J · BOSN 실시간 운영체제 ②

J · BOSN 실시간 나노커널에서 멀티태스킹의 구현

J · BOSN 실시간 운영체제는 고성능의 멀티-태스킹(Multi-tasking)을 지원하고 매우 강력한 실시간 응용프로그램을 개발할 수 있는 바탕을 제공하여 준다. J · BOSN 실시간 운영체제의 개념을 알아본 지난호에 이어, 이번호에는 매크로커널, 디바이스 드라이버, 응용 프로그램 등과 태스크/쓰레드 기능에 대하여 중점적으로 알아본다.

자료제공: 아이보슨시스템즈
www.jbosn.com

J · BOSN 실시간 운영체제

1회 J · BOSN 실시간 운영체제란 무엇인가 I
2회 J · BOSN 실시간 운영체제란 무엇인가 II
3회 J · BOSN의 동기화 및 통신 모델

마이크로커널(Micro Kernel)

마이크로커널은 나노 커널에서 제공해 주는 라이브러리를 사용하여 상위 스택에 있는 모듈에 나노커널의 서비스를 연결시켜 주거나, 나노 커널의 라이브러리의 기능을 완벽하게 구현을 하는 곳이다. 또한 멀티태스킹을 완벽히 구현 할 수 있는 기능을 제공하고 있다. 물론 이곳의 모듈들은 모두 서버 형태로 제작이 되어 있어서 나노커널의 메시지 통신을 근간으로 동작된다. 각 기능은 3개의 서버로 나누어서 구현이 되어 있고, 각 서버의 역할은 다음과 같다.

시간 서버(Time Server)

이 모듈은 시간과 관련된 기능을 담당하고 서버의 형태를 갖고 있다. J · BOSN 운영체제는 시간에 관한 관리, 타임-아웃(Time-Out), 와치독(Watich-Dog)과 쓰레드(Thread)의 퀀텀(Quantum)에 관련된 서비스가 구현이 되어 있다. 이 서버는 Tick Timer의 인터럽트에 의해서 구동되거나 시간에 관한 서비

스를 요청받을 때 이에 대한 서비스를 제공하여 준다.

시스템 서버

이 모듈은 J · BOSN 운영체제의 모든 리소스에 대한 관리를 담당하고 서버의 형태를 갖고 있다. J · BOSN 운영체제는 커널이 사용하는 객체와 애플리케이션이나 다른 서버들이 사용하는 객체(Object)의 생성, 소멸 그리고 관리의 모든 것을 서비스 해주고 있다. 따라서 어떤 객체를 서비스 받기 위해서는 이곳에서 서비스 객체를 할당받아야 사용을 할 수 있고, 모두 사용이 된 객체는 이 서버를 통하여 되돌려 주어야 한다. 실제 J · BOSN 운영체제의 다른 서버들도 이 서버를 통하여 객체의 할당과 소멸을 보장 받는다.

동기화 서버

이 모듈은 J · BOSN 운영체제의 쓰레드 간의 동기화와 통신을 하기 위한 도구를 제공하여 주며, 서버의 형태를 갖고 있다. 이 서버에서 제공하는 도구로는 세마포어(semaphore), Mutex, Event, Conditional Variables, Message Queue 객체를 제공하고 있다. 또한 크리티컬섹션(criticalsection) 기능이 이 서버에서 직접 서비스하지 않지만 연관이 되어 시스템 콜을 통하여 제공이 되고 있다. 이 기능을 사용할 때는 실시간 애플리케이션에서는 상당한 추가부담(overhead)을 초래할 수 있으므로 적절히 사용하여 부담이 되지 않게 사용하여야 한다. 이 기능을 사용하여 애플리케이션을 작성할 때는 여기서 제공하는 리소스를 사용하여 생기는 모든 문제는 사용자에게 전적으로 책임이 있다. 예를 들어 이벤트 리소스를 기다리는 쓰레드는 다른 쓰레드에서 이벤트리소스를 제공하여야 실행을 계속하게 된다. 만일 영원히 이러한 리소스를 다른 쓰레드가 제공하지 않으면, 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 주의를 필요로 한다.

