센서 네트워크용 실시간 운영체제의 설계 및 구현

Design and Implementaion of Real-Time Operating System for Sensor Networks

강희성, 전상호, 정근재, 이숭열, 김용희, 이철훈
충남대학교 컴퓨터공학과

Hui-Sung Kang(hskang@cnu.ac.kr), Shang-Ho Jeon(shjun@cnu.ac.kr),
Geun-Jae Jeong(gjjeong@cnu.ac.kr), Soong-Yeol Lee(sy-lee@cnu.ac.kr),
Yong-Hee Kim(yonghee@cnu.ac.kr), Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

최근 들어 마이크로컨트롤러가 물리적인 환경을 정교하게 제어하고 감시하기 위해서 센서 네트워크에 사용되고 있다. 응용프로그램이 더욱 더 정교해짐에 따라 설계와 개발과정이 복잡하게 되었고 그 결과로 복잡성을 제어하고 코드의 호환성을 위한 추상화를 제공해주기 위해서 운영체제가 필요하게 되었다. 본 논문에서는 센서 네트워크를 위해 설계된 저전력 실시간 운영체제, UbiFOS-USN을 소개하고,. 센서 네트워크에서 일반적으로 사용되는 초소형, 저전력 마이크로 컨트롤러에 적합한 UbiFOS-USN의 특징에 대해서 기술한다. 실험 결과를 통해서, UbiFOS-USN이 시스템 성능과 메모리 요구사항 측면에서 센서 네트워크에 효율적임을 보여준다.

■ 중심어 : | 유비쿼터스 센서 네트워크 | 실시간 운영체제 | 동적 전원 관리 |

Abstract

Recently microcontrollers are being used in sensor networks to handle sophisticated control and monitoring activities. As applications become more sophisticated, their design and development processes become more complex which consequently necessitates the use of an operating system to manage the complexity and provide an abstraction for portability of code. This paper presents a Low-power real-time operating system, called UbiFOS-USN, designed for sensor networks. We present some of the features that make UbiFOS-USN appropriate especially for small, low-cost microcontrollers typically found in sensor networks. Through experimental results, we show that UbiFOS-USN is quite efficient for a sensor network, both in terms of system performance and memory requirement.

■ keyword : | Ubiquitous Sensor Network | Real-Time Operating System | Dynamic Power Management(DPM) |

I. 서 론

오늘날 무선 센서 네트워크는 새롭고 매우 중요한 연구 분야로 주목 받고 있다. 센서 노드는 사무실, 공장 및 그보다 혹독한 환경에서 주변 상황을 모니터링하고 필요한 정보를 수집하는 용도로 사용되었다. 최근 센서

* 본 연구는 정보통신부의 선도기반기술개발사업의 지원으로 수행되었습니다.

접수번호 : #061113-001
접수일자 : 2006년 11월 13일
심사완료일 : 2006년 12월 29일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

노드 제조 기술의 발달로 저전력, 저비용으로 다양한 기능을 지원하는 노도들이 개발되고 있으며, 홈 네트워크, 자연 환경, 건강관리 및 다양한 종류의 임베디드 시스템 등 센서 노드가 사용되는 분야도 점점 넓어지고 있다. 센서 노드의 활용 분야가 다양해짐에 따라 자원을 효율적으로 관리하는 것과 응용 프로그램의 개발 속도를 향상 시키는 것이 중요한 이슈가 되고 있다. 이를 위해 다양한 플랫폼을 지원하고 자원을 효율적으로 관리하며, 응용 프로그램을 개발할 수 있는 환경을 제공하는 무선 센서 네트워크를 위한 운영체제들이 연구되고 있다[1][2].

무선 센서 네트워크의 운영체제는 작은 코드의 크기, 자원의 효율적인 이용, 표준 API 제공, 적은 에너지 소비, 쉬운 프로그래밍을 고려하여 설계하여야 한다. 또한, 센서 네트워크 응용 소프트웨어가 탑재된 장치에 정보를 제공해줄 수 있어야 그 의미가 있다. 따라서 작업의 시간 제약조건을 만족 시켜주는 것은 센서 네트워크를 위한 운영체제의 중요한 요소라고 할 수 있다[3].

