원격 코드 업데이트가 가능한 무선 센서 노드용 운영체제†

(Design of Operating System for Wireless Sensor Nodes with Enhanced Remote Code Update Functionality)

김 창 훈*, 차 정 우**, 김 일 휴**
(Chang-Hoon Kim, Jeong-Woo Cha, and Il-Hyu Kim)

요 약 센서 네트워크 기술은 이러한 센서 노드를 특정한 지역에 대량으로 배치하여 네트워크를 구성한 후 센서 노드를 정보 제공의 매체로 활용하는 기술이다. 센서 노드는 매우 제한적인 하드웨어 자원을 가지고 있기 때문에 효율적인 자원 관리와 센서 네트워크상에서의 다양한 응용 환경을 제공하는 운영체제가 필요하다. 또한 원격 코드 업데이트는 이미 배포된 네트워크에서 응용 프로그램의 오류가 발생하여 이를 수정하거나 성능 개선을 위해 꼭 필요한 기술이다. 본 논문에서는 원격 코드 업데이트가 용이한 새로운 센서 노드용 운영체제, EPRCU (Easy to Perform Remote Code Update)를 제안한다. EPRCU는 이벤트 드리븐 방식의 실행 모델에서 에이징(Aging) 기법을 적용한 우선순위 기반의 프로세스 스케줄링 방식을 사용한다. 작업의 기본 단위인 프로세스는 로더에 의한 동적 메모리 할당 및 프로그램 메모리 관리 기능을 제공함으로써 무선 통신을 이용한 코드 업데이트 수행이 용이할 뿐만 아니라 다양한 센서 네트워크 응용에도 적합하다.

핵심주제어 : 무선 센서 네트워크, 센서 노드용 운영체제, 소프트웨어 업데이트

Abstract Sensor networks monitor the environment, collect sensed data, and relay the data back to a collection point. Although sensor nodes have very limited hardware resources, they require an operating system that can provide efficient resource management and various application environments. In addition, the wireless sensor networks require the code update previouly deployed to patch bugs in program and to improve performance of kernel service routines and application programs.

This paper presents EPRCU (Easy to Perform Remote Code Update), a new operating system for wireless sensor nodes, which has enhanced functionalities to perform remote code update. To achieve an efficient code update, the EPRCU provides dynamic memory allocation and program memory management. It supports the event-driven kernel, which uses priority-based scheduling with the application of aging techniques. Therefore, the proposed operating system is not only easy to perform wireless communication with the remote code update but also suitable for various sensor network applications.

Key Words : Sensor Network, Operating System for Sensor Nodes, Software Update

1. 서 론

최근 정보통신 기술의 발전으로 기존 계산기로서의 컴퓨터는 정보 단말로써 발전하여 우리의 생활에 밀

† 이 논문은 2008학년도 대구대학교 학술연구비 지원에 의해 수행되었음
* 대구대학교 컴퓨터·IT 공학부, 제1저자
** 대구대학교 대학원 컴퓨터정보공학과

접한 영향을 주고 있으며, 정보통신 기술의 진보는 유비쿼터스 컴퓨팅(Ubiquitous Computing)이라는 새로운 정보통신 혁명을 야기하게 되었고, 사회 발전의 흐름과 끊임없이 환경을 인간 친화적으로 바꾸고자하는 인간의 욕구와 맞물려 무선 센서 네트워크(Wireless Sensor Networks)의 필요성이 제기되었다. 무선 센서 네트워크는 각종 센서가 부착되어 있어 센싱이 가능하고 센싱된 정보를 가공할 수 있는 소형 MCU 및 이를 송수신할 수 있는 무선 통신 칩으로 구성된 저가의 소형 장치, 즉 센서 노드로 구성된 네트워크를 의미한다. 센서 네트워크 기술은 이러한 센서 노드를 특정한 지역에 대량으로 배치하여 네트워크를 구성한 후 각종 데이터 수집 및 중요한 이벤트를 감지하도록 하여 센서 노드를 정보 제공의 매체로 활용하는 기술이며, 최근 건축, 과학, 해양 등과 같은 다양한 분야에 적용되고 있다[1]. 그러나 센서 노드는 적은 메모리, 배터리 용량의 제한, 컴퓨팅 성능의 제약 등 제한적인 하드웨어 자원을 가지고 있기 때문에 효율적으로 자원을 관리하고 센서 네트워크상에서의 다양한 응용 환경을 제공하기 위한 센서 노드용 운영체제 연구의 필요성이 강조되고 있다[2-5].

무선 센서 네트워크를 위한 운영체제는 센서 노드의 자원 제약을 극복하고, 다양한 분야의 응용개발을 쉽게 하기 위하여 개발되었다. 초기에는 자원 제약의 극복, 장치사용의 독립성, 유연성 보장 등의 기본적인 목적에 맞춘 소형 운영체제 형태로 개발되었으나 최근에는 기술 상용화에 대한 요구를 충족시키기 위하여 운영체제의 안전성 보장에서부터 보안, 무선 네트워크 기술, 파일 시스템, 저전력 관리 기술 등의 운영체제로서의 전반적인 기능은 물론 사용자 편의 및 대규모 센서 배포를 위한 GUI 툴과 같은 실용화 기술에 대한 연구가 활발하게 진행 중이다.

본 논문은 무선 센서 네트워크에 적용 가능한 효율적인 센서 노드용 운영체제를 설계하는데 그 목적이 있다. 이를 위해서 본 논문에서는 새로운 센서 노드용 운영체제, EPRCU를 제안한다. EPRCU는 이벤트 드리븐 방식의 실행 모델에서 에이징 기법을 적용한 우선순위 기반의 프로세스 스케줄링 방식을 사용하며 프로세스 사이의 상호작용은 동기적 또는 비동기적인 이벤트 전달로 이루어진다. 작업 수행의 기본 단위인 프로세스는 실행 중에 동적으로 로드 및 언로드하는 것이 가능하며 프로세스 로더에 의한 동적 메모리 할당 및 프로그램 메모리 관리 기능을 제공함으로써 무선 통신을 이용한 코드 업데이트 수행이 용이하다.

2. 관련연구

2.1 무선 센서 노드용 운영체제의 개요

본 절에서는 무선 센서 노드용 운영체제가 이벤트 드리븐 모델일 때 멀티 스레드 모델과 비교해서 가지는 장·단점과 원격 소프트웨어 업데이트 시스템의 중요성에 대해서 설명한다.