매크로커널(Macro Kernel)

J · BOSN의 커널은 나노커널과 마이크로 커널을 사용하여 부가적으로 시스템에서 사용하려는 디바이스를 관리하고 디바이스를 통해 데이터 서비스를 애플리케이션에게 제공하여 주는 부분을 담당하려는 확장이다. 마이크로커널까지는 순수한 운영체제의 핵심기능을 제공한다면, J · BOSN의 매크로커널에 부가된 기능은 외부 확장과 관계된 부분이다. 이곳은 디바이스 관리와 통신을 위한 I/O 서버, 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. 외부 디바이스의 관리를 기초로 하고 있기 때문에 I/O 서버의 기능이 상당히 중요하다.

J · BOSN 운영체제에서 디바이스를 사용하려면 I/O 서버에 디바이스 드라이버를 반드시 등록시켜야 하며 애플리케이션에서 이 I/O 서버의 도움으로 디바이스에 접근을 할 수가 있도록 되어 있다. 하지만 J · BOSN 운영체제의 설계는 어떤 특수한 제약조건을 부과하여 기능의 구현을 제한하는 일은 하지 않도록 하는 것이 목적이다.

따라서 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.

입출력관리 서버(I/O Server)

이 모듈은 J · BOSN 운영체제의 디바이스 드라이버의 등록과 해제를 담당하는 서비스를 제공해 주며 서버의 형태를 갖고 있다. 드라이버는 시스템의 재부팅 없이 자유롭게 사용중에도 등록될 수 있고 또한 해제될 수도 있다. 모든 드라이버는 이곳을 통하여 디바이스에 대한 접근 권한을 가지게 되므

로 반드시 이곳에 등록이 되어야만 접근을 허용하는 것이 기본적인 개념이다.

파일시스템 서버

이 모듈은 J · BOSN 운영체제에서 블록디바이스에 접근하여 데이터를 서비스할 수 있도록 작성된 서버이다. 애플리케이션은 블록 디바이스에 접근하려할 때 이 서버를 통하여 접근하여 각종 서비스를 제공받을 수 있다. 현재는 FAT 파일시스템과 ROM 파일시스템, RAM 파일시스템을 서비스 할 수 있지만 서버의 구조상 다른 파일시스템을 자유롭게 추가할 수 있는 구조로 설계가 되어있다.

또한 디바이스 접근 방법은 I/O 서버에 등록된 디바이스 드라이버를 기본으로 사용하고 있기 때문에 사용자는 특별히 전용드라이버를 작성하지 않고 통상적인 디바이스 드라이버를 작성하는 방법만으로도 블록디바이스 드라이버를 작성할 수 있다.

윈도우관리 서버

이 모듈은 J · BOSN 운영체제에서 디스플레이 장치에서 윈도우를 관리해주는 서비스를 제공할 수 있도록 작성된 서버이다. 애플리케이션은 윈도우를 기본으로 그래픽 작업을 할 수 있어서 디스플레이 장치에 원하는 영상을 표시하고 싶을 때, 이 서버를 통하여 구현할 수 있다. 모든 윈도우는 위젯(widget)을 기본단위로 처리를 하게 되어 있다. 위젯은 윈도우관리 서버에서 제공되는 위젯프로시져를 통하여 관리가 되며, JBOSN 운영체제어서 제공되는 위젯드로우(widget draw)와 그래픽라이브러리를 통하여 디스플레이 장치에 표시를 할 수 있다.

네트워크 서버

이 모듈은 J · BOSN 운영체제에서 네트워크에 관련된 서비스할 수 있도록 작성된 서버이다. 애플리케이션은 네트워크에 접근하려할 때 이 서버를 통하여 접근하여 각종 서비스를 제공받을 수 있다.

현재 소켓 인터페이스(socket interface)를 기본으로 제공하며, TCP/UDP, IP, ICMP, ARP, RARP 등 여러 가지 프로토콜을 서비스 할 수 있지만 서버의 구조상 다른 프로토콜도 자유롭게 추가할 수 있는 구조로 설계가 되어있다. 또한 디바이스 접근 방법은 I/O 서버에 등록된 디바이스 드라이버를 기본으로 사용하고 있기 때문에 사용자는 특별히 전용 드라이버를 작성하지 않고 통상적인 디바이스 드라이버를 작성하는 방법만으로도 네트워크디바이스 드라이버를 작성할 수 있다.