이러한 요구사항들을 충족시키기 위해서 기존의 운영체제를 축소시켜 개발하거나 센서 노드에 맞춘 설계를 통한 운영체제의 연구가 진행이 되었다. 하지만 작은 코드의 크기, 자원의 효율적인 이용, 적은 에너지 소비 등의 기본 요구사항들을 충족시켰지만 실시간성에 대한 요구사항을 충족시키지 못하고 있다.

본 논문에서는 앞서 보았던 조건들을 만족하는 센서 네트워크를 위한 초소형, 저전력 실시간 운영체제인 UbiFOS-USN(Ubiquitous Flexible real-time OS for USN)을 제안한다.

II. 관련 연구

1. 센서 네트워크를 위한 운영체제

1.1 특 징

다양한 응용 분야에서 활용되는 센서 노드는 대부분 사물에 내장되기 때문에, 일반적으로 배터리로 동작하며 그 크기가 작고 전력 소모가 적어야 한다. 그리고 외부의 환경 변화에도 가능한 영향을 받지 않아야 하고, 센서 노드에 탑재되는 운영체제는 제한된 메모리와 CPU 자원을 최대한 활용할 수 있어야 하며 노드에 할당된 작업들을 처리하고 네트워크 내에서의 통신이 원활하게 수행될 수 있도록 보장해야 한다. 센서 노드에 있어서 운영체제는 다음과 같은 기능이 요구된다.

- 제한된 메모리와 CPU 자원을 가지는 센서 노드에 적합한 태스크 구조
- 센서 노드 내부 및 센서 노드간의 효율적인 에너지, 센서 네트워크 응용에 적합한 메시지 처리 기능
- 표준화된 HAL 및 디바이스 드라이버 기능
- 저전력 관리 기능
- 멀티 홉(Multihop), 스타(Star), 메시(Mesh) 네트워크 지원
- 표준 센서 노드 운영체제 API 기능

1.2 센서 네트워크를 위한 운영체제 소개

(1) TinyOS

미국 버클리 대학에서 개발된 TinyOS는 이벤트 발생에 의한 상태 천이 방식을 채택한 머신 기반의 프로그래밍 개념을 사용한 운영체제로서 제한된 메모리 공간의 효율적인 이용 및 프로세싱의 동시성 등을 지원한다. TinyOS는 가상 머신을 이용하여 응용프로그램을 수행함으로 에너지 측면에서 비효율적이며, Mate의 구조와 Instruction Set을 이용하여 프로그래밍 하기 때문에 프로그래머가 접근하기 어렵다. 또한 운영체제의 실시간성에 대한 고려 없이 설계되었기 때문에 작업의 시간 제약 조건을 보장해 주지 못한다는 단점이 있다[7].

(2) MANTIS

미국 콜로라도 대학에서 개발된 MANTIS는 쓰레드 기반의 센서 네트워크 운영체제로 선점 가능한 다중 쓰레드 작업을 지원한다. 또한 복잡한 작업을 수행하는 태스크와 시간 제약성을 갖는 태스크가 적절하게 상호 배치될 수 있도록 스케줄링 한다. 다시 말해서 긴 수행 시간을 필요로 하는 작업이 빠른 시간 내에 수행을 마쳐야 하는 작업의 수행을 가로막지 않도록 해준다. 이

벤트 드리븐 방식의 TinyOS는 이러한 문제를 해결하지 못하지만 MANTIS는 멀티 쓰레드라는 개념을 사용해서 이를 해결하고 있다[8].

(3) SOS

미국 UCLA에서 개발된 SOS 커널은 메시지 처리, 동적 메모리 할당, 모듈의 로드와 언로드, 그 밖에 다른 서비스를 제공하는 부분으로 구성되어 있다. 모듈은 네트워크를 통해서 동적으로 추가, 수정 및 제거가 가능하다. 그러나 모듈의 수행이 독립적으로 이루어지고, 이를 전체적으로 관리하는 실시간 스케줄러가 없기 때문에 실시간성을 지원하지 않는다[9].