2.1.1 이벤트 드리븐 시스템

센서 노드와 같이 엄격하게 제한된 메모리 환경에서 멀티 스레드 모델은 상당히 많은 메모리 자원을 필요로 한다. 각 스레드는 자신만의 스택을 가져야하고 각 스택을 위한 메모리는 스레드가 생성될 때 할당되어야 한다. 이는 스레드가 얼마만큼의 메모리를 사용할지 알 수 없는 상황에서 오버플로우를 발생시킬 확률이 매우 높다. 또한 스레드 스택으로 할당된 메모리는 시스템 내에서 병행 스레드(Concurrent Thread)들 사이에서 공유될 수 없으며 공유 자원 접근에 대한 락킹(Locking) 메커니즘을 필요로 한다.

이러한 스레드 스택과 락킹 메커니즘 없이 시스템에 병행성(Concurrency)을 제공하기 위해서 이벤트 드리븐 시스템이 제안되었다[2]. 이벤트 드리븐 시스템에서는 프로세스가 Run-to-Completion한 이벤트 함수로 구현되고 모든 프로세스들이 하나의 동일한 스택을 사용함으로써 부족한 메모리 자원을 효율적으로 공유할 수 있으며 동시에 수행되지 않기 때문에 서로에 대한 락킹 메커니즘 역시 필요하지 않게 된다.

그러나 이벤트 드리븐 시스템은 상태 기반의 프로그래밍 모델을 사용한다. 상태 기반의 프로그래밍 모델은 프로그래머가 다루기 어려울 뿐만 아니라 모든 프로그램이 상태 머신으로 쉽게 표현되는 것은 아니다. 예를 들어 장시간의 계산을 요하는 암호 연산의 경우 CPU를 점유한 채로 오랜 시간 수행하면 시스템

이 외부 이벤트에 반응할 수 없는 상태가 된다.

2.1.2 원격 코드 업데이트 시스템

무선 센서 네트워크는 100에서 1000개 이상의 노드들로 구성되는 큰 규모로 동작할 수 있다. 배포된 노드에 적재된 프로그램은 새로운 소프트웨어로 교체되거나 버그가 발생하여 수정되어야하는 경우가 발생할 수 있으며 이러한 경우에 수많은 노드들을 수거하고 모든 노드들을 다시 프로그래밍 하는 것은 많은 노력이 요구되며 거의 불가능하다고 할 수 있다. 이를 위해서 네트워크를 통해서 새로운 프로그램을 전파하고 프로그램을 업데이트하는 것은 센서네트워크에 있어서 꼭 필요한 기술로 인식되었고 현재까지 많은 연구가 진행되어 왔다[6-10]. 대부분의 운영체제들은 커널 및 시스템 라이브러리, 각종 어플리케이션을 포함하여 시스템 전체를 완전한 이미지 형태로 생성하고 이를 각 장치로 다운로드한다. 시스템 전체 이미지는 어플리케이션 프로세스에 비해서 그 크기가 크고 네트워크를 통해서 전송하는데 있어서 많은 시간과 에너지가 소요된다. 따라서 통신 중에 소비하는 에너지와 시간을 고려하여 전송해야하는 바이트수를 최소로 줄이는 연구는 매우 중요하다[11-14].

2.2 기존의 무선 센서 노드용 운영체제

기존에 개발된 대표적인 센서 노드 운영체제에는 UC Berkeley의 TinyOS, Networked and Embedded Systems Lab of UCLA의 SOS, Swedish Institute of Computer Science의 Contiki, University of Colorado의 MANTIS, University of Virginia의 t-kernel, University of Illinois at Urbana-Champaign의 LiteOS, Carnegie Mellon University의 Nano-RK, 연세대학교 모바일 임베디드 시스템 연구실의 RETOS, 그리고 한국 전자 통신 연구원(ETRI)의 Nano Qplus가 있으며, 각 운영체제의 특징 및 기능성을 요약하여 [표 2.1]에 나타내었다. TinyOS는 다양한 하드웨어 및 시스템 기능을 모듈의 형태로 지원하며 프로그램은 모듈화 된 컴포넌트를 연결하고 컴포넌트 사이에 이벤트를 주고받게 함으로써 전체 기능을 수행하고 이벤트 기반(event-driven), 그리고 단일(monolithic) 커널 센서 운영체제의 특징을 가진다. MANTIS는 콜로라도 대학에서 개발된 멀티 스레드 지원 및 빠른 프로토타이핑을 목표로 개발된 센서 운영체제이다. MANTIS의 스레드는 다섯 단계의 우선순위를 가지고 있으며 선점 가능하여 작은 단위로 스케줄링을 수행할 수 있다. 기본 스케줄링 기법으로 미리 정의된 단위 시간을 기준으로 하는 라운드 로빈 스케줄링을 사용하였다. SOS는 모듈 기반의 센서 운영체제로써 커널을 정적인 모듈과 동적 모듈 두 개의 영역으로 구분하여 개발 및 배포 가능하다. 동적 모듈은 커널과 독립적으로 개발되고 링크되어 시스템이 배포된 후 수정이 비교적 자유롭다는 장점을 가진다. 가상 메모

<표 2.1> 센서 노드 운영체제 특징 및 기능성 비교

	TinyOS	SOS	MANTIS	Contiki	t-kernel	LiteOS	NanoRK	RETOS	Nano Qplus
Model	E	E	T	Pro.T	P.S	T	T	T	T
Module	O Static	O Dynamic	O Static	O Dynamic	X	O Dynamic	X	O Dynamic	O Static
Kernel Size	소	중	대	대	중	대	소	중	중
Realtime	-	-	X	Soft	Hard	Soft	Hard	Soft	Soft
D.Mem Alloc	X	O	O	X	O	O	X	O	O
Protection	O	O	X	X	O	X	O	O	O
Stack	X	O	X	X	O	X	O	O	O
Security	△	△	X	X	X	X	X	X	X
Language	nesC	C	C	C	Any	C, C++	C	C	C

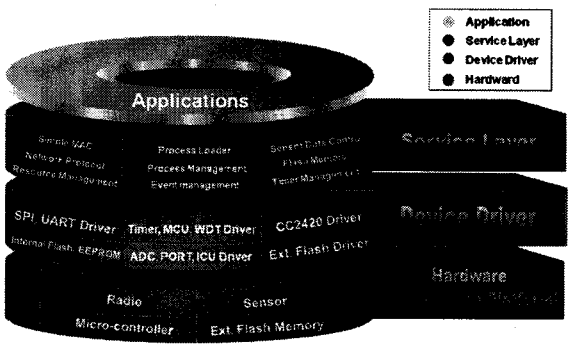
E : Event-driven model
T : Threading model
Pro.T : Proto-Threads
P.S : Preemptive Scheduling