디바이스 드라이버(Device Drivers)

이 모듈은 J · BOSN 운영체제의 외부확장을 위해 제어하려는 하드웨어에 관련된 모든 관리를 담당하는 디바이스 관리 서버이다. 이 서버를 통하여 데이터의 서비스나 디바이스를 제어하는 기능을 제공할 수 있도록 한다. 디바이스는 각각의 특성이 모두 달라서 특별히 제약조건을 부여하면 시스템의 성능에 영향을 미칠 수 있다. 따라서 단지 통신 채널만을 규약하고 내부에 접근하는 방법은 각 디바이스의 특성에 맞게 시스템 설계자가 정의를 하는 것이 가능하도록 하여 드라이버를 작성하는데 특별한 제약을 부과하지는 않았다. 하지만 J · BOSN 운영체제는 일반적으로 많이 사용되는 드라이버의 모델을 제공하고 있다. 첫째로 시리얼 통신과 같은 종류의 스트림 디바이스(Stream device)를 사용하려 할 때 최적으로 사용될 수 있는 스트림 디바이스 드라이버(Stream Device Driver) 모델을 제공하고 있고, 둘째는 사운드와 같은 디바이스의 드라이버의 모델을 제공하고 있다. 제공되는 드라이버의 모델도 역시 기본적으로 통신 채널만을 규약 받아서 사용되고 있어 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 I/O 서버

에 등록이 되어 디바이스에 접근하려는 쓰레드들에게 제공된다.

응용 프로그램(Applications)

이것은 J · BOSN 운영체제의 모든 기능을 사용하여 원하는 시스템을 구현하는 부분으로 실시가 시스템 설계자가 직접 작성하는 소프트웨어이다. J · BOSN 운영체제는 원하는 기능을 완벽히 제공하는 실시간 시스템을 쉽게 구현할 수 있는 도구만을 제공하고 있고, 애플리케이션은 이러한 도구를 잘 조합하여 실제 시스템을 완성하는 것이다. 제공되는 도구를 사용하기 때문에 J · BOSN 운영체제에 대한 충분한 이해가 있어야 한다.

태스크/쓰레드(Task/Thread)

멀티태스킹 구현

멀티태스킹 기능은 애플리케이션이 다양하고 비정규적인 실세계의 다양한 이벤트를 처리할 수 있는 기본적인 기능을 구현할 수 있게 해주어서 매우 역동적인 멀티태스킹 시스템을 구축하게 한다. 기본적인 멀티태스킹의 구현은 J · BOSN 운영체제의 실시간 나노커널에서 구현 해준다. 멀티태스킹 커널이 스케줄링 알고리듬을 사용하여 많은 프로세스들을 교체하면서 실행하여 주며, 많은 프로세스가 동시에 수행이 되는 것처럼 보이게 된다. 하나의 태스크는 여러 개의 쓰레드를 가지고 있지만, 이 쓰레드들의 모임은 한 가지 일을 사용자에게 서비스를 해준다.

실시간 시스템은 이런 태스크를 여러 개 가지고 있을 수 있다. J · BOSN 운영체제는 쓰레드를 기본단위로 처리를 하고 있으므로 J · BOSN 운영체제에서 쓰레드는 각자의 독립적인 컨텍스트를 가지고 있지만, 태스크에 공통되는 리소스는 태스크에서 관리를 하게 된다. 그러나 가장 중요한 메모리 자원는 쓰레드나 태스크의 리소스로 관리가 되지 않고 J · BOSN 운영체제에서 통합적으로 관리하고 있다. 즉, J · BOSN 운영체제에서는 모든 쓰레드는 하나의 어드레스 스페이스(address space)를 사용하고 있고, 메모리 자원의 관리는 커널에서 통합적으로 관리를 한다는 뜻이다. 적은 메모리 자원을 효율적으로 사용하고 쓰레드 간의 어드레스 스페이스 분리를 하지 않음으로서 고효율의 실시간시스템을 구축할 수 있다.