(4) NanoQplus

한국전자통신연구원에서 2005년 개발된 Nano-Qplus는 센서 네트워크의 두 가지 요구조건을 만족시키기 위해 설계되고 개발되었다. 첫 번째는 확장성과 시스템 재구성의 용이성이다. NanoQplus에서는 다양한 하드웨어를 NanoHAL(Hardware Abstraction Layer)를 통해 동일한 시스템 모델로 간주한다. 따라서 프로그래머가 하드웨어에 대한 고려 없이 의도하는 대로 프로그램을 쉽게 작성할 수 있다. 두 번째는 쓰레드 기반의 우선순위 스케줄러이다. MANTIS와 같이 우선순위에 따라 선점 가능한 스케줄러를 사용하기 때문에 실시간성을 지원한다[10].

2. DPM(Dynamic Power Management: DPM) 기법

IBM과 MotaVista Software에서 제안한 DPM 기법은 소비전력이 가장 큰 CPU의 frequency 뿐만 아니라 Bus Clock 까지 조절하여 전체 시스템의 소비전력을 감소시키는 전력 관리 기법으로, 시스템에 탑재되는 애플리케이션과 각 디바이스에서 요구하는 Bus Clock 까지 고려하여 소비전력을 감소시키는 융통성을 갖춘 전력 관리 기법이다[4][5].

2.1 DPM 구조모델

시스템 디자이너는 정책 관리자(Policy Manager)를 통해 시스템에서 제공하는 CPU Frequency와 Bus Clock에 맞춰 각 애플리케이션이 요구하는 CPU Frequency와 Bus Clock의 집합을 정의한 정책들(Policies)을 생성한다.

[그림 1]은 커널에서 제공하는 DPM 관련 API를 통해 정책을 결정하고 각 응용프로그램에 미리 정의된 정책을 할당하여 수행하는 DPM 구조모델이다.

2.2 DPM 구동정책(Operating Policies)

IBM과 MontaVista Software에서는 IBM PowerPC 405LP 보드를 사용하였는데, 최대 266MHz에서 33MHz까지 지원하는 하드웨어를 통해 5단계의 구동정책을 제안하였다.

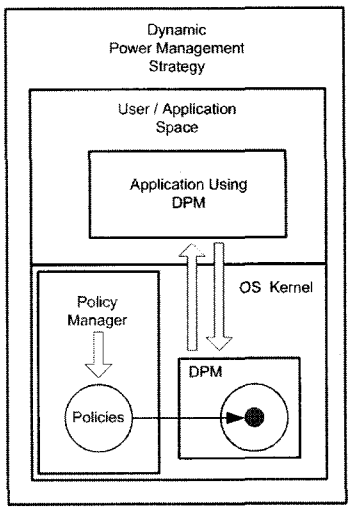


그림 1. DPM 구조모델

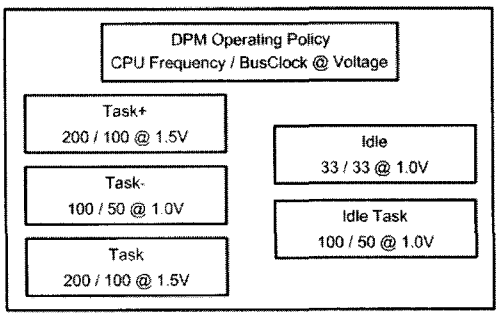


그림 2. DPM 구동정책

[그림 2]를 보면, 5단계(Task, Task+, Task-, Idle, IdleTask)로 구분하여, 각 단계별로 CPU Frequency, Bus Clock과 이에 따른 전압 레벨을 집합으로 하는 구동정책을 정의하였다.

2.3 DPM 동작상태(Operating States)

[그림 3]은 DPM 동작 상태로, 각 태스크는 자신에게 할당된 정책에 따라 동작하며, DPM 스케줄링 API에 의해 할당된 정책을 변화시킬 수 도 있다.

