J · BOSN RTOS 개발하기-마이크로 커널⑤

매크로 커널-디바이스 서버

장치 서버(Device Server)는 시스템의 안정성과 보호를 극대화하고, 장치를 사용할 수 있는 권한, 등록하거나 등록을 해지하는 기능을 조율하는 기능을 구현하는데 중요한 역할을 담당하는 운영체제의 필수적인 부분이다. J · BOSN RTOS의 설계는 어떤 특수한 제약조건을 부과하여 기능의 구현을 제한하는 일은 하지 않도록 하는 것이 목적이므로, 사용자는 각자가 사용하려는 특수한 목적에 맞게 디바이스 접근 방법을 가지고 사용할 수도 있다. 이러한 기능은 실시간시스템을 설계할 때 설계자의 자율성을 최대한 보장함으로서 설계제품의 목적에 최적화된 해결책을 제시할 수 있다.

글 : 정병오 대표 / 아이보슨시스템즈 www.jbosn.com

연재 차례

개요

J · BOSN RTOS의 구조는 그림 1에 나타나 있다, 구조적인 측면에서 보면 그림 1에서 보여지 듯 계층적이고 모듈화되어 있는 각 모듈 중에 JBOSN RTOS의 디바이스 관리 및 제어에 관련된 기능을 제공하는 목적으로 제작된 것이 장치 서버(Device Server)이다. 운영체제의 측면에서 보면, 장치 서버는 시스템의 안정성과 보호를 극대화하고, 장치를 사용할 수 있는 권한, 등록하거나 등록을 해지하는 기능을 조율하는 기능을 구현하는데 중요한 역할을 담당하는 운영체제의 필수적인 부분이다.

J · BOSN RTOS의 구조는 나노커널을 기본으로하여 마이크로 커널(시스템서버, 시간서버, 동기화서버)을 사용하면 순수한 운영체제의 핵심기능을 제공해 준다. 매크로 커널은 외부확장기능을 담당한다. 이곳은 디바이스 관리와 통신을 위한 'Device server', 블록디바이스의 데이터를 관리해 주는 파일시스템 서버인 'File System Server', 네트워크 통신을 담당하고 있는 'Network Server', 그래픽과 윈도우 시스템의 제어하여 GUI를 제공하여 주는 'Window Server' 등 4가지로 이루어져 있다.

장치서버를 사용하여 부가적으로 타깃 시스템에서 사용하려는 디바이스를 관리하고 디바이스를 통해

데이터 서비스를 애플리케이션에게 제공하여 주는 부분을 담당하려는 확장이다. 외부 디바이스의 관리를 기초로 하고 있기 때문에 장치 서버의 기능이 매우 중요하다.

J · BOSN RTOS에서 디바이스를 사용하려면 디바이스 드라이버를 장치 서버에 반드시 등록시켜야 하며 애플리케이션에서 이 장치 서버의 도움으로 디바이스에 접근을 할 수가 있도록 되어 있다. 하지만 J · BOSN RTOS의 설계는 어떤 특수한 제약조건을 부과하여 기능의 구현을 제한하는 일은 하지 않도록 하는 것이 목적이므로, 사용자는 각자가 사용하려는 특수한 목적에 맞게 디바이스 접근 방법을 가지고 사용할 수도 있다. 이러한 기능은 실시간시스템을 설계할 때 설계자의 자율성을 최대한 보장함으로서 설계제품의 목적에 최적화된 해결책을 제시할 수 있다.

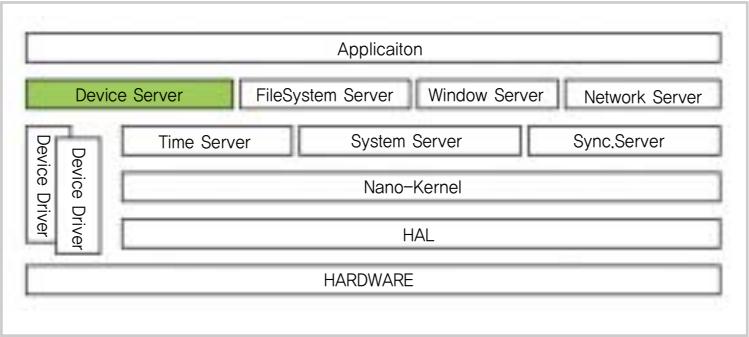

그림 1. JBOSN RTOS 구조

장치 서버 (Device Server)

이 모듈은 J · BOSN RTOS의 디바이스 드라이버의 등록과 해제 및 관리를 담당하는 서비스를 제공해 주며 서버의 형태를 갖고 있다. 드라이버는 시스템의 재부팅 없이 자유롭게 사용 중에도 등록될 수 있고 또한 해제될 수도 있다. 모든 드라이버는 이곳을 통하여 디바이스에 대한 접근 권한을 가지게 되므로 반드시 이곳에 등록이 되어야만 접근을 허용하는 것이 기본적인 개념이다.