쓰레드(Thread)

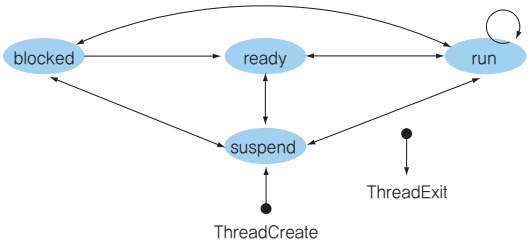
시스템의 측면에서 보면 쓰레드는 시스템 자원을 사용하여 자신에게 주어진 일을 수행할 수 있는 가장 작은 단위를 말한다. 즉, 쓰레드는 J · BOSN 운영체제에서 처리되고 관리되는 가장 작은 단위이다. 이런 측면은 다른 쓰레드들이 간섭을 하지 않는다면, 쓰레드가 시스템 자원을 사용할 수 있거나, 사용가능한 시점까지 기다릴 수 있다는 뜻이다. 여기서 말하는 시스템 자원은 CPU, I/O 디바이스, 메모리, 통신기능, 기타 등등 하드웨어나 소프트웨어의 모든 자원을 일컫는다.

개념적으로 쓰레드들은 서로 독립적이고 동시에 실행을 할 수 있다. 정확히 말하면 J · BOSN 운영체제는 단지 실행할 수 있는 쓰레드 등을 스위칭만을 하여주는 것이다. 예를 들면, 하나의 쓰레드가 시스템 서비스를 요청하였지만 원하는 서비스를 받지 못하였을 경우에 서비스가 준비될 때까지 멈추게 된다. 이때 커널은 다른 실행 가능한 쓰레드를 이전에 멈추었던 곳부터 실행시켜 주는 것이다.

비록 각 쓰레드들은 논리적으로는 서로 분리가 되어 있지만, 서로간은 동기화를 시키거나 상호협력을 함으로서 요구되는 일을 함께 완성하게 된다.

쓰레드 상태 천이(Thread State Transition)

쓰레드 상태라는 것은 쓰레드가 현재 시스템 자원에 대하여 어떤 상태에 있는가를 나타낸다. 쓰레드는 여러 가지의 상태 중에 한 가지를 가지고 있다. 또한 시스템 자원의 요구와 획득에 따라서 상태가 천이된다. 상태와 상태천이를 완벽하



'run' 상태: CPU 자원을 획득하여 실행하고 있는 상태
'ready' 상태: CPU 자원만 획득을 하면 즉시 실행될 수 있는 상태
'blocked' 상태: 현재 사용이 가능하지 않는 자원을 기다리고 있는 상태
'suspend' 상태: 스스로 또는 강제로 실행이 멈추어진 상태

그림 1. Thread의 상태 천이도

게 이해하는 것은 J · BOSN 운영체제를 이용하여 완벽한 멀티태스킹 애플리케이션을 제작하는데 매우 중요한다.

J · BOSN 운영체제 입장에서 보면 쓰레드는 생성되기 이전이나 종료된 후에는 아무런 의미가 없다. 쓰레드가 생성이 되었을 때 의미를 가지고 또한 실행이 되어야 시스템을 구성하는 요소가 된다. J · BOSN 운영체제는 각 쓰레드의 상태를 항상 관리하고 있다. 상태천이는 쓰레드가 시스템 서비스를 요구할 때, 처음 생성이 될 때, 종료될 때 등 여러 가지 원인으로 인하여 발생할 수 있다. J · BOSN 운영체제에서 관리하는 쓰레드의 상태는 4가지로 나눌 수 있다. 그림 1은 가능한 쓰레드 상태와 상태천이를 나타내고 있다.