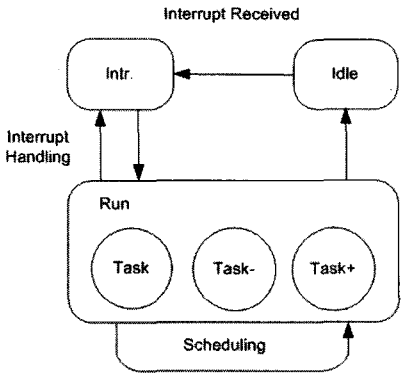


그림 3. DPM 동작 상태

III. 센서 네트워크를 위한 실시간 운영체제 설계 및 구현

본 논문에서 구현한 실시간 운영체제(UbiFOS-USN)에서는 멀티태스킹 환경의 우선순위 기반 선점형 실시간 운영체제이다. 태스크는 쓰레드(Thread) 기반으로 동작되는 구조로, 공통의 작업 영역을 자유롭게 접근할 수 있다. 그리고 각각의 태스크는 중요도에 따라 우선순위가 부여되고 가장 높은 우선순위의 태스크가 CPU를 선점하여 수행한다. [그림 4]는 구현한 실시간 운영체제의 전체 구성을 나타낸 것이다. 크게 핵심이 되는 커널 부분과 각종 센서들을 위한 드라이버, 하드웨어들을 제어하고 추상화하는 HAL(Hardware Abstraction Layer) 부분, 센싱과 구동 그리고 센서 네트워크의 핵심이 되는 통신 프로토콜 스택으로 이루어져 있다.

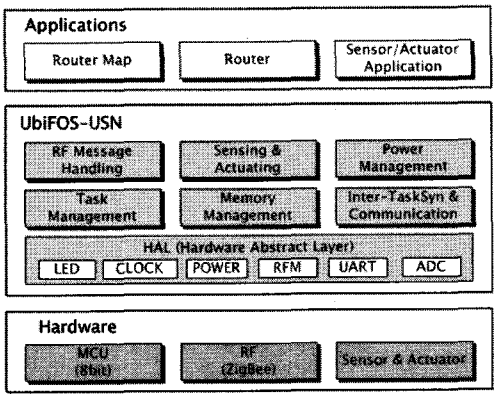


그림 4. UbiFOS-USN의 구조

1. UbiFOS-USN 커널

본 논문에서 구현한 UbiFOS-USN 커널의 첫 번째 특징은 멀티쓰레드를 지원한다는 점이다. 이벤트 드리븐(Event-Driven) 형식으로 동작하는 다른 센서 네트워크 운영체제와는 다르게 멀티쓰레드와 관련된 함수들을 제공한다. 또한 실시간 지원의 핵심이라 할 수 있는 태스크간의 선점(Preemption)을 지원하고 빠른 응답성을 보장한다.

1.1 태스크 관리(Task Management)

일반적으로 응용프로그램은 독립적인 여러 개의 프로그램으로 구성된다. 이러한 독립적인 프로그램 각각을 태스크(Task)라고 하며, 각 태스크들은 자신만의 실행 함수, 스택 영역, 우선순위, 타임슬라이스 등을 할당받는다. 하나의 CPU에서 여러 개의 태스크가 동시에 수행될 수 있는 멀티태스킹 환경을 제공하므로 CPU, 메모리 등과 같은 시스템의 자원을 서로 공유한다. 따라서 여러 태스크들이 동시에 공유자원을 접근할 때의 문제점을 해결하는 상호 배제 접근(Mutual Exclusion Access) 및 동기화(Synchronizing)의 방법을 제공해야 한다.

1.1.1 멀티태스킹

멀티태스킹은 하나의 CPU에서 여러 개의 태스크가 동시에 수행될 수 있도록 하는 방식이다. UbiFOS-USN 커널은 이러한 기본적인 멀티태스킹 환

경을 지원하고 있다. 각 태스크는 자신의 문맥(Context)를 가지고 있으며 태스크 제어 블록(TCB, Task Control Block)에 저장된다. TCB는 다음과 같은 내용들이 포함된다.