실시간 시스템용 소프트웨어는 시스템 효율을 극대화하기 위하여 멀티태스킹을 이용한다. 응용프로그램 설계에는 보통 여러 개의 쓰레드가 병행하여 동작하는데 이들 객체간의 작업들은 하드웨어 장치로부터 데이터의 이동과 제어를 담당하는 경우가 많다. 여러 쓰레드가 동일한 하드웨어 장치를 사용하려면 장치 서버에서 접근 권한(Handle)을 획득해야 하는데, 한정된 장치 자원(device resorce)을 효율적으로 분배하기 위한 작업을 조율하기 위해서는 장치관리가 필요하다.

장치 서버는 쓰레드들이 응용 프로그램과 하드웨어 간의 의존성에 매우 중요한 역할을 담당한다. 장치 서버를 사용하면 표준적인 디바이스 접근 API(IoOpen, IoRead, IoWrite, IoControl, IoClose)를 사용할 수 있다. 응용 프로그램은 디바이스를 접근할 때, 표준 API만을 사용하면 해당 디바이스의 제어 방법에 따라 영향을 받지 않는다. 즉, 디바이스가 달라지더라도 응용 프로그램은 변경되지 않아도 되므로, 하드웨어로부터 응용 소프트웨어가 완벽하게 분리되는 것이다. 디바이스 드라이버가 정확하게 디바이스를 제어해 주면 응용 프로그램은 정확하게 동작하게 된다.

시스템을 개발할 때, 상호의존성을 배재하게 되어 소프트웨어의 재사용성, 안정성, 확장성 및 표준화가 가능하다. 이런 특성은 개발제품의 안정성뿐만 아니라 개발기간의 단축 및 비용절감을 획기적으로 개선해 준다. 기능강화/변화에 따른 차기 모델에서 하드웨어가 변경되더라도 큰문제가 발생하지 않고 개발을 빠르게 진행할 수 있다.

장치 서버는 등록된 디바이스들의 전원을 시스템의 상태에 맞게 제어를 해 줄 수 있다. 결론적으로 이상의 기능들을 사용하면 J · BOSN RTOS에 다양한 다바이스를 연결하거나 분리할 수 있으며, 응용프로그램 및 기타 모듈들은 하드웨어와 분리가 가능하고 등록된 디바이스들은 전원관리를 받을 수 있다.

다음은 장치 서버에서 제공하는 기능에 대하여 자세히 살펴보겠다.

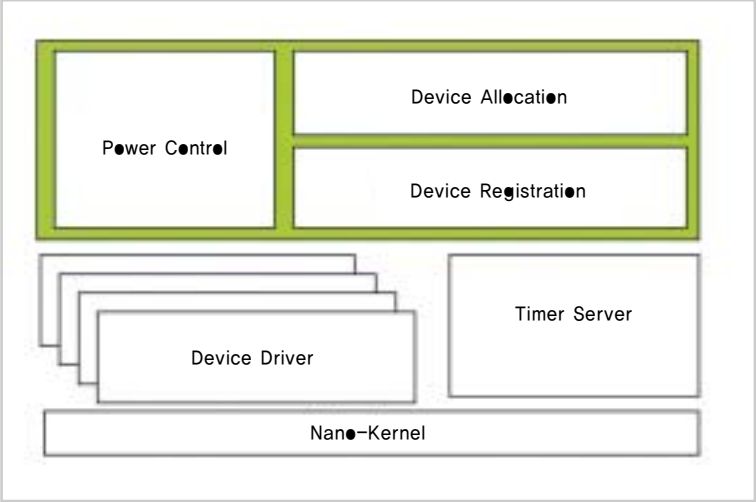

그림 2. 장치 서버 구성도

장치 등록(Device Registration)

장치 등록 기능은 디바이스들의 등록과 해제를 담당한다. 등록할 수 있는 디바이스의 최대 크기(숫자)는 부팅할 때, "NKERNEL_BOOT" 구조체에서 받게 된다. 사용자는 이 설정값을 적절한 값으로 조절하여 준다. 디바이스들이 등록을 할 때, 자신의 장치이름(예: COM1)과 통신에 필요한 채널 ID를 장치 서버에 넘겨주게 된다. 장치 이름은 Device Allocation에서 사용하게 되고, 채널 ID는 장치서버가 디바이스드라이버를 제어하거나 통신이 필요한 경우 사용하게 된다.