'run' 상태는 실행 가능한 쓰레드들 중 가장 우선순위가 높은 것이 CPU 자원을 점유하여 사용하고 있는 상태를 말한다. 이 상태는 'ready' 상태에 있는 쓰레드 중에서 선택될 수 있고, 또는 'blocked' 상태에 있던 쓰레드들 중 바로 기다리던 자원을 받아 'ready' 상태에 있는 것보다 우선순위가 높을 때 바로 'run' 상태로 천이하게 된다. 현재 CPU를 사용하고 있는 쓰레드보다 우선순위가 높은 쓰레드가 실행가능하게 되면, 현재 실행되던 쓰레드는 'ready' 상태로 천이되고 높은 우선순위를 가진 쓰레드가 'run' 상태로 천이된다.

'ready' 상태는 실행이 가능한 상태로서 자신보다 높은 우선순위를 가지고 있는 쓰레드가 CPU 자원을 놓아 줄 때까지 기다리고 있는 곳이다. 쓰레드는 준비되지 않은 시스템 자원을 획득하려고 할 때만 'blocked' 상태로 천이를 하게 되므로 'run' 상태로 천이되지 못한 쓰레드는 절대로 'blocked' 상태로 천이될 수 없다.

'blocked' 상태는 요청한 시스템 자원이 사용 가능할 때까지 기다리는 상태를 말한다. 기다리던 자원을 획득하게 되면 'ready' 상태에 있는 쓰레드들과 우선순위를 비교한 후 가장 높다면 곧바로 'run' 상태로 천이하게 되며, 그렇지 않다면 'ready' 상태로 천이되어 CPU 자원을 기다린다.

'suspend' 상태는 쓰레드가 생성을 되었지만 실행을 멈춘 상태를 말한다. 스스로 또는 강제로 이 상태로 천이될 수 있다. 하지만 원래 있던 상태로 되돌아가려면 다른 쓰레드의 도움이나 커널 시스템의 호출이 필요하다.

위 천이도의 핵심은 쓰레드는 'run' 상태에서 자신의 실행에 의해서만 'blocked' 상태로 이동이 가능하고 다른 쓰레드에 의해서 강제로 'blocked' 상태로 천이되지 않는다. 또한 'blocked' 상태에 있는 쓰레드도 'suspend' 상태로 천이될 수 있다. 즉 실행을 할 때 스스로 'suspend' 상태로 천이할 수 있지만, 다른 쓰레드에 의해서 강제로 'suspend' 상태로 천이가 될 수 있다.

쓰레드 스케줄링(Thread scheduling)

멀티태스킹 기능을 구현하기 위해서는 여러 개의 실행 가능한 쓰레드 중에 CPU 자원을 할당해주는 스케줄링 기능을 필요로 한다. J · BOSN 운영체제는 우선순위(priority based) 및 선점형(preemptive) 스케줄링 알고리즘을 기본으로 사용하고 있다.

또한 같은 우선순위를 가진다면 라운드로빈(round-robin) 알고리듬을 사용하고 있다. 일반적으로 J · BOSN 운영체제의 스케줄링은 언제든지 실행 가능한 쓰레드들 중에 가장 높은 우선순위를 가진 쓰레드가 반드시 실행되는 것을 보장한다. 이 선점형 기능은 보다 높은 우선순위를 가진 쓰레드가 실행가능 상태가 되면 즉시 현재 쓰레드의 컨텍스트를 저장을

하고 보다 높은 우선순위의 쓰레드의 컨텍스트에 CPU를 할당하게 된다.

하지만 같은 우선순위를 갖는 쓰레드들이 서로 경쟁을 할 때는 J · BOSN 커널이 현재 실행되고 있는 쓰레드가 얼마나 오래 실행되었는가를 자동으로 추적하여 미리 정하여진 timeslice값(Quantum)을 초과하였다면 'ready' 상태로 천이를 시키고, 같은 우선순위를 갖는 다른 쓰레드에게 CPU를 할당한다.

즉, Timeslice 값(Quantum)은 같은 우선순위를 갖는 쓰레드가 여러 개 있을 때 의미를 가지게 되며, 라운드로빈(round-robin) 스케줄링을 구현하는 기본단위가 된다. 결론적으로 우선순위가 가장 높은 쓰레드는 자신이 원할 때 언제든지 실행이 가능하고, 실행이 멈추는 것은 스스로 시스템자원을 얻기 위해서 'blocked' 상태로 천이를 하거나, 보다 높은 우선순위를 갖는 쓰레드에 의해서 선점(preemption)이 되거나 같은 우선순위를 갖는 쓰레드가 있을 때, 퀀텀값을 모두 소진하는 경우이다. 따라서 쓰레드의 우선순위를 정의하는 것은 매우 중요한 일이다.