1) 태스크의 프로그램 카운터
2) CPU 레지스터
3) 스택(stack) 및 함수 콜(function call)
4) 지연 타이머(delay timer)
5) 타임슬라이스 타이머(timeslice timer)
6) 커널 제어 구조 및 태스크의 우선순위
6) 시그널 핸들러

1.1.2 태스크 상태 천이도

각각의 태스크는 SUSPEND, READY, RUNNING, DELAYED, PENDING 5가지 상태 중 하나의 상태를 가지며, 특정 이벤트가 발생하면 다른 상태로 변화된다. 그리고 태스크가 특정 자원을 얻기 위해 기다릴 때 기다리는 최대 시간을 설정을 할 수 있으며 PENDING+DELAYED 상태가 된다. [그림 5]는 각 이벤트에 따른 태스크의 상태 천이도를 나타낸 것이다.

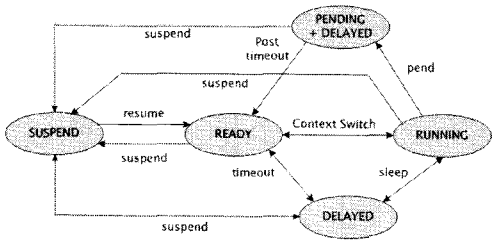


그림 5. 태스크 상태 천이도

1.1.3 태스크 스케줄링

본 논문에서 구현한 스케줄러는 0부터 7까지 8단계의 우선순위를 지원하며 동일한 우선순위에 대해서는 라운드 로빈을, 필요에 따라서 FIFO 스케줄링도 지원한다.

스케줄러는 각각의 우선순위에 따른 Ready List를 관리하고 있는데 스케줄러는 Ready List에서 가장 높은 우선순위를 찾아 실행한다. 그러나 태스크의 시간 결정성을 위해 Ready List에 삽입, 삭제, 검색은 비트맵 방식을 이용하여 정해진 몇 번의 수행으로 가능하다. [그림 6]은 8개의 우선순위를 갖는 태스크를 지원하기 위한 태스크의 자료 구조이다[6].

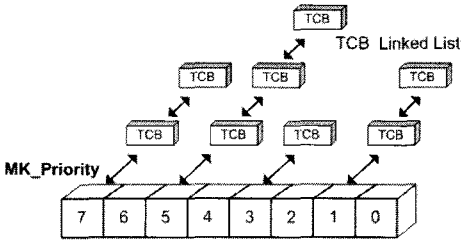


그림 6. 8개의 우선순위를 위한 커널 구조

1.2 메모리 관리(Memory Management)

동적 메모리 관리를 위해 [그림 7]과 같이 두 단계의 메모리 관리 기법을 제공한다. 하위 단계는 가변 크기의 메모리를 할당 및 해제할 수 있는 힙 스토리지 매니저(Heap Storage Manager)이고, 상위 단계는 고정 크기의 메모리를 할당 및 해제할 수 있는 메모리 풀(Memory Pool)이다.

힙 스토리지 매니저는 두 가지 문제를 안고 있다. 첫째, 힙(Heap)에서 메모리를 할당 할 경우 메모리 할당 시간의 예측이 불가능하여 시간 결정성을 저해하게 된다. 적당한 크기의 메모리 블록(Memory Block)을 찾기 위해서는 메모리의 프리 리스트(Free List)에서 찾아야 하는데, 이때 걸리는 시간은 O(n)으로 프리 블록의 수에 따라 결정되기 때문이다. 둘째, 힙(Heap)의 외부 단편화(External Fragmentation) 때문에 메모리 낭비의 문제가 있을 수 있다.

메모리 풀(Memory Pool)은 필요한 메모리의 크기를 미리 추정하여 동일한 크기의 메모리 버퍼를 할당하고 해제할 수 있도록 하는 것으로 힙에서 발생할 수 있는 외부 단편화 및 시간 결정성 저해 문제를 해결할 수 있다.