처음 디바이스가 등록이 되면, 장치 서버는 디바이스 드라이버에게 초기화를 명령하여 모든 등록절차가 끝난다.

장치 할당(Device Allocation)

장치 할당 기능은 응용 프로그램이나 커널모듈들 또는 기타 모듈들에서 원하는 디바이스 드라이버에 접근하기 위한 권한을 획득하려 할 때 동작된다. 장치 등록 블록에서 등록된 디바이스 드라이버의 이름과 호출된 디바이스 드라이버의 이름을 비교하여, 호출된 디바이스 드라이버의 통신 채널로 오픈 명령을 보내서 성공하면 접근권한을 부여하게 된다. 만일 이 과정에서 오류가 발생하면, 접근 권한을 획득하지 못한다.

전제조건으로 디바이스 드라이버의 이름을 호출하는 프로그램에서 미리알고 있어야 한다. 따라서 J · BOSN RTOS는 많이 사용되는 표준적인 디바이스들의 일반적인 이름을 사용하기를 권장한다. 만일 현재 시스템에서 제공하는 디바이스들의 이름을 알 수 없을 경우에 대비하여, 장치 할당 블록에서는 상위의 프로그램들에게 요청이 오면 제공 가능한 디바이스들의 이름과 특성들을 알려준다.

파워 컨트롤(Power Control)

파워 컨트롤 기능은 시스템의 상태에 따라서 장치 서버에 등록된 디바이스들의 전원상태를 조절할 수 있는 기능이다. 일반적으로 시스템이 슬립(Sleep) 모드에 진입할 때, 디바이스들의 전원을 슬립 상태에 놓거나, 웨이크 업(Wake up)하여 일반 모드로 진입할 경우, 디바이스들도 같이 웨이크 업 시켜준다. 그러나 모든 디바이스들이 이 기능으로 통제가 되는 것은 아니고, 디스플레이 장치처럼 비표준 장치는 장치 서버에서 통제하지 않고 별도의 블록에서 통제가 된다(참조: System Server, Window Server).

디바이스 드라이버(Device Driver)

J · BOSN RTOS를 사용하는 시스템의 외부확장을 위해 제어하려는 하드웨어에 관련된 관리를 담당하는 서버가 장치 서버이다. 장치 서버를 통하여 데이터의 서비스나 디바이스를 제어하는 기능을 제공할 수 있도록 한다. 디바이스는 각각

의 특성이 모두 달라서 특별히 제약조건을 부여하면 시스템의 성능에 영향을 미칠 수 있다. 따라서 J · BOSN RTOS에서는 통신 채널만을 규약하고 내부에 접근하는 방법은 각 디바이스의 특성에 맞게 시스템 설계자가 정의를 하는 것이 가능하도록 하여 구조를 구성하여 드라이버를 작성하는데 특별한 제약을 부과하지는 않았다. 그러나 J · BOSN RTOS는 일반적으로 많이 사용되는 드라이버의 모델들도 제공하고 있다. 예를 들면, 시리얼 통신과 같은 종류의 스트림 디바이스(Stream device)를 사용하려 할 때 최적으로 사용될 수 있는 스트림 디바이스 드라이버 모델을 제공하고 있고, 사운드와 같은 디바이스의 드라이버의 모델을 제공하고 있다. 제공되는 드라이버의 모델도 역시 통신 채널만을 규약 받아서 사용되고 있어서 사용자 스스로 재작성을 하여도 다른 모듈에 영향을 주지는 않는다.

모든 드라이버는 서버의 형태를 갖추어야 하며, 각 서버는 고유의 통신 채널을 확보할 수 있다. 이 통신 채널은 장치 서버에 등록이 되어 디바이스에 접근하려는 쓰레드들에게 암호화된 핸들로 제공된다.