리(Virtual Memory)를 지원하지 않는 센서 네트워크에서 동적 모듈을 지원하기 위해 SOS는 각 모듈을 상대 주소 참조(Position Independent Code)방식을 사용한다. 동적 모듈과 정적 커널은 상호 간 통신을 위해 이벤트를 사용하며 우선순위 기반의 이벤트 스케줄링 기법을 제공한다. Contiki는 멀티태스킹 지원을 위한 센서 운영체제이다. 커널은 이벤트 드리븐 방식이며 응용 프로그램이 실행 중에 동적으로 적재되며 lightweight 스레드를 사용하여 멀티태스킹 응용 개발 환경을 제공한다. 프로세스 단위의 선점형 멀티 스레드와 이벤트를 통한 메시지 전달 방식의 프로세스 간 통신을 지원하며 GUI 시스템을 지원한다.

3. 원격 코드 업데이트가 용이한 무선 센서 노드용 운영체제

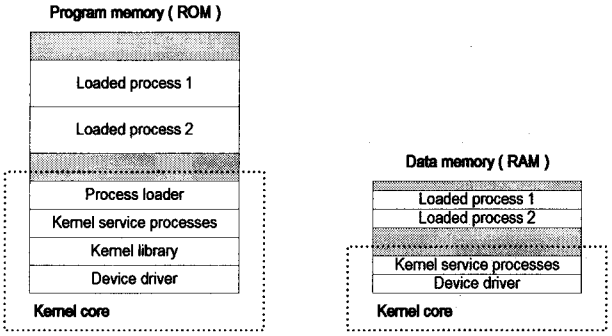
3.1 시스템 개요

EPRCU는 전체 시스템을 <그림 3.1>과 같이 하드웨어 의존정도에 따라 4개(하드웨어, 디바이스 드라이버, 커널 서비스, 어플리케이션)의 논리적인 계층으로 구분한다. 커널 서비스 계층이 어플리케이션 계층에 하드웨어 추상화를 제공하고 있으며 디바이스 드라이버와 어플리케이션이 직접적으로 통신하는 것은 허용하지 않는다.



<그림 3.1> EPRCU의 논리적인 시스템 구성

EPRCU 시스템은 커널 라이브러리, 디바이스 드라이버, 서비스 프로세스, 그리고 로드된 프로세스들로 구성된다. 로드된 프로세스는 어플리케이션 프로그램 또는 커널 서비스일 수 있다. 어플리케이션 프로그램은 하나 이상의 프로세스로 구성될 수 있으며 모든 프로세스는 실행 중에 동적으로 교체되거나 삭제될 수 있다. 프로세스들 사이의 통신은 동기적인 이벤트를 포스팅(Posting)하는 방법을 사용하며 항상 커널을 통해서만 이루어진다. 이벤트는 파라미터와 함께 프로세스에 포스팅되고 프로세스 스레드에 의해서 처리된다. 프로세스의 모든 상태는 15바이트로 구성된 프로세스 제어 블록(Process Control Block)에 저장되고 커널은 프로세스 제어 블록에 대한 포인터만을 유지하고 이를 통해서 프로세스를 관리한다. 단일 스택을 사용하는 EPRCU는 모든 프로세스들이 같은 주소공간을 공유하며 다른 보호 도메인에서 동작하지 않는다.



<그림 3.2> 실행중인 EPRCU 시스템의 메모리 구성

실행중인 EPRCU 시스템은 <그림 3.2>와 같이 커널 코어와 로드된 프로세스영역으로 분할되어 있다. 커널 코어 영역은 디바이스 드라이버, 커널 라이브러리, 커널 서비스 프로세스 그리고 프로세스 로더로 구성되며 하드웨어 구성에 따라 사용되는 코드만을 포함하여 컴파일 된 단일 이미지 형태로 배포된다. 일반적으로 커널 코어는 한번 배포되면 수정이 불가능하며 이를 가능하도록 하기 위해서는 특수한 부트로더 모듈을 필요로 한다.

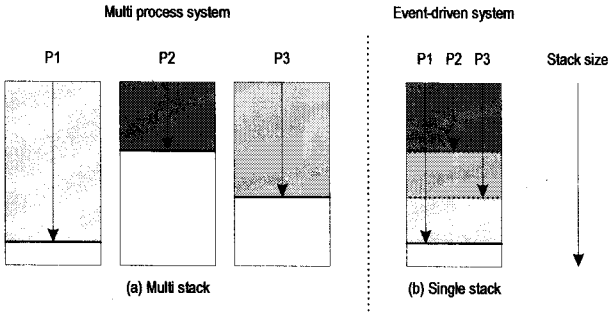
프로세스는 프로세스 로더에 의해서 시스템으로 로드된다. 프로세스 로더는 무선 및 시리얼 통신 그리고 노드에 직접 연결된 외부 플래시 메모리로부터 프로세스 이미지를 얻을 수 있다. 그러나 일반적으로 시스템의 안정성을 고려하여 무선 및 시리얼 통신으로부터 얻은 프로세스 이미지를 코드 메모리에 프로그램하기 전에 외부 플래시 메모리에 먼저 저장하고 외부

플래시 메모리로부터 다시 프로세스 이미지를 얻는 방식을 사용한다.

3.2 커널 구조

EPRCU의 커널은 대기 중인 프로세스에 이벤트를 전달하는 이벤트 스케줄러와 통신 스택을 포함한 커널 서비스 프로세스로 구성된다. 모든 프로세스의 실행은 이벤트 발생에 의해서 동작한다. 커널은 한번 스케줄 된 프로세스를 선점하지 않기 때문에 프로세스 스레드는 Run-to-Completion 하다고 할 수 있다. 그러나 프로세스는 하드웨어 인터럽트에 의해서 선점당할 수 있다.

커널은 동기와 비동기적인 두 가지 이벤트 유형을 제공한다. 비동기적인 이벤트들은 커널이 제공하는 이벤트 큐를 이용하여 발생한 이벤트에 대한 정보를 저장한 후 타겟 프로세스 전달된다. 동기적인 이벤트들은 비동기 이벤트들과는 달리 이벤트 큐를 사용하지 않으며 타겟 프로세스에 직접 전달된다. 이벤트는 프로세스 레벨에서 발생하며 파라미터와 함께 타겟 프로세스로 전달되기 때문에 커널 내에서 프로세스 사이의 데이터 통신을 가능하게 한다.



