귀하려면, 거꾸로 CPU와 전원을 동작모드로 전환하고, 제어권이 다시시스템 서버에 넘어오게 된다. 시스템 서버는 디바이스서버에게 등록된 디바이스들의 파워온(Power-On)을 명령하게 된다. 등록된 디바이스들이 동작하면 제어권을 애플리케이션에게 되돌려 주게 된다.

➔ 다음호에는 마이크로커널 중 'Synchronization Server'에 대하여 알아보겠습니다.