그림 3을 보면 모든 다비이스 드라이버는 장치서버에 등록을 하게 된다. 응용 프로그램을 포함한 모든 모듈은 장치 서버를 통하여 접근 권한을 얻은 후 원하는 디바이스 드라이버에 접근이 가능하다. 그림 3에서 보듯 매크로 커널의 다른 서버들도 장치 서버의 도움을 받아야 한다.

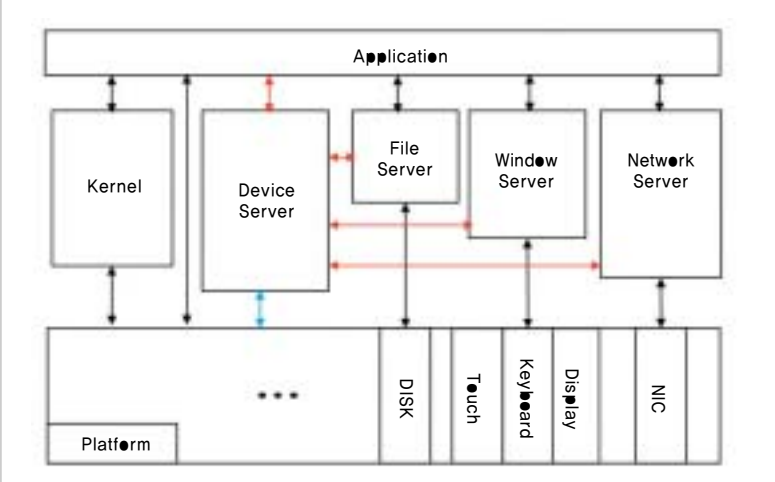

그림 3. 장치서버의 역할

개요

J • BOSN RTOS의 디바이스 드라이버의 작성에 대하여 보다 상세하게 살펴 보겠다. 디바이스 드라이버도 서버 모델을 기본으로하여 작성한다. 차이점이 있다면, 디바이스 드라이버는 하드웨어를 통한 데이터를 서비스하거나 하드웨어를 제어하는 목적으로 작성이 된 서버이므로, 일반 서버와는 다르게 장치 서버에 등록이 되어야 다른 모듈이나 애플리케이션에서 접근할 수 있다. 디바이스의 등록 이외에도 디바이스 초기화와 하드웨어 인터럽트 등록 등의 항목이 추가 된다.

디바이스 드라이버 서버는 하드웨어를 제어하여 하드웨어에 대한 데이터 서비스나 하드웨어 제어를 담당하는 기능도 포함이 되어 있어서, 다른 모듈로부터 서비스 요청을 받아 서비스를 제공해야 할 뿐만 아니라 하드웨어에서 발생하는 인터럽트에 대해서도 서비스해야 한다. 상당수의 서비스 요청은 하드웨어 인터럽트와 연관이 되므로 서비스 요청과 인터럽트의 절묘한 조화가 요구된다. 예를 들어, 드라이버가 512바이트의 데이터를 디바이스로부터 읽어달라는 요청을 받았으나 현재 디바이스에 있는 데이터가 500바이트이면 나머지 12 바이트의 데이터가 도착할 때까지 서비스를 끝낼 수 없다. 데이터가 디바이스로부터 들어오는 것을 기다리기 위해서 폴링 기다림(Polling wait)이나 인터럽트 기다림(interrupt wait) 방법을 선택한다. 여기서는 인터럽트 기다림을 선택했다고 가정하면, 다음 인터럽트가 발생할 때까지 기다려야 한다. 그러나 현재 제어하는 디바이스의 통신이 쌍

방향 통신(Full duplex)이 가능하다면 데이터를 쓰는 서비스 요청도 동시에 서비스를 해주어야 한다. 그러나 현재 인터럽트를 기다리고 있기 때문에 쓰기 요청이 왔을 때는 서비스를 제공하지 못하게 드라이버가 작성 될 수도 있다. 이런 상황이 발생하지 않도록 드라이버는 항상 서비스를 제공할 수 있는 준비가 되어 있어야 한다.

조금 전의 예제에서 보았듯이 드라이버는 언제나 서비스를 제공할 수 있어야 하므로 인터럽트 기다림을 서비스 요청의 기다림과 동시에 해야 한다.