<그림 3.3> 다중 프로세스 시스템에서 다중 스택(a)과 단일 스택(b)의 메모리 요구 사항

EPRCU 커널에서는 모든 실행 프로세스들이 하나의 동일한 스택을 사용함으로써 부족한 메모리 자원을 효율적으로 공유한다. 또한 비동기 이벤트의 사용은 각 이벤트를 처리하는 프로세스 스레드들 사이에서 스택을 리와인드(rewind) 함으로써 스택 공간의 요구사항과 오버플로우 발생을 줄인다.

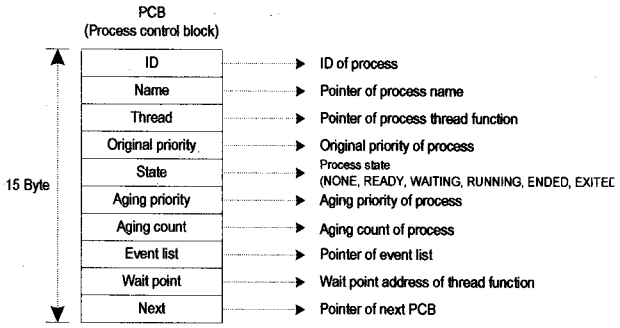
이벤트 드리븐 시스템에서 모든 실행 프로세스들이 단일 스택을 공유하여 사용하면 다중 프로세스 시스템에서 다중 스택을 사용하는 것보다 센서 노드상의 부족한 메모리 자원을 효율적으로 사용할 수 있다. <그림 3.3>은 3개의 프로세스가 존재하는 다중 프로세스 시스템과 이벤트 드리븐 시스템에서 각 프로세스가 자신의 고유한 스택을 가지는 경우 (a)와 하나의 동일한 스택을 공유하는 (b)의 경우를 나타낸 것이다. (a)와 (b) 두 경우 모두 각 스레드가 요구하는 최소 스택 메모리양과 실행 중에 사용하게 되는 최대 스택 메모리양은 동일하다. (a)의 경우 각 프로세스들은 자신만의 고유한 스택 공간을 가지며 서로 독립적으로 동작하기 때문에 특정 시점에서 스택 메모리의 전체 사용량은 각 프로세스가 사용 중인 메모리를 모두 합한 것과 같다. 그러나 (b)의 경우 프로세스는 스택 메모리를 이용하여 작업을 수행 하고 작업을 완료하면 자신이 사용했던 스택 메모리를 반납한 후 다른 프로세스가 작업을 수행하기 때문에 특정 시점에서 스택 메모리의 전체 사용량은 현재 동작중인 프로세스의 스택 메모리 사용량과 동일하다. 즉 기본적으로 (a)가 (b)보다 더 많은 메모리 자원을 필요로 한다. 따라서 극도로 제한된 메모리 공간을 가지는 센서 노드 환경에서는 (a)보다 (b)가 더 적합하다.

3.2.1 프로세스

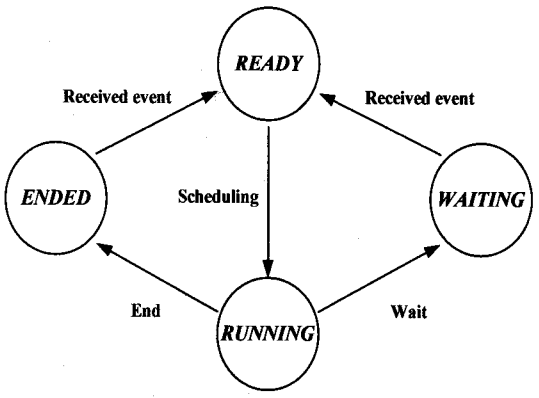
EPRCU의 프로세스는 프로세스 제어 블록과 스레드 함수로 구성된다. 프로세스 제어 블록에는 프로세스의 상태를 비롯하여 실행중인 프로세스에 대한 여러 가지 정보가 저장된다. <그림 3.4>는 EPRCU의 프로세스 제어 블록과 제어 블록을 구성하는 각 필드들을 나타낸 것이다. 제어 블록을 구성하는 각 필드는 초기에 컴파일 시간에 그 값이 결정되고 실행 중에 커널에 의해서 변경될 수 있다.

커널은 프로세스 제어 블록에 대한 포인터를 리스트 형태로 관리하며 실행 중에 프로세스 로더를 이용하여 프로세스를 제거하거나 새로운 프로세스를 추가하는 것이 가능하다.

시스템에 존재하는 프로세스는 총 6가지(NONE, READY, WAITING, RUNNING, ENDED, EXITED) 상태로 존재할 수 있다. 커널과 함께 배포되는 프로세스는 초기에 NONE 상태로 생성되고 프로세스 초기



<그림 3.4> 프로세스 제어 블록



<그림 3.5> 프로세스 상태 천이 다이어그램

화 과정에서 RUNNING 상태가 된다. 초기화가 완료된 프로세스는 WAITING 또는 ENDED 상태로 프로세스 리스트에 존재하게 되며 커널 또는 다른 프로세스로부터 이벤트를 받으면 READY 상태로 전환되어 스케줄링의 대상이 될 수 있다. READY 상태의 프로세스 중에서 스케줄러에 의해서 선택된 프로세스는 RUNNING 상태로 변경되어 쓰레드 함수를 실행한다. 이때 대기 지점의 설정 유무에 따라서 쓰레드 함수의 첫 번째 명령부터 실행할 것인지 아니면 이전에 실행 중이었던 작업을 이어서 실행한다. 실행 중이던 프로세스는 커널로부터 요청된 작업이나 이벤트 처리를 완료하여 ENDED 상태가 되거나 요청된 작업이나 이벤트 처리를 완료하기 위한 조건이 만족되지 못하면 대기 지점을 설정하고 WAITING 상태가 된다. 프로세스는 ENDED 상태가 될 때 대기 지점을 초기화한다. <그림 3.5>는 프로세스 상태 천이 다이어그램을 나타낸 것이다. 프로세스 상태 천이 다이어그램에는 나타나있지 않지만 시스템에 존재하는 모든 프로세스들은 종료(EXIT) 이벤트를 받으면 EXITED 상태로 변경되고 메모리를 비롯하여 사용 중이던 기타 모든 자원들을 시스템으로 반납하고 시스템에서 제거된다.