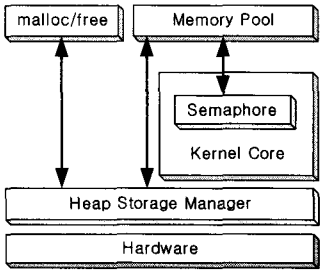


그림 7. 메모리 관리

1.3 세마포(Semaphore)

본 논문에서 구현한 세마포는 공유자원에 대한 상호 배타적인 접근 방법 및 태스크간 동기화라는 두 가지 요구 사항을 충족시키는 중요한 의미를 갖는다. 여러 개의 태스크가 동시에 공유 자원을 접근하면 데이터의 충돌이나 잘못된 연산을 수행할 수 있기 때문에 세마포를 두어 이를 얻은 태스크만이 공유 자원을 접근하고, 수행을 마친 뒤 세마포를 풀어주어 다른 태스크가 공유 자원을 획득할 수 있도록 한다. 세마포가 관리하는 공유자원의 수에 따라 세마포 카운트 값을 0 또는 양의 정수 값으로 초기화한다. [그림 8]은 세마포를 이용해 태스크간의 상호 배제의 예를 보여준다.

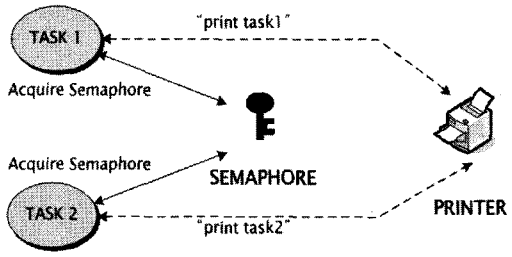


그림 8. 세마포를 이용한 태스크 사이의 상호 배제

1.4 ITC(inter-Task Communication)

실시간 운영체제는 여러 개의 독립적인 태스크가 동시에 실행될 수 있는 멀티태스킹 환경을 제공한다. 그러나 독립적인 태스크들 사이에 정보를 주고받기 위해서는 태스크 사이의 통신이 필요하다.

본 논문에서 구현한 실시간 운영체제에서는 메시지 큐, 메시지 포트, 메일박스, 태스크 포트 같은 다양한 태스크간 통신 매카니즘을 제공한다.

1.4.1 메시지 큐(Message Queue)

메시지 큐는 [그림 9]와 같이 태스크 또는 ISR이 여러 개의 메시지를 다른 태스크로 전달하려 할 때 사용한다. 메시지 큐를 통해 전달되는 메시지의 크기는 고정 길이 메시지 큐만 제공하며 메시지는 포인터의 전달이 아닌 메시지 복사에 의한 전달 방법을 사용한다.

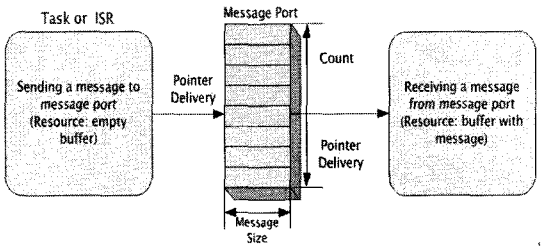


그림 9. 메시지 큐에서의 메시지 전달

1.4.2 메시지 포트(Message Port)

메시지 포트는 메시지 큐와 유사하지만 메시지 복사가 아닌 메시지의 포인터와 크기로서 전달이 된다.

1.4.3 메시지 메일박스(Message MailBoxes)

메시지 메일박스는 저장 가능한 메시지가 한 개인 메시지 큐이다. 메시지가 한 개이므로 여러 개의 메시지를 처리하는 메시지 큐에 비해 오버헤드가 적다.

1.4.4 태스크 포트(Task Port)

태스크 포트는 메시지 큐, 메시지 메일박스, 메시지 포트와 달리 태스크와 태스크간 1:1 동기적인 통신을 제공한다. 태스크 포트는 본 논문에서 구현한 태스크간 통신 도구들 중에서 메시지 전달을 위한 방법에서 가장 빠른 성능을 보여준다. 왜냐하면 태스크 포트는 별도의 자료구조를 사용하지 않고 TCB의 필드를 사용해 구현되었기 때문이다.

