USN 위한 이벤트 중심의 선점형 커널의 디자인과 구현

한상우, 한상은, 김중헌
LG전자 DMP연구소, 성공회대, LG전자 DM연구소

Design and Implementation of the USN kernel with Event-based Preemption

Han Sang Woo, Han Sang Eun, Kim Joong Heon
LG Electronics Inc. SungkongHoe Univ, LG Electronics Inc.

Abstract - The various sensor nodes operating in Ubiquitous Sensor Network environment require the tiny Operating System different from the existing pc-type operating system because of their characteristics. Also Sensor Network operating system needs to support the rapid event handling which sensor node must implement. In this paper, we overcome the drawbacks of the existing sensor network operating system and propose the new kernel which is designed to assist developer to construct event-central operating system entirely. We also evaluate the performance of the super tiny sensor network operating system based on proposed kernel, comparing with that of the existing sensor network operating system.

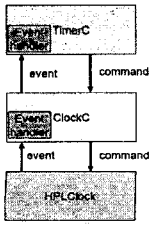
1. 서 론

예전의 컴퓨터의 개념은 계산기에 불과하였다. 그러나 시간이 흘러 컴퓨터는 우리의 생활 어느 곳에서나 존재하는 필수적인 요소가 되었다. 바야흐로 유비쿼터스 세상이 도래하고 있는 것이다. 이런 흐름 안에서 센서 네트워크의 활용도가 커지고 있으며 센서 네트워크의 독특한 특성 때문에 기존의 PC나 임베디드 운영체제와 다른 개념을 갖는 센서 네트워크 용 운영체제가 필요하게 되었다. 기존에 가장 많이 사용되고 있는 UC버클리의 Tinyos는 event-driven 방식으로 동작하고 소스 코드의 크기가 매우 작다. 또한 nesC을 활용하여 최적화된 실행 이미지를 얻을 수 있다. 그러나 작은 소스 코드의 수정에도 전체 시스템을 재컴파일하여 로딩해야 하는 번거로운 단점이 있다. 이를 보완하기 위해 실시간으로 코드를 수정할 수 있는 SOS가 등장하였고, 함수 포인터 테이블을 사용하여 약간의 오버헤드를 갖는다. 그 외에도 MANTIS OS는 초소형 쓰레드에 기반한 멀티 쓰레드 구조를 채택하여 일반 프로그래머에게 익숙한 프로그래밍 환경을 제공하고 있다. 그러나 기존의센서 네트워크 운영체제 중 이벤트가 태스크 스케줄링에 직접 관여하여 선점을 하도록 하는 기능이 없기 때문에 이벤트 처리에 효과적이지 않는 모습을 보이고 있다. 본 논문에서 제안하는 센서 네트워크 운영체제는 센서 네트워크의 데이타 중심적인 특성을 반영하여 이벤트 처리에 효과적으로 대응할 수 있는 구조를 갖고 있다.

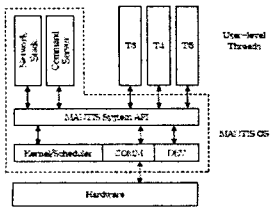
2. 본 론

2.1 기존 USN OS 소개

논 본문에서 살펴볼 기존의 센서 네트워크 운영체제는 tinyos와 mantis os이다. 두 운영체제는 센서 네트워크 용 운영체제라는 공통의 목적을 갖고 있으면서도 가장 극명하게 대립적인 구조를 갖고 있다. 먼저 tinyos에 대해 간략히 설명한다. Tinyos는 event-driven 방식으로 이벤트를 중심으로 구동되는 특성을 지닌다. 이벤트가 없는 동안은 sleep모드로 전환하여 전원 소비를 최소화한다. 태스크는 round robin방식으로 스케줄링을 하며, 먼저 구동되는 태스크가 끝나면 다른 태스크를 수행할 수 있는 yield형 스케줄러를 구현한다. Tinyos의 어플리케이션은 컴포넌트로 이뤄져 있으며 각 컴포넌트가 FSM(Finite State Machine)의 각 state을 상징하고 있다. 각 컴포넌트는 그림 1 와 같이 command와 event을 처리한다. 각 컴포넌트는 다른 컴포넌트에게 COMMAND를 주고 그에 대한 응답으로 EVENT을 받아 처리하는 방식이다.