따라서 J · BOSN RTOS에서는 디바이스 드라이버에 전달되는 인터럽트도 메시지의 형태로 전달이 되도록 설계되어 있다. 여러 개의 서비스 요청이 동시에 들어 왔을 때 서버는 각 서비스의 인터럽트 기다림을 수행해야 하고 현재 서비스가 완료되기를 기다리고 있는 쓰레드를 관리할 수 있어야 한다. 또한 인터럽트가 발생했을 때 어떤 서비스에 대한 인터럽트인가를 결정하여 서비스를 수행한 후 완료된 서비스 요청은 곧 응답을 해 주어야 한다.

결론적으로 다른 서버와의 차이점은 여러 개의 서비스를 동시에 수행해야 한다는 것이다. 이것을 이루기 위해서 서비스 요청들의 관리와 인터럽트를 각각의 서비스와 연계할 수 있는 능력이 필요한데 J · BOSN RTOS는 이런 기능을 구현하기 위하여 몇 가지 함수를 제공하고 있다. 일단 드라이버 모델을 한번 살펴보자.

초기 설정

```
1. STATIC BOOL
2. th_DeviceDriver( UINT32 Arg )
3. {
4. JB_RECVMESSAGE Message;
5. HANDLE          hPort;
6. UINT32          DevHandle;
7. HANDLE          OpenHandle = NULL;
8. DRIVER_REQUEST  DeviceRequest[MAX_DEVICEREQUEST] = {{NULL,},};
9. PDEVICECONFIG pDeviceConfig = (PDEVICECONFIG)Arg;
10. hPort = PortGet();
11. PortSetType(hPort, PORTTYPE_MESSAGE|
    PORTTYPE_INTERRUPT|
    PORTTYPE_SIGNAL);
12. pDeviceConfig->Name[3] = (UINT32)pDeviceConfig->Minor + '1';
13. pDeviceConfig->Channel = ChannelGet( hPort );
14. DeviceRegister( (PDEVICECONFIG)pDeviceConfig );
15. InterruptRegister(nInterrupt, hPort);
```

그림 4. 디바이스 드라이버 초기 설정부 예

줄11까지는 서버 모델과 동일하고, 줄12에서 줄14까지 드라이버를 장치 서버에 등록을 하는 부분이다. 이곳은 실제 디바이스의 이름(줄12)과 통신을 하려는 채널키(줄13)를 설정한다. 줄15에서는 인터럽트를 등록시킨다. 인터럽트가 발생했을 때 메시지를 전달할 포트를 지정하는 줄이다.

```
1. while(1)
2. {
3.   MessageReceive( hPort, &Message);
4.   switch(Message.Type)
5.   {
```

그림 5. 디바이스 드라이버 메시지 수신부 예

이 부분은 서버의 기본모델과 동일하여 서비스 요청 메시지를 받아들이고, "Type" 항목을 보고 서비스 목록을 결정한다.

인터럽트 및 Timeout 서비스

```
1. case MESSAGE_TYPE_SIGNAL:
2. {
3.    if( Message.Param & SIG_RESERVED_ALARM)
4.   {
5.       INT32    i;
6.       Message.Type = TRUE;
7.         for( i = 0; i < MAX_DEVICEREQUEST; i++)
8.         {
9.          BOOL bRet;
10.  bRet = ThreadDeviceRequestTimeout( Device
Request[i].hRequester, &Message, DeviceRequest[i].nByte -
DeviceRequest[i].nToBe);
11.             if( bRet )
12.            {
13.              BOOL   bWait;
14.              bWait = ThreadDeviceRequestWaitRetrieve(
&RequestWaitHead[i], &Message );
15.             if( bWait )
16.             {
17.                 if( i == 0 )
18.                     goto READ_REQUEST;
19.                 else
20.                     goto WRITE_REQUEST;
21.             }
22.         }
23.     }
```

그림 6. 디바이스 드라이버 Timeout(소프트웨어)인터럽트 예

J · BOSN RTOS는 크게 두 가지의 인터럽트를 가지고 있다. 첫 번째는 하드웨어로부터 발생하는 하드웨어 인터럽트로서 하드웨어가 특정 서비스를 받고 싶을 때, 운영체제에게 서비스를 요청하는 신호이다. 이것은 해당하는 디바이스의 서비스를 담당하는 드라이버에게 전달되어 처리가 된다. 두 번째는 소프트웨어에 강제로 발생시키는 소프트웨어 인터럽트이다. 이것은 특정 소프트웨어가 긴급한 정보를 알리고 싶을 때 사용된다. 드라이버는 서비스를 요청할 때 타임아웃 기능을 설정 하면 마이크로커널의 시간 서버(Time Server)가 드라이버에게 타임아웃이 발생했음을 소프트웨어 인터럽트를 이용해 알려준다. 이때 메시지 타입은 "MESSAGE_TYPE_SIGNAL" 이고, 인터럽트 번호는 "SYSINTR_RESERVED_ALARM" 이다.