쓰레드 우선순위

우선순위는 반드시 쓰레드가 생성될 때 부여가 되어야 한다. 우선순위기반의 스케쥴러는 부여된 우선 순위을 참조하여 실행 가능한 쓰레드들 중 가장 높은 우선순위를 가진 쓰레드에게 CPU를 할당하기 때문이다. 하지만 실행이 될 때 우선순위를 변경할 수 있다. 이런 동적인 우선순위 부여는 실시간 시스템이 실세계의 환경변화에 곧바로 대응하게 지원한다. J · BOSN 운영체제는 256개의 우선순위 레벨을 제공한다.

레벨0번이 가장 높은 우선순위이고 레벨255가 가장 낮은 우선순위를 가리킨다. 몇 개의 우선순위는 J · BOSN 운영체제의 내부 사용을 위하여 예약(reserved)이 되었다. 레벨255는 J · BOSN의 나노커널이 생성한 "Idle" 쓰레드를 위하여 예약되었다.

레벨0은 J · BOSN의 나노커널에서 특수용도로 사용하기 위해서 예약이 되었다. J · BOSN의 시스템에서 제공하는 마이크로커널의 서버들을 위하여 레벨16부터 레벨31 까지는 예약이 되어있다. 하지만 다른 마이크로 레벨의 서버를 생성하는 사용자는 레벨16부터 레벨31까지 사용할 수 있다. 레벨64에서 레벨95까지는 메크로커널의 서버들을 위해서 사용할 수 있다.

레벨	설명
레벨 0	Reserved for JBOSN RTOS
레벨 1	Do not use
레벨 2 ~ 레벨 15	For real-time use
레벨 16 ~ 레벨 31	For microkernel servers
레벨 32 ~ 레벨 63	For high priority device driver
레벨 64 ~ 레벨 95	For macrokernel servers
레벨 96 ~ 레벨 127	For device drivers
레벨 128 ~ 레벨 254	For applications
레벨 255	Reserved for idle kernel thread

표 1. 우선순위 표

I/O 서버는 레벨80를 사용하고 있고 파일시스템 서버는 레벨72, 윈도우서버는 레벨 68, 네트워크서버는 레벨 76을 사용하고 있다. 디바이스 드라이버가 사용하는 것은 조금 복잡하게 나누어지고 있는데, 실시간성이 매우 요구되는 것은 레벨 2에서 레벨 15까지 사용이 될 수 있는데 매우 제한적으로 사용하기를 권장한다. 나머지 레벌 96 에서 레벨 127은 일반 디바이스 드라이버를 위하여 정의되어 있고, 레벨128부터 레벨254까지는 애플리케이션을 위해서 정의되어 있다.

표 1과 파일 "JBOSN200/OS/INCLUDE/thread.h" 를 참조하기 바란다.

Thread가 'ready' 상태로 천이될 때, J · BOSN 커널은 우선순위 순서대로 쓰레드를 정렬시키고 같은 우선순위가 있다면 바로 뒤쪽에 정열을 한다. 따라서 모든 쓰레드의 삽입이나 제거는 매우 빠르고 일정한 시간 내에 동작이 완료된다. 또한 탐색 루틴이 필요 없어진다.

라운드 로빈(Round-robin)

우선순위를 기반으로 하면서도 J · BOSN 운영체제의 스케

줄러는 Timeslice(Quantum)를 사용한다. 퀀텀 사용하는 라운드-로빈(round-robin) 스케줄링 알고리즘은 같은 우선순위를 가지고 있는 실행 가능한 쓰레드들에게 CPU를 균등하게 할당하여 준다. 만일 이 알고리즘을 사용하지 못하면 우선순위가 같은 여러 개의 쓰레드가 있다면, 현재 실행되고 있는 쓰레드가 스스로 'blocked' 상태로 나가면 CPU를 획득할 수 있지만, 만일 그렇지 않다면 영원히 실행할 기회가 없게 된다.