<그림 1. tinyos의 구조>



<그림 2. mantis os 의 시스템 구조>

그러나 state역할을 하는 컴포넌트 사이에 실제 변이(transition)이 일어나는 것은 아니며 함수의 재사용 의미가 더욱 강하다. 또한 이벤트 사이에 우선순위를 둘 수 없으며, 중요한 이벤트를 선점하여 처리할 수 없다. 두 번째로 mantis os에 대한 설명이다. 콜로라도 대학에서 개발된 멀티 쓰레드 방식의 운영체제로써 프로그래머에게 tinyos보다 친숙한 개발 환경을 제공하고 있다. 또한 레이어 기반으로 되어 있으며, 선점형 스케줄러를 구현한다. 상호배제를 위한 I/O 동기화 함수를 제공하고 디바이스 드라이버로 하드웨어를 추상화시킨다. Mantis os의 경우 PC나 임베디드 운영체제의 모습과 유사하며 그렇기 때문에 오버헤드가 많다. 또한 PC 운영체제의 어플리케이션 모델을 그대로 사용하였기 때문에 센서 네트워크의 데이타 중심 구조와 상이하다.

2.2 이벤트 중심의 선점형 커널 제안

센서 네트워크의 특성은 이벤트가 중심이 되어 모든 처리가 이루어진다는 점이다. 즉, 센서 네트워크 프로그래밍은 프로세스(혹은 프로세서) 중심이 아니라, 데이타 (혹은 이벤트) 중심이다. 그러므로 센서 네트워크 운영체제는 태스크의 동작에 많은 시간을 할애하지 않고, 이벤트를 기다리는 것과 발생한 이벤트를 처리하는 데 많은 CPU 시간을 할애해야 한다. 운영체제는 대부분의 시간을 태스크의 코드가 아니라, 이벤트를 감지하는데 보내야 한다. 또한 이벤트가 발생하면 빠른 루틴을 통해 태스크에게 이벤트를 전달하여 처리하도록 지원해야 한다. 즉, 센서 네트워크에서는 태스크가 매우 수동적인 자세로 이벤트 핸들러의 역할에 충실할수록 좋은 것이다.
본 논문에서 제시한 센서 네트워크 커널은 위에서 살펴본 센서 노드의 특성을 반영하여 이벤트 중심 구조를 갖고 있으며 빠른 이벤트 처리를 지향하고 있다.

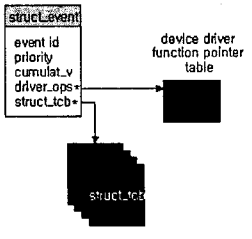
2.2.1 커널의 초기화 과정

```
Process of kernel Initialization
0: h/w initialization start
1: struct_event * CreateEvent (priority)
2: struct_tcb * CreateTask (void (*transition)(void * pData))
3: RegisterDeviceDriverToEvent (struct_event*, void (*
InitDevice)(), void (*IoCtrl)(Command, void* pData))
4: RegisterTaskToEvent (structt_event*, struct_tcb*)
5: kernel start
```

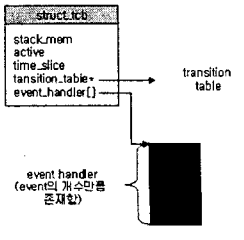
먼저 커널의 초기화 과정을 간략히 정리하면 다음과 같다.

<그림 3. 커널의 초기화 과정>

가장 먼저 하는 일은 이벤트를 정의하고 등록하는 과정이다. 이벤트는 다음과 같은 구조로 설계되었다.



<그림 3. 이벤트 구조체와 연결도>



<그림 4. 태스크 구조체>

그림 3 에서와 같이 이벤트 1개당 이벤트를 감지하고 처리하는데 사용하는 디바이스 드라이버 하나가 함수 포인터 테이블을 통해서 연결되어 있다. 또한 이벤트는 여러 태스크에서 공유하며, 이벤트가 발생하면 발생한 이벤트를 공유하는 모든 태스크의 이벤트 핸들러가 동작하게 된다는 것이다. 중요한 구조체의 상관관계는 표 1에서 정리하였다.