EPRCU에서는 프로세스 스레드가 수행 중에 작업을 계속 이어갈 수 없을 때 스스로 점유하고 있던 CPU를 반환하고 다른 프로세스가 작업을 수행하도록 할 수 있다. 이때 대기 지점(Wait point)을 저장하고 프로세스가 스케줄러에 의해 다시 선택되었을 때 저장해 두었던 지점부터 작업을 이어가도록 함으로써 함수의 지역적 연속성을 유지할 수 있다.

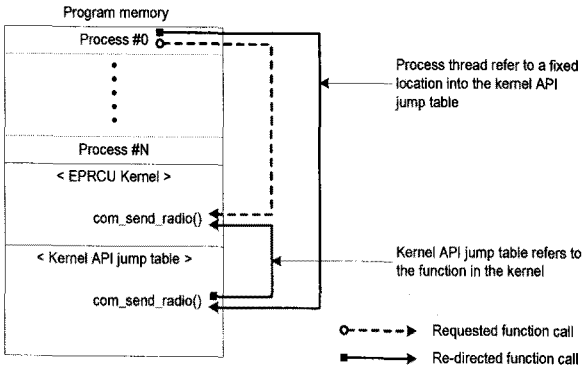
확장 문법을 사용하여 프로세스 스레드 함수의 지역적 연속성을 유지하기 위해서는 스레드 함수의 시작 부분에 대기 지점의 설정 여부에 따라 함수의 처음부터 수행할 것인지 아니면 대기 지점부터 수행할 것인지를 결정하는 RESUME 코드를 삽입해야 한다. 그리고 스레드 함수를 수행하면서 특정 이벤트나 조건 등을 만족하지 못하여 수행 중이던 작업을 계속 이어갈 수 없거나 스스로 CPU를 반환하고 대기 상태로 빠지고자하는 경우 SET 코드를 삽입하여 대기 지점을 설정하도록 해야 한다.

그러나 확장 문법을 이용한 함수의 지역적 연속성을 유지하는 것은 스택의 내용까지는 포함하지 않는다. 즉 함수 내에서 사용되는 변수의 값은 유지되지 않는다. EPRCU는 이 문제를 해결하기 위해서 정적 변수(Static variable)를 사용함으로써 함수가 종료되더라도 변수는 소멸되지 않고 그 값을 유지한다.

실행중인 EPRCU 시스템에는 정적으로 링크된 프로세스와 동적으로 링크된 프로세스가 함께 존재한다. 정적으로 링크된 프로세스는 커널과 함께 컴파일되고 커널 이미지에 함께 포함되어 배포되기 때문에 프로세스 스레드 함수에서 추가적인 작업 없이 커널 API를 사용할 수 있고 커널에서도 일반적인 방법으로 프로세스 스레드 함수를 실행시킬 수 있다.

정적으로 링크된 프로세스와는 달리 동적으로 링크되는 프로세스는 커널과는 별도로 컴파일되고 시리얼 또는 무선 통신을 이용하여 실행중인 노드에 배포된다. 따라서 정적으로 링크된 프로세스와 같이 일반적인 방법으로는 커널 API를 사용하는 것이 불가능하며 실행 중에 노드로 다운로드 된 프로세스가 프로그램 메모리의 어느 위치에 적재될지 알 수 없기 때문에 커널이 프로세스 스레드 함수를 올바르게 호출하기

위해서는 추가적인 작업이 필요하다. EPRCU 커널은 <그림 3.6>과 같이 커널 API 점프 테이블을 이용하여 시스템으로 새롭게 로드된 프로세스에서도 커널 API를 호출할 수 있도록 한다. 또한 커널은 필요에 따라 점프 테이블에 새로운 항목을 추가하거나 기존에 존재하는 항목을 삭제하는 것이 가능하다.



<그림 3.6> 커널 API 점프 테이블의 사용

동적으로 링크되는 프로세스는 실행 중에 재배치될 함수의 코드와 재배치 정보를 포함한 이미지로 구성된다. 프로세스 로더는 재배치 정보를 이용하여 시스템으로 프로세스를 로드하기 위해 필요한 메모리 공간을 할당하고 함수들을 재배치한다.

EPRCU 시스템에서 새로운 프로세스가 원격 코드 업데이트 매니저로부터 실행중인 원격지의 노드로 배포되는 과정은 다음과 같다.

1) 원격 코드 업데이트 매니저는 원격지의 노드들에게 새로운 프로세스를 배포하기 위해서 먼저 알림(ADVERTISEMENT) 메시지를 전송한다. 이 메시지를 받은 노드는 메시지에 포함된 프로세스 정보를 확인하여 시스템에 이미 존재하는 프로세스이면 업데이트된 버전인지 확인하고 새로운 프로세스이면 해당 프로세스를 시스템에 로드할 수 있는 충분한 공간이 있는지 확인한다.

2) 원격지의 노드는 1)의 두 조건 중 한 가지라도 만족되면 매니저에게 프로세스 요청(REQUEST) 메시지를 전송하고 다운로드 대기 상태가 된다. 매니저는 알림 메시지를 일정 횟수 반복적으로 전송한 후 하나 이상의 노드로부터 프로세스 요청 메시지를 받으면 프로세스 이미지 전송을 시작한다. 다운로드 대기 상태에 있는 노드들은 다운로드(DOWNLOAD) 메시지와 함께 전송되는 프로세스 이미지 데이터를 수신하여 외부 플래시 메모리와 같은 2차 저장 공간에 저장한다. 프로세스 이미지는 네트워크 전송에 알맞도록 여러 개의 캡슐(Capsule) 형태로 분할되어 전송 된다. 매니저는 마지막 캡슐을 전송한 후 다운로드 완료(END_DOWNLOAD) 메시지를 전송한다.

3) 원격지의 노드는 다운로드 완료 메시지를 수신하면 모든 캡슐을 빠짐없이 수신하였는지 확인하고 손실되거나 수신하지 못한 캡슐이 존재하면 재요청을 통해서 프로세스 이미지 수신을 완료한다.

원격 코드 업데이트 매니저로부터 새로운 프로세스 이미지 다운로드를 완료한 후 프로세스 로더가 외부 플래시 메모리로부터 실행중인 시스템으로 프로세스를 로드하는 과정은 다음과 같다.

1) 프로세스가 시스템으로 로드될 때 프로세스 로더는 먼저 이미지에 포함된 정보를 바탕으로 프로그램과 데이터 메모리에 충분한 공간이 있는지 확인한다. 두 메모리 중 어느 하나라도 공간이 부족하면 프로세스 로드는 중단된다.