1.5 파워 관리(Power Management)

본 논문에서 구현한 DPM은 공급 전압을 조정하는 기법인 DVS(Dynamic Voltage Scaling : DVS)는 불가능하지만, 공급 주파수를 조정하는 기법인 DFS(Dynamic Frequency Scaling : DFS)는 가능한 하드웨어 플랫폼(MBA 2440 보드)에서 개발하였다[11]. 하지만 현재 센서 응용보드에 많이 사용되고 있는 atmega 계열의 8bit CPU에서는 이런 기능을 제공하지 않는다. 만약 센서 응용 개발 보드에서 DFS를 지원한다면 기존 UbiFOS-USN 의 커널수정 없이 저전력 구현이 가능하다.

표 1. DPM 구동 정책

Policy	Task+	Task	Task-	Idle
Battery Good	399	399	266	96
Battery Low	399	266	96	96
Battery Critical	399	96	96	96

DPM 기법을 적용한 실시간 운영체제는 배터리로 동작하는 하드웨어 플랫폼에서 제공하는 Frequency 레벨을 기준으로 구동정책을 구성하였으며, 본 논문에서는 총 3가지의 정책을 정의하여 구현하였다. 각 태스크는 실행 중에 DPM 스케줄링 API에 의해서 구동정책을 동적으로 변경할 수 있다.

[표 1]은 399, 266, 96MHz로 CPU frequency 변경이 가능한 MBA2440 보드에서 USN 보드를 가정하여 배터리의 상태에 따라 제안한 3단계의 DPM 구동정책으로, 애플리케이션에 따라 동적으로 CPU Frequency의 변경이 가능하도록 구성하였다. Task+, Task, Task-, Idle 은 각 DPM 정책 적용 시 동적으로 변경할 수 있는 CPU Frequency 모드로 기본은 Task 모드이다. 즉 각 DPM 정책에서 애플리케이션 수행 시 CPU Frequency를 높이려면 Task+ 모드, 낮추려면 Task- 모드로 Frequency를 동적으로 변경할 수 있다. 애플리케이션 수행 중 태스크가 유휴(Idle) 상태가 되면 Idle 모드로 자동 변경 된다.

표 2. DPM 관련 API

함 수	실 명
LP_Set_DPM_Policies()	구동정책 설정 함수
LP_MBA2440_Fscaler()	구동정책에 따른 레지스터 설정 함수
LP_Cal_Frequency()	CPU 주파수 계산 함수

[표 2]는 본 논문에서 구현한 DPM 기법의 저전력 관련 API로 LP_Set_DPM_Policies()함수는 구동정책을 인자로 CPU Frequency를 설정하는 함수이고, LP_MBA2440_Fscaler()는 구동정책에 따라 Frequency 값을 변경하기 위해 관련 레지스터의 값을 변경하는 함수이며, LP_Cal_Frequency()는 현재 태스크에 할당된 CPU Frequency를 계산하기 위한 함수이다.

또한 DPM 기능과 더불어 CPU의 상태를 SLEEP 모드(IDLE, ADC_NR, POWER_DOWN, POWER_SAVE)로 전환시켜 주는 APM(Advanced Power Management) 모듈도 구현하였다.

IV. 실험 결과

본 논문에서 구현한 실시간 운영체제가 최종적으로 탑재될 시스템은 (주)옥타컴의 Nano24 보드로서 다음과 같은 특징을 가진다.

- CPU Clock : 12MHz
- Flash : 128Kbyte
- RAM : 4Kbyte
- Radio : : 250kBoud
- Radio 형태 : Zigbee

[그림 10]은 센서 네트워크를 위한 실시간 운영체제를 개발하기 위한 테스트 환경이다.