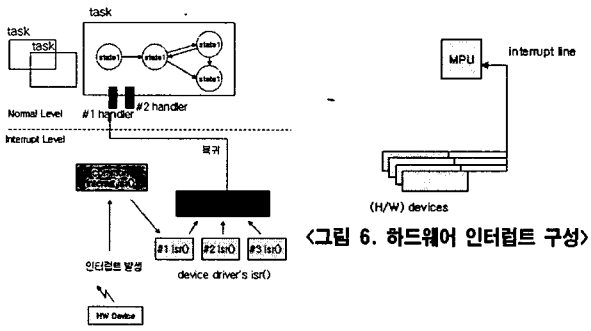
상관관계 A	상관관계 B
이벤트 : 디바이스 드라이버	1 : 1
태스크 : 디바이스 드라이버	N : N

<표 1. 각구조체의 상관 관계>

표 1에서 보듯이, 하나의 태스크는 여러 개의 이벤트를 관리할 수 있으며, 또한 여러 태스크가 같은 이벤트를 공유할 수 있다. 위의 초기화 과정에서 보여 지듯이 디바이스 드라이버와 태스크를 모두 이벤트에 연결시킨다. 다음은 태스크 구조체이다. 그림 4 을 통해 알 수 있듯이 태스크는 다른 선점형 운영체제에서 볼 수 있는 우선순위가 없다. 즉, 태스크는 반드시 이벤트와 연결이 되어야 실행 시간을 가질 수 있으며 실행 시간은 모두 이벤트 핸들러를 동작하거나 transition 테이블 함수를 동작하는데 사용된다. 그림 3 에서 보면 이벤트에는 디바이스 드라이버 관련 함수 포인터 테이블이 있으며 실제 함수는 RegisterDeviceDriverToEvent()를 통해서 실제 디바이스 드라이버에서 제공하는 두 개의 함수(InitDevice(), IoCtrl())와 연결 된다.

2.2.2 이벤트 발생시 처리하는 과정

커널의 초기화 과정을 통해서 등록된 이벤트와 태스크, 디바이스 드라이버는 이벤트가 발생하였을 때 그림 5 과 같은 동작을 통해서 이벤트를 처리한다.



<그림 5. 이벤트 발생시 처리 프로세스>

<그림 6. 하드웨어 인터럽트 구성>

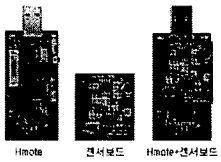
이벤트가 발생하여 디바이스로부터 인터럽트가 발생하면 디바이스 드라이버의 인터럽트 서비스 루틴을 호출하게 되고, 해당 디바이스 드라이버는 이벤트 매니저를 호출하게 된다. 이벤트매니저는 발생한 이벤트를 구분하여 이벤트에 연결된 태스크들의 이벤트 핸들러 위치로 차례대로 복귀하게 한다. 그림 5 에서와 같이 #1번의 이벤트 핸들러가 수행을 마치면 다음 state로 이동하거나 다음 이벤트를 기다리기 위해 대기하게 된다. 그림 5 에서 보듯이, 제안된 커널은 이벤트가 발생하면 그 이벤트를 기다리는 모든 태스크의 핸들러를 수행하여 빠른 반응을 유도하고 있다. 즉, 이벤트가 중심이 되어서 운영되는 구조를 보여주고 있다. 또한 중요한 이벤트에 즉각적인 반응을 일으키기 위해서 각각의 이벤트에 우선순위를 두고, 우선순위가 상대적으로 낮은 이벤트를 처리하는 중간에서 선점이 가능하도록 하고 있다. 그러므로 우선순위가 높은 이벤트는 발생과 동시에 이벤트 핸들러가 다른 태스크를 선점하여 수행할 수 있다. 그림 5 에서 인터럽트가 발생하면 바로 디바이스 드라이버의 인터럽트 서비스 루틴을 호출하는 것이 아니라 중간에 common interrupt service routine을 통과하게 되는데 이것은 MPU의 인터럽트 라인이 제한적이기 때문이다. 제한된 인터럽트 라인을 여러 디바이스가 공유할 경우, 인터럽트가 어떤 디바이스에서 발생했는지 모른다. 그러므로 모든 등록된 디바이스 드라이버의 인터럽트 서비스 루틴을 실행하여 실제 인터럽트가 발생한 디바이스를 찾고 그 디바이스와 연관된 이벤트를 이벤트 매니저에게 알리도록 하는 것이다. (그림 6 참조)

2.2.3 성능 평가

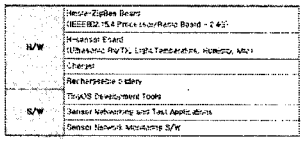
현재 가장 많이 사용되고 있는 tinyos를 비교 군으로 하여 성능 평가를 하고자 한다. 본 논문에서 제안하는 운영체제와 tinyos는 공통적으로 event-driven 방식을 택하고 있으며 선점 가능 여부의 차이가 있다.