2) 프로세스 로더에 의해서 실행 중에 동적으로 시스템으로 로드되는 프로세스는 프로세스 스레드 함수의 지역적 연속성을 유지하기 위해서 전역 변수나 정적 변수를 사용할 수 없다. 프로세스 이미지에는 해당 프로세스가 수행에 필요한 데이터를 저장하기 위한 메모리 공간의 크기 정보가 기록되어 있다. 프로세스 로더는 프로세스 이미지를 로드할 프로그램 메모리 공간과 데이터 저장에 필요한 메모리 공간을 할당하고 할당된 데이터 메모리의 포인터를 프로세스에 전달한다.

3) 프로세스 로드에 필요한 메모리 공간 할당이 완료되면 본격적으로 프로세스 이미지 로드가 시작된다. 프로세스 로더는 프로세스 이미지에 포함된 재배치 정보를 이용하여 외부 플래시 메모리에 저장된 이미지를 2)에서 할당한 프로그램 메모리에 링크하여 프로세스가 올바른 동작을 수행할 수 있도록 한다.

4) 마지막으로 프로세스 로드가 완료되면 EPRCU 커널은 초기화 이벤트와 함께 프로세스 스레드 함수를 호출하여 프로세스를 초기화한다.

3.2.2 이벤트

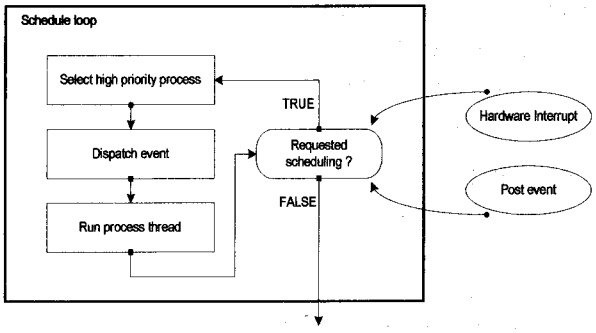
EPRCU의 이벤트는 프로세스와 프로세스 또는 프로세스와 커널 사이의 상호작용을 위한 수단으로 사용된다. 이벤트의 생성 및 전달은 오직 프로세스 레벨에서만 가능하며 특수한 경우(UART 데이터 수신, RADIO 패킷 수신)를 제외하고 인터럽트 레벨에서 이벤트와 관련된 작업의 수행을 제한한다. 또한 이벤트는 커널 또는 프로세스 레벨에서 작업의 시작이나 완료등과 같이 특정한 사건의 발생을 특정 프로세스 또는 커널에 알리기 위한 용도로 사용된다는 점에서 하드웨어 인터럽트와는 구분된다.

커널은 동기와 비동기적인 두 가지 이벤트 유형을 제공한다. 비동기적인 이벤트들은 커널이 제공하는 이벤트 큐를 이용하여 발생한 이벤트에 대한 정보를 저장한 후 타겟 프로세스에 전달된다. 프로세스는 비동기적인 이벤트를 전달받으면 READY 상태로 변경되고 스케줄링을 통해서 발생한 이벤트에 대한 작업을 수행할 수 있다. 동기적인 이벤트들은 비동기 이벤트들과는 달리 이벤트 큐를 사용하지 않으며 타겟 프로세스에 직접 전달된다. 프로세스는 동기적인 이벤트를 전달받으면 RUNNING 상태로 변경되고 스케줄링 없이 바로 이벤트에 대한 작업을 수행한다. 이벤트는 프로세스 레벨에서 발생하며 파라미터와 함께 타겟 프로세스로 전달되기 때문에 커널 내에서 프로세스 사이의 데이터 통신을 가능하게 한다.

3.2.3 스케줄링

EPRCU에서 모든 프로세스에 대한 이벤트 스케줄링은 단일 레벨에서 완료되고 서로 선점할 수 없다. 프로세스 이벤트는 오직 하드웨어 인터럽트에 의해서만 선점될 수 있다.

커널은 명확한 스케줄링 포인트를 위한 플래그를 제공한다. 스케줄링 플래그는 하드웨어 인터럽트의 발생이나 실행중인 프로세스의 이벤트 포스팅으로 인해 설정될 수 있다. 모든 프로세스는 우선순위를 가지며 커널은 이벤트가 발생하거나 스케줄링 플래그가 설정되면 프로세스 리스트에 존재하는 프로세스들의 우선순위를 비교하여 가장 높은 우선순위를 가지는 프로세스를 선택한다. 스케줄러는 선택된 프로세스의 이벤트 리스트에 처리되지 않은 이벤트가 존재하면 프로세스 스레드 함수를 실행할 때 이벤트를 함께 전달한다. 해당 프로세스에 대기 중인 이벤트가 존재하거나 프로세스 리스트에 대기 중인 프로세스가 존재하면 스케줄링 플래그를 설정하여 스케줄링 루프를 빠져나가지 않도록 한다. <그림 3.7>은 EPRCU 프로세스 스케줄러의 구조를 나타낸 것이다.



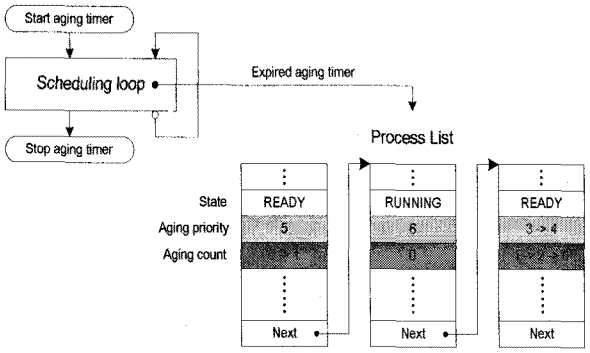
<그림 3.7> 프로세스 스케줄러 구조

3.2.4 에이징

에이징 기법은 SJF(Shortest Job First) 또는 우선순위 기반의 스케줄링 기법을 사용하는 시스템에서 특정 프로세스의 우선순위가 낮아 스케줄러에 의해 선택되지 못하고 무한정 기다리게 되는 프로세스 기아(Starvation)를 방지하기 위하여, 한번 양보하거나 기다린 시간에 비례하여 일정 시간이 지나면 우선순위를 한 단계씩 높여 가까운 시간 안에 작업의 수행을 시작할 수 있도록 하는 것이다.