소프트웨어 인터럽트는 이미 시간 서버에게 타임아웃값을 제공하여 일정시간이 지나면 소프트웨어 인터럽트를 발생시켜줄 것을 예약한 후에 발생이 된다. 서비스를 요청하여 디바이스를 접근할 때 일정 시간동안 서비스가 제공되지 않으면 에러를 발생시키고 서비스 실패를 알려주기를 원하는 애플리케이션이 이를 알 수 있도록 하기 위해 만들어 놓은 것이다. 줄1은 인터럽트 메시지를 의미하고, 줄2는 인터럽트가 타임아웃으로 인한 소프트 인터럽트라는 것을 나타내서 줄3부터 줄16까지는 현재 서비스를 기다리고 있는 것 중 타임아웃된 서비스 요청목록을 찾아서 타임아웃이 발생하였음을 요청한 쓰레드에 설정을 하고 서비스를 중단하는 것을 나타낸다. 여기서는 J · BOSN RTOS가 제공하는 "ThreadDeviceRequestTimeOut" 함수를 사용하면 쉽게 기능을 구현할 수 있다.

```
1.  case MESSAGE_TYPE_INTERRUPT:
2.  {
3.   INTERRUPT_TYPE IntType;
4.   IntType = 인터럽트의 타입을 얻는다.
5.   if( IntType == INTERRUPT_TYPE_NONE )
6.   {
7.   InterruptDone( 인터럽트 번호 );
8.    continue;
9.   }
10.   if( IntType & INTERRUPT_TYPE_WRITE )
```

```
11. {
12.  데이터를 디바이스에 쓰고, 서비스를 완료하였다면
13. ThreadDeviceRequestDone함 수 를  호 출 하 고 ThreadDeviceRequestWaitRetrieve를 사용하여 다시 서비스대기행렬을 검사한다.
14. }
15. if( IntType & INTERRUPT_TYPE_READ )
16. {
17.  데이터를 디바이스에서 읽고, 서비스를 완료하였다면
18. ThreadDeviceRequestDone함수를 호출하고 ThreadDeviceRequestWaitRetrieve를사용하여 다시 서비스대기행렬을 검사한다.
19. }
20. }
```

그림 7. 디바이스 드라이버 하드웨어 인터럽트 예

하드웨어 인터럽트는 하드웨어로부터 서비스가 도착하였거나, 하드웨어가 서비스를 받기 원할 때 발생한다. 예를 들면, 시리얼통신과 같은 경우 데이터가 외부에서 도착하였거나, 외부로 하드웨어가 요청받은 데이터를 모두 전송했을 경우 발생하게 된다.

하드웨어 인터럽트가 디바이스로부터 전달이 되었다면, 먼저 처리할 인터럽트의 종류를 얻어서 해당하는 인터럽트 서비스와 연결시킨다. 여기에서는 디바이스로부터 데이터를 읽는 경우와 디바이스에 데이터를 쓰는 경우를 가정했다. 첫 번째 경우는 줄10부터 줄14까지인데 데이터를 읽고 서비스 요청에 만족되었을 경우, J · BOSN RTOS가 제공하는 "ThreadDeviceRequestDone" 함수를 사용하여 읽기 서비스의 종료할 수 있다. 둘째의 경우 줄15에서 줄19까지인데 데이터를 쓰고 서비스 요청에 만족되었을 경우 읽기 서비스 종료와 동일하게 처리할 수 있다.