라운드-로빈 스케줄링은 같은 우선순위의 쓰레드들에게 퀀텀을 이용하여 공정한 CPU 할당을 보장하여준다. 퀀텀은 쓰레드가 생성될 때 기본값이 설정이 되고 실행할 때 퀀텀값을 재 설정할 수 있다. 그림 2에서 시간축은 퀀텀에 의하여 일정한 조각으로 분리가 되고 각 시간 슬라이스는 해당하는 쓰레드들이 CPU 사용을 보장받는 시간이 된다. 보다 정확하게 말을 하면, 쓰레드는 실행시간을 측정할 수 있는 카운터를 가지고 있고, Tick Timer가 발생되면 이 카운터가 감소를 한다. 실행시간 카운터가 주어진 퀀텀값을 기준으로 충분히 감소되었을 때 이 카운터는 초기화되며 쓰레드는 같은 우선순위를 가지고 있는 쓰레드들의 맨 뒤에 삽입이 된다. 다음 쓰레드가 CPU를 할당받아서 실행되고 다시 설명된 과정을 거치게 된다.

만일 위에서 설명한 과정을 반복하다가 중간에 보다 높은 우선순위를 가진 쓰레드가 CPU를 선점하여 실행되면 현재의 실행 카운터는 초기화 되지 않고 저장되었다가 나중에 다시 실행될 때 사용되게 된다.

그림 2는 라운드-로빈 스케줄을 보여주고 있다. 같은 우선

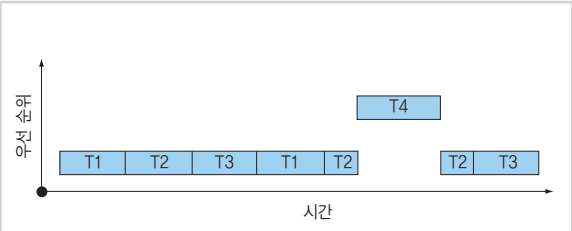


그림 2. Round-robin 스케줄링 예

순위를 가지고 있는 쓰레드 T1, T2, T3과 보다 높은 우선순위를 가지는 T4가 있는데 처음에는 T4가 실행 가능하지 않아서, T1, T2, T3가 교대로 실행이 되고 있다. 그런데 T2가 실행이 되는 중간에 T4가 실행 가능이 되어 T4 선점형 스케줄러에 의하여 CPU를 선점하게 된다. T4가 실행가능에서 대기상태로 이동하면 이전에 실행되던 T2가 다시 실행을 하는데 이전에 실행될 때 남은 퀀텀의 기간만큼만 실행이 허용된다.

일반적으로 실시간 시스템은 timeslice는 거의 사용이 되지 않는다. 다만, 특정 쓰레드가 고의성은 없지만 CPU를 독점하는 것을 막기 위해서이다.

선점형 잠금(Preemption Locks)

J · BOSN 운영체제는 위험하지만 쓰레드의 스케줄링 기능이 동작하지 않도록 설정을 할 수 있다. 이것은 전적으로 프로그래머의 책임으로 하였다. 쓰레드가 스케줄링 기능을 동작하지 못하게 하면 그 쓰레드가 동작할 때, 다른 쓰레드에 의해서 선점이 되거나 퀀텀을 모두 소비하여도 계속해서 동작된다. 그러나 스스로 CPU를 내놓으면 정상적인 스케줄러가 동작 된다. 이 기능은 쓰레드간의 스위칭을 못하도록 CPU를 독점하기는 하지만 하드웨어 인터럽트까지 막지는 못하고 상호배재(mutual exclusion)을 구현하는데 매우 효과적일 수 있다.

다시한번 강조하는데 이 기능을 사용하는 사용자는 상당한 주의를 기울여야 시스템의 성능에 영향을 미치지 않는다.