EPRCU는 이벤트 드리븐 방식의 실행 모델에서 우선순위 기반의 프로세스 스케줄링 기법을 사용하지만 에이징 기법을 적용하여 우선순위가 낮은 특정 프로세스가 무한 연기 또는 기아 상태에 빠지는 것을 방지한다. 이를 위해 각 프로세스는 제어 블록 내에 에이징된 우선순위와 에이징 카운트를 저장하는 변수를 가진다. 프로세스가 READY 상태로 전환될 때 에이징 우선순위는 해당 프로세스의 기본 우선순위 값으로 설정되고 에이징 카운터는 0으로 초기화된다. <그림 3.8>과 같이 에이징 타이머는 스케줄링 루프로 들어가기 전에 활성 상태가 되고 커널이 지정한 시간이 만료되면 프로세스 리스트에서 READY 상태로 존재하는 모든 프로세스들의 에이징 카운트를 1씩 증가시키고 그 카운트 값에 따라 우선순위를 조절한다. 커널

은 프로세스 리스트에 READY 상태의 프로세스가 더 이상 존재하지 않으면 스케줄링 루프를 빠져나오게 되며 이때 에이징 타이머를 비활성 상태로 변경한다.



<그림 3.8> 에이징 타이머를 이용한 프로세스 에이징

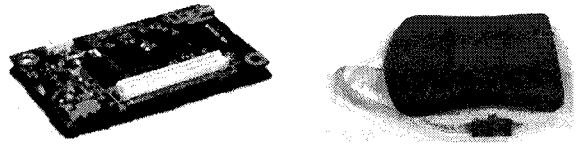
3.2.5 파워 절약 모드

센서 네트워크에서 시스템에 더 이상 수행할 작업이 없거나 네트워크가 유휴상태일 때 센서 노드의 에너지 소비를 줄이기 위해서 CPU를 저전력 모드로 전환하는 것이 필요하다. 노드의 에너지 소비량은 어플리케이션과 네트워크 프로토콜에 따라 다를 수 있다.

EPRCU는 더 이상 시스템에 이벤트 스케줄러가 처리해야할 이벤트가 존재하지 않으면 스케줄링 루프를 종료하고 저전력 모드로 전환한다. 저전력 모드 상태의 CPU는 인터럽트에 의해서 깨어날 수 있으며 시스템은 다시 스케줄링 루프를 시작하여 발생한 이벤트를 처리함으로써 센서 노드상의 제한된 에너지 자원을 효율적으로 사용할 수 있다.

4. 성능 분석

본 절에서는 새로운 무선 센서 노드용 운영체제인 EPRCU를 구현하고 UTRC(Ubiquitous Technology Research Center)의 SN100 센서 보드와 AVR JTAG ICE mk II를 이용하여 테스트 및 성능 평가를 수행하였다. <그림 4.1>과 <표 4.1>은 테스트에 사용된 SN100 센서 노드의 하드웨어 구성을 나타낸 것이다.



<그림 4.1> UTRC SN100 센서 노드와 AVR JTAG ICE mk-II

<표 4.1> UTRC SN100 센서 노드의 하드웨어 구성

Mote Hardware Platform		UTRC SN100
MCU	Chip	ATmega128L
	Type	8MHz, 8bit
	Program Memory (kB)	128
	SRAM(kB)	4
RF Transceiver (Radio)	Chip	CC2420
	Radio Frequency (MHz)	2400
	Max. Data Rate (kbits/sec)	250
	Antenna Connector	MMCX
Flash Data Logger Memory	Chip	AT45DB014B
	Connection Type	SPI
	Size(kB)	512
Default power Source	Type	AA, 2x
	Typical capacity (mA-hr)	2000
	3.3 V booster	N/A

<표 4.2>에는 프로세스 사이의 상호작용과 커널 및 프로세스의 수행을 위한 주요 API에 소요되는 CPU 사이클을 나타내었으며 [표 3.8]에는 프로세스 로더에 의해서 수행되는 메모리 관리 API에 소요되는 CPU 사이클을 나타내었다.

<표 4.2> EPRCU 주요 커널 API에 소요되는 CPU 사이클(1)

Communication method	Clock cycles
Post synchronous event	54
Post asynchronous event	186
Dispatch event from scheduler	137
Call using kernel jump table	12
Direct function call	4

<표 4.3> EPRCU 주요 커널 API에 소요되는 CPU 사이클(2)

Memory management	Clock cycles
Allocate data memory	64
Deallocate data memory	92
Allocate code memory	88
Deallocate code memory	109
Change memory owner	48
Check memory crash	1124

<표 4.4> 3.9 원격 코드 업데이트 기능을 포함하는 운영체제의 메모리 사용량 비교

Platform	Program memory (ROM)	Data memory (RAM)
SOS	20464 B	1163 B
TinyOS with deluge	21132 B	597 B
Bombilla virtual Machine	39746 B	3196 B
EPRCU Core	**11566 B**	**857 B**

또한 본 논문은 기존에 개발된 운영체제 중 원격 코드 업데이트 기능을 포함하는 시스템과 제안된 시스템의 메모리(ROM, RAM) 사용량을 측정하여 그 결과를 <표 4.4>에 나타내었다. 비교 결과 EPRCU는 기존에 개발된 시스템에 비해서 약 2~3배적은 프로그램 메모리와 1K 미만의 데이터 메모리만으로도 원격 코드 업데이트 기능을 수행할 수 있다는 것을 확인하였다.

5. 결 론

무선 센서 네트워크를 위한 운영체제는 자원 제약적인 센서 노드 환경에서 효율적으로 자원을 관리하고 센서 네트워크상에서의 다양한 응용 환경을 제공해야할 뿐만 아니라 이미 배포된 네트워크에서 응용 프로그램의 오류가 발생하여 이를 수정하거나 성능 개선을 위해서 무선 네트워크를 이용한 안전한 코드 업데이트 기능도 제공해야 한다. 이를 위해서 본 논문에서는 원격 코드 업데이트가 용이한 새로운 센서 노드용 운영체제, EPRCU를 제안하였다.

EPRCU는 이벤트 드리븐 방식의 실행 모델에서 우선순위 기반의 프로세스 스케줄링 방식을 사용하였다. GCC C 컴파일러의 labels-as-values 확장 문법을 이용하여 프로세스 스레드 함수의 지역적 연속성을 유지하였고 이를 이용하여 프로세스 스스로가 CPU를 양보하고 다음 스케줄링을 통해서 긴 작업을 계속 이어가도록 함으로써 단일 공유 스택을 사용하는 이벤트 드리븐 시스템의 장점을 유지하고 단점을 보완하였다. 또한 에이징 기법을 적용하여 우선순위 기반의 스케줄링 기법을 사용하는 시스템에서 우선순위가 낮은 특정 프로세스가 스케줄러에 의해 선택되지 못하고 무한 연기 또는 기아 상태에 빠지는 것을 방지하였다.