OPEN 서비스

```
1. case MESSAGE_TYPE_DEVICE_OPEN:
2. {
3.  if( OpenHandle != NULL)
4.  {
5.  ThreadSetLastError( Message.hSender, ERR_BUSY);
6.  Message.Type = NULL;
7.  break;
8.  }
9.  if( !Do_OpenDevice(DevHandle, pOps, Message.Param) )
10.  {
11.  ThreadSetLastError( Message.hSender, ERR_IO);
12.  Message.Type = NULL;
13.  break;
14.  }
15.  OpenHandle = ChannelOpenHandleGetFrom( pDeviceConfig->Channel );
16.  Message.Type = OpenHandle;
17.  break;
18. }
```

그림 8. 디바이스 드라이버 Open 서비스 예

"Open" 서비스는 요청한 쓰레드가 속해있는 태스크가 현재 드라이버를 사용하기 위하여 접근 허가권을 얻는 행위(IoOpen) 중에서 디바이스 드라이버에게 허가를 요청하는 부분을 말한다. J · BOSN RTOS에서 제시된 드라이버 모델에서는 현재 오픈되고 다음에 접근을 할 때 제시해야할 접근 코드(OpenHandle)를 만들어 리턴하여 준다.

줄1은 서비스타입이 Open 명령임을 나타낸다. 줄3은 이미 다른 오픈 서비스가 실행이 되었는지를 검사하는 것이다. 이미 다른 서비스를 실행했다면 줄5에서 줄7은 에러를 표시하고 줄9는 실제 오픈 서비스를 위한 디바이스에 접근한다. 에러가 발생하면 에러를 표시한다. 줄15는 정상적으로 디바이

스가 동작을 했을 때 자신이 가지고 있는 채널을 사용하여 다른 서비스를 접근할 때, "ChannelOpenHandleGetFrom" 함수를 사용하여 허가 받은 서비스 요청임을 확인하려는 OpenHandle을 얻어 줄16, 줄17에서 제공하게 된다. 물론 이런 복잡한 과정을 거치지 않고 여러 개의 태스크들이 동시에 접근하도록 가능성을 열어 둘 수도 있다. 하지만 여기서는 하나의 태스크만 접근할 수 있도록 하게 하였다.

READ 서비스

```
1. case MESSAGE_TYPE_DEVICE_READ:
2. {
3.    UINT32                    nRead = 0;

      PIOREAD_PARAM          pIoReadParam;
4.    if( Message.Handle != OpenHandle )
5.    {
6.          ThreadSetLastError( Message.hSender,
ERR_INVAL);
7.       Message.Type = 0;
8.       break;
9.    }
10.   if( 이미 서비스를 받을 다른
```

그림 9. 디바이스 드라이버 Read 서비스 예

"Read" 서비스는 디바이스로부터 데이터를 읽어서 서비스(IoRead)를 요청한 쓰레드에게 데이터를 전송하여 주는 작업을 하게 된다. 서비스를 요청한 디바이스는 데이터를 전송받을 메모리 버퍼와 데이터 개수를 바이트 단위로 요청한다.

줄1은 메시지타입이 데이터를 디바이스로부터 읽는 서비스가 요청된 것을 나타낸다. 줄4부터 줄9는 OpenHandle과 현재 서비스 요청에 있는 핸들이 서로 다른 경우, 에러를 발생시킨다.

줄10부터 줄15까지는 이미 읽기 서비스를 요청한 쓰레드가 있으면 대기행렬에 서비스 요청을 대기시킨다. 줄16부터 줄17은 실제 디바이스로부터 데이터를 읽는다.

줄18은 실제 요청된 데이터의 크기와 디바이스로부터 읽은 데이터의 크기를 비교하여 이미 읽기 서비스가 완료되었으면 줄25부터 실행하여 데이터 완료를 나타내고, 그렇지 않다면 줄19부터 줄24까지 실행한다.

줄26은 서비스가 완전히 제공되고 난후 수행이 되는데, 다른 서비스 요청이 있는지를 검사하여 있으면 다시 서비스를 제공한다. 여기서는 드라이버 자체의 데이터 관리 구조(DeviceRequest: J · BOSN RTOS는 제공하지 않음)를 통하여 서비스를 진행할 목록을 저장하고 현재 서비스된 상태를 기록하여 다음에 인터럽트 서비스에서 사용하게 된다. 또한 이곳은 서비스의 타임아웃 값을 지정할 수 있다. "TimeOut"을 지정할 때는 J · BOSN RTOS가 제공하는 "Thread DevcieRequestSet" 함수를 사용하여 지정할 수 있다. 그리고 서비스가 완료가 되지 않았으므로 완료 메시지를 보내지는 않는다.