태스크/쓰레드(Task/Thread) 관리

여기서는 J · BOSN 운영체제의 기본적인 태스크/쓰레드 관리방법을 살펴본다. 일반적인 태스크/쓰레드 관리는 태스크/쓰레드의 동적 생성과 종료 및 그들의 특성을 제어하고 현재 상태정보를 볼 수 있는 도구를 제공한다. J · BOSN은 상당히 다양한 인터페이스를 제공하고 있다.

태스크/쓰레드 생성

태스크나 쓰레드를 생성할 때는 우선순위, 스택을 크기, 스택, 전달하고 싶은 파라미터 등과 같은 태스크/쓰레드가 기본적으로 필요한 값을 전달한다. 커널은 관련된 다양한 자원을 획득, 초기화 및 활성화를 시킨다. 태스크를 생성하면 태스크 구조체와 하나의 쓰레드 구조체가 생성이 되어 하나의 쓰레드를 가지는 태스크가 생성된다.

생성된 쓰레드는 곧바로 활성화되어 실행 가능한 상태로 천이, CPU 자원을 기다린다. 태스크가 생성이 될 때는 이 태스크에 속한 쓰레드의 핸들을 돌려준다. 쓰레드를 생성할 때는 쓰레드에 대한 핸들을 돌려주어, 다음에 이 쓰레드에 접근을 할 때 사용한다. 당연히 태스크/쓰레드는 실행시간에 동적으로 생성될 수 있다.

태스크/쓰레드 종료

태스크/쓰레드는 실행 중 동적으로 소멸될 수 있다. 스스로 종료를 하거나 다른 쓰레드에 의해서 강제로 종료가 될 수 있다. 종료시킬 때는 상당한 주의를 필요로 한다. 생성될 때 할당받은 자원에 관련한 부분은 커널이 스스로 정리를 하여준다. 하지만 현재 사용하고 있는 시스템 자원을 모두 커널이 관리를 할 수 없으므로 종료를 할 때 사용하고 있는 자원을 스스로 초기화나 시스템에 되돌려 주어야 한다.

그렇지 않으면 오류가 발생할 수 있다. 다른 쓰레드가 종료를 시킬 때는 매우 조심스럽고 확실한 상황에서만 종료하는 것을 추천한다. 실제로 실시간 시스템에서는 특수한 에러에 의해 종료가 되지 않는 한 대부분의쓰레드는 생성된 후 종료가 되지는 않는다.

쓰레드 메모리

각 Thread는 자신만의 스택을 가지고 있다. 처음 생성이 될 때 사용하려는 스택의 크기가 지정이 된다. 하지만 나머지의 사용 메모리에 대한 정보(코드와 데이터 영역등)는 커널이 직접 관리를 하지 않는다. 대부분의 어플리케이션은 시스템이 시작하기 전에 이미 롬이나 램메모리에 장착이 되어 있다. 어떤 것은 시작하기 직전에 메모리에 탑재가 된다. 이때는 메모리의 할당과 위치가 매우 중요한 요소가 된다.

Idle 쓰레드

처음 나노커널이 시작할 때 커널 태스크가 생성되고 기본 쓰레드로는 우선순위 레벨255를 갖는 Idle 쓰레드가 생성이 된다. 이 Idle 쓰레드의 역할은 시스템에 실행 시킬 쓰레드가 없을 경우에 CPU를 점유하여 시간을 낭비하는(?) 역할을 하게 된다. 그러나 실시간 시스템이 처리할 일이 없으므로 시스템을 낭비할 수 있는 여지가 있다. 많은 CPU는 단순히 무한 루프를 이용하여 기능을 구현하다.

그러나 어떤 것들은 다양한 기능을 구현할 수 있도록 설계가 되어있다. 이를 대응하기 위해서 J · BOSN은 사용자가 직접 접근할 수 있도록 함수를 지원하고 있다. 사용자는 이 함수를 이용하여 시스템의 낭비(예: 전력소모)를 최소화 할 수 있는 기능을 넣으면 보다 효율적인 시스템을 설계할 수 있다. 단순하지만 이것은 시스템에서 없어서는 안되는 매우 중요한 기능이다. 이 쓰레드는 오직 생성이 될 뿐 소멸되지 않고, 'run', 'ready' 상태만을 천이하게 된다. Real Time