2.2.3.1 실험 환경

대표적인 센서 네트워크용 mote를 사용하여 실험을 수행하였다. 관련 사양은 표 2와 그림 9 에서 보여주고 있다. 이벤트는 온도와 빛 두 가지를 주고, 빛이 상대적으로 높은 우선순위라고 가정한다. 상온 섭씨 25도에서 실험하고 이벤트 온도는 섭씨 30도 이며, 빛이 없는 곳에서 실험하였다.



<그림 7. HMOTE>



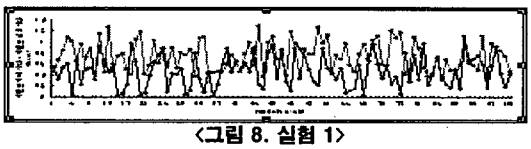
<표 2. 하드웨어 사양>

2.2.3.2 이벤트 반응 비교

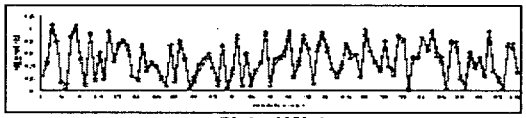
이 실험에서는 두 운영체제가 얼마나 빠르게 이벤트에 반응하고 처리를 완료하는가를 비교 측정하려고 한다.

2.2.3.2.1 이벤트를 준 시각으로부터 이벤트 처리 시작 시각까지의 값비교

두 운영체제에 대해서 무작위로 이벤트를 발생시키고, 이 이벤트 처리를 시작하는 시각까지의 값을 비교하는 실험이다. 이 실험을 통해서 얼마나 빠르게 이벤트에 대해서 대응하는지 측정하려고 한다. USN 환경의 경우, 빠른 처리를 요구하는 이벤트가 다수 존재하기 때문에 실험을 수행하였다.



<그림 8. 실험 1>



<그림 9. 실험 2>

실험 1 에서 tinyos의 값이 상대적으로 큰 값을 갖는 것은 이미 다른 이벤트에 대한 처리를 하고 있으며, 이벤트 처리가 완료된 후에 다른 이벤트 처리를 시작할 수 있기 때문이다. 그러나 제안한 커널은 선점이 가능하므로 빠르게 새로운 이벤트 처리를 수행한다. 여기서 값이 같은 경우는 우선 순위가 높은 이벤트를 처리하는 도중, 하위 우선 순위 이벤트가 발생한 경우이다. 이 경우 선점이 일어나지 않으므로 두 운영체제는 이벤트 처리 시작 시각까지 같은 시간이 걸린다.

2.2.3.2.2 이벤트를 처리하는 데 걸리는 시간을 측정.

이벤트는 빠르게 처리된 후 차후 발생하는 이벤트에 대비해야 한다. 그러므로 가능한 이벤트는 처리는 빠르게 수행되어야 한다. 이벤트를 처리하는데 걸리는 시간은 두 운영체제 모두 비슷하다. 두 운영체제가 모두 유사한 component 기반의 어플리케이션 구조이기 때문이다.

3. 결 론

제안된 커널의 핵심은 이벤트가 커널의 중심에 있다는 점이며, 그렇기 때문에 다른 os보다 빠르게 이벤트에 반응한다는 점이다. 센서 네트워크의 특성은 프로세스 중심이 아니라, 데이타 (혹은 이벤트) 중심 이라고 할 수 있다. 그러므로 대부분의 시간은 이벤트를 기다리는 시간과 이벤트를 처리하는 시간에 할애되어야 한다. 또한 본 논문에서 제안한 커널은 event-driven 방식에서 더 나아가 이벤트가 태스크 스케줄링에 직접 관여하며, 우선 순위가 높은 이벤트에 대한 처리가 기존의 이벤트 처리를 선점하여 수행될 수 있다.

[참 고 문 헌]

[1] Part 15.4: Wireless Medium Access Control(MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)

[2] ason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister, "System Archi- tecture Directions for Network Sensors," ASPLOS 2000, Cambridge, Nov. 2000.

[3] Ralph M. Kling, "Intel Mote: An Enhanced Sensor Network Node," Intel Workshop on Advanced Sensors, Structural Health Monitoring, and Smart Structures, Nov. 2003.