WRITE 서비스

```
1.  case MESSAGE_TYPE_DEVICE_WRITE:
2.  {
3.  UINT32              nWrite = 0;
    PIOWRITE_PARAM  pIoWriteParam;
4.   if( Message.Handle != OpenHandle )
5.   {
6.  ThreadSetLastError( Message.hSender, ERR_INVAL);
7.   Message.Type = 0;
8.   break;
9.   }
10.  if( 이미 서비스를 받을 다른 요청이 있다면 )
11.  {
12.    이미 서비스가 진행되고 있으므로, 현재 서비스요청은 진
행할 수 없다.
```

```
13.  따라서 ThreadDeviceRequestWait함수를 이용하여 서비스요청을 대기행렬에 넣는다.
14.  continue;
15.  }
16.  pIoWriteParam = (PIOREAD_PARAM)Message.Param;
17.  nWrite = 해당하는 디바이스에서 데이터를 전송한다.
18.  if( 요청한 크기의 데이터를 전송하지 못했다면 )
19.  {
20.  //대기요청에 들어가서 데이터를 전송할 기회가 올까지 대기하기 위하여 다음을 호출한다.
21.  // Register the thread to requestwait link chain
22.  ThreadDeviceRequestSet( Handle for request thread , hPort, WriteTimeOut);
23.  continue;
24.  }
25.  서비스를 완료하였으면 ThreadDeviceRequestWaitRetrieve를 사용하여 서비스요청이 있는지를 검사한후 다시 서비스를 수행한다.
26.}
```

그림 10. 디바이스 드라이버 Write 서비스 예

"Write" 서비스는 서비스(IoWrite)를 요청한 쓰레드로부터 데이터를 받아서 디바이스에게 데이터를 전송하여 주는 작업을 하게 된다. 서비스를 요청한 디바이스는 전송될 데이터가 있는 메모리 버퍼와 데이터 개수를 바이트 단위로 요청한다.

줄1은 메시지타입이 데이터를 디바이스에 쓰는 것이다. 줄4부터 줄9는 OpenHandle과 현재 서비스 요청에 있는 핸들이 서로 다른 경우 에러를 발생시킨다. 줄10부터 줄15까지는 이미 쓰기 서비스를 요청한 쓰레드가 있으면 대기행렬에 서비스를 대기시킨다.

줄16부터 줄 17까지는 실제 디바이스로 데이터를 쓴다.

줄18은 실제 요청된 데이터의 크기와 디바이스에 쓴 데이터의 크기를 비교하여 이미 쓰기 서비스가 완료되었으면 줄25부터 실행하여 데이터 완료를 나타내고, 그렇지 않다면 줄19부터 줄24까지 실행한다.

줄25은 서비스가 완전히 제공된 후 수행이 되는데, 다른 서비스 요청이 있는지를 검사하여 다시 서비스를 제공한다. 여기서는 드라이버 자체의 데이터관리 구조 (DeviceRequest: J · BOSN RTOS는 제공하지 않음)를 통하여 서비스를 진행할 목록을 저장하고 현재 서비스된 상태를 기록하여 다음에 인터럽트 서비스에서 사용하게 된다. 또한 이곳은 서비스의 Timeout값을 지정할 수 있다. "TimeOut"을 지정할 때는 J · BOSN RTOS가 제공하는 "ThreadDevcieRequestSet" 함수를 사용하여 지정할 수 있다. 그리고 서비스가 완료가 되지 않았으므로 완료 메시지를 보내지는 않는다.

기타 서비스

다른 드라이버 서비스도 이와 비슷하게 제작을 할 수가 있다. 드라이버의 자세한 예제는 J · BOSN RTOS의 제공되는 패키지의 예제를 참조하기 바란다. 다시한번 강조를 하지만, 위에서 제시한 방법론은 하나의 샘플이고 반드시 위와 같은 방법으로 작성을 강요하는 것은 아니다. 사용자는 각 디바이스의 특성을 고려하여 서버 모델을 벗어나지 않는 범위 내에서 자유로이 드라이버를 작성할 수 있다.

자세한 내용은 J · BOSN RTOS의 샘플을 참조하기 바란다. [J · BOSN RTOS ROOT]/platform/common/driver/serial/

➔ 다음호에는 매크로커널 중 "FileSystem Server"에 대하여 알아보겠다.