프로세스 로더는 작업 수행의 기본 단위인 프로세스를 실행 중 동적으로 시스템에 로드 및 언로드하는 것이 가능하며 새롭게 시스템으로 로드 또는 언로드되는 프로세스를 위한 데이터 메모리와 프로그램 메모리 관리 기능을 제공한다. 이는 무선 네트워크를 이용한 원격 코드 업데이트의 수행을 용이하도록 할 뿐만 아니라 원격 코드 업데이트를 수행할 때 새롭게 배포하고자 하는 어플리케이션 프로세스 이미지만을 전송하도록 하여 무선 네트워크를 통한 코드 업데이트에 소요되는 에너지와 시간을 절약할 수 있도록 하였다.

따라서 본 논문에서 설계하고 구현한 EPRCU는 안정적인 시스템을 유지하면서 센서 노드의 부족한 자원을 효율적으로 관리하고 시스템 유지 보수를 위한 효율적인 원격 코드 업데이트 기능을 지원함으로써 무선 센서 네트워크의 다양한 응용 플랫폼을 제공하는 센서 노드용 운영체제로서 적합하다 할 수 있다.

참 고 문 헌

[1] R. Kay, F. Mattern, "The Design Space of Wireless Sensor Networks." IEEE Wireless Communications, Vol. 11, Issue: 6, December,

2004, pp. 54-61.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System Architecture Directions for Networked Sensors," In Architectural Support for Programming Languages and Operating Systems, pp. 93-104, 2000.

[3] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms," ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks, Vol. 10, No. 4, Aug. 2005.

[4] 최석원, 신효정, 윤찬민, 최학수(Hak Soo Choi), 차호정, "무선 센서 네트워크 운영체제", 한국정보과학회, 정보과학회지 제25권 제12호, 2007. 12, pp. 25-34

[5] 민홍, 이상호, 구본철, 허준영, 김용태, 조유근, "무선 센서 네트워크를 위한 운영체제들의 성능분석", 한국정보과학회, 한국정보과학회 2006 한국 컴퓨터 학술종합학술대회 논문집(A), 2006. 6, pp. 331-333

[6] C.-C. Han, R. Kumar, R. Shea, and M. Srivastava, "Sensor Network Software Update Management: A Survey," International Journal on Network Management, Vol. 15, No. 4, pp. 283-294, July 2005.

[7] 나재훈, 채기준, 정교일, "센서 네트워크 보안 연구 동향", 전자통신동향분석, Vol. 20, No. 1, pp. 112-122, Feb. 2005.

[8] 김신효, 강유성, 정병호, 정교일, "u-센서 네트워크 보안 기술 동향", 전자통신동향분석, Vol. 20, No. 1, pp. 93-99, Feb. 2005.

[9] K. Akkaya and M. Younis, "A Survey of Routing Protocols in Wireless Sensor Networks," Elsevier Ad Hoc Network Journal, Vol 3/3, pp. 325-349, 2005.

[10] Crossbow Technology Inc., "Mote In-Network Programming User Reference," 2003.

[11] T. Stathopoulos, J. Heidemann, and D. Estrin, "A Remote Code Update Mechanism for Wireless Sensor Networks," Tech. rep. CENS-TR-20, UCLA, Center for Embedded Networked Computing, Nov. 2003.

[12] J. W. Hui and D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," Proc. of the 2nd International Conference on Embedded Networked Sensor Systmes (SenSys '04), pp. 81-94, 2004.

[13] S. S. Kulkarni and L. Wang, "MNP: Multihop Network Programming Service for Sensor Networks," Proc. of the International Conference on Distributed Computing Systems (ICDCS '05), pp. 7-16, 2005.

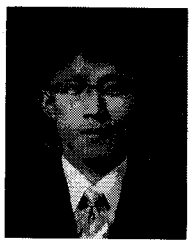
[14] P. Levis and D. Culler, "Maté: A Tiny Virtual Machine for Sensor Networks," Proc. 10th Int'l. Conf. Architectural Support for Programming Languages and Operating Systems., pp. 85-95, Oct. 2002.

[15] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, D. Culler, "The nesC Language: A Holistic Approach to Network Embedded Systems," In Proc. of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation(PLDI), San Diego, CA, June 2003.

[16] C. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, I. Stoica, "A Modular Network Layer for Sensornets," In the Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06), Seattle, WA, November 2006, USENIX Association, pp. 249-262.

[17] C. Han, R. Rengaswamy, R. Shea, E. Kohler, M. Srivastava, "SOS: A dynamic operating system for sensor networks,"In Proc. of the Third International Conference on Mobile Systems, Applications, And Services(Mobisys),

Seattle, WA, June 2005.
[18] A. Dunkels, B. Grönvall, T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," In Proc. of the First IEEE Workshop on Embedded Networked Sensors (EmNets), Tampa, Florida, November 2004.
[19] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, I. Stoica, "A Unifying Link Abstraction for Wireless Sensor Networks," In Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems(SenSys), November 2-4, 2005.

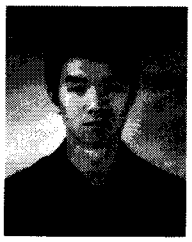


김 창 훈 (Chang-Hoon Kim)

- 정회원
- 대구대학교 컴퓨터정보공학부공학사
- 대구대학교 컴퓨터정보공학과 공학석사
- 대구대학교 컴퓨터정보공학과 공학박사
- 대구대학교 컴퓨터·IT 공학부 조교수
- 관심분야 : 암호시스템, Embedded System, RFID/USN 보안

차 정 우 (Jeong-Woo Cha)

- 정회원
- 대구대학교 멀티미디어공학부공학사
- 대구대학교 컴퓨터정보공학과 공학석사
- 대구대학교 컴퓨터정보공학과 박사과정
- 관심분야 : 암호시스템, Embedded System



김 일 휴 (Il-Hyu Kim)

- 정회원
- 대구대학교 멀티미디어공학부공학사
- 대구대학교 컴퓨터정보공학과 공학석사
- 대구대학교 컴퓨터정보공학과 박사과정
- 관심분야 : 6LoWPAN, 무선 센서 네트워크

논 문 접 수 일 : 2011년 01월 27일
1차수정완료일 : 2011년 03월 10일
2차수정완료일 : 2011년 03월 15일
게 재 확 정 일 : 2011년 03월 22일