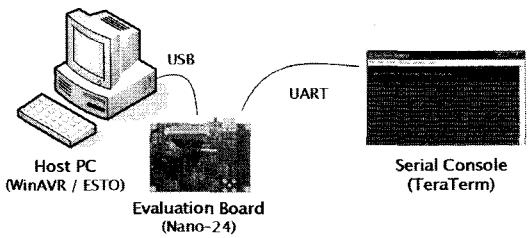


그림 10. 개발 환경

1.1 실시간 운영체제 커널 테스트

본 논문에서 구현한 실시간 운영체제는 센서 네트워크의 CPU와 메모리 같은 자원 제약적인 특성을 고려하고 실시간 운영체제가 갖추어야 할 시간결정성에 영향을 미칠 수 있는 스케줄링 시간과 인터럽트 응답시간을 고려하여 설계하였다. 메모리 자원의 제약을 극복하고 시간 결정성을 보장하기 위해 μC/OS와 같은 기존의 실시간 운영체제에서 맵 테이블(8byte)과 언맵 테이블(256byte)을 사용하여 시간결정성을 보장하는 방법 대신 심플 비트 맵 방식을 사용하여 맵 테이블과 언맵 테

이블을 사용하지 않으면서 시간결정성을 보장할 수 있는 스케줄링 방식을 제안하였다. 본 논문에서 제안한 스케줄링 방식은 8단계의 우선순위만을 제공하도록 제한하였는데 이는 센서 네트워크의 응용 프로그램에서 사용되는 태스크의 수가 적기 때문이며, 이를 통해 기존의 스케줄링 방식보다 적은 메모리를 사용하면서 성능면에서 보다 나은 스케줄링 방식을 구현할 수 있었다.

Nano24 보드를 사용하여 실험한 결과 기존의 스케줄링 방식을 사용한 경우스케줄링 시간이 평균 51.4μs가 소요되었고 제안한 스케줄링 기법을 사용할 경우 평균 45.7μs가 소요되었다. 인터럽트 응답시간은 평균 38μs가 소용됨을 확인하였다. 또한, 본 논문에서 구현한 실시간 운영체제가 안정적으로 커널 서비스를 수행하는지 확인하기 위하여 스케줄링과 태스크간 통신을 중점으로 테스트하였다. [그림 11]은 우선순위 선점형 스케줄링과 메시지 큐를 이용한 메시지 전송에 대한 테스트 결과이고 [그림 12]는 동일한 우선순위에 대해 FIFO와 라운드로빈(RR) 스케줄링을 테스트하고 세마포를 이용하여 태스크 간 상호배제를 테스트한 것이다. 또한 [그림 13]은 태스크 간 통신을 위해 사용되는 메시지 큐와 메시지 박스, 태스크 포트의 성능을 비교한 결과이다. 메시지 큐나 메시지 박스는 메시지의 복사를 통해 통신이 이루어지는데 비해 태스크 포트는 메시지의 포인터를 전달하는 방식으로 다른 방식에 비해 태스크 간 통신에 적은 시간이 걸린다는 것을 확인하였다.

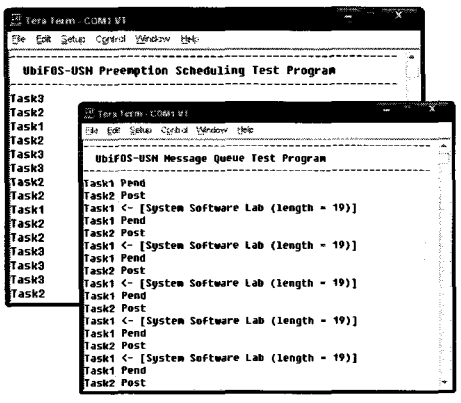


그림 11. 스케줄링 & 태스크간 통신 테스트

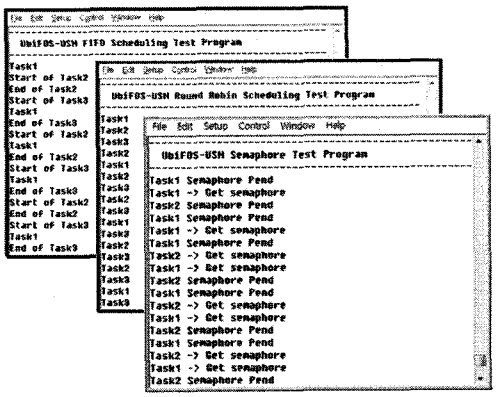


그림 12. 동일한 우선순위 스케줄링 & 세마포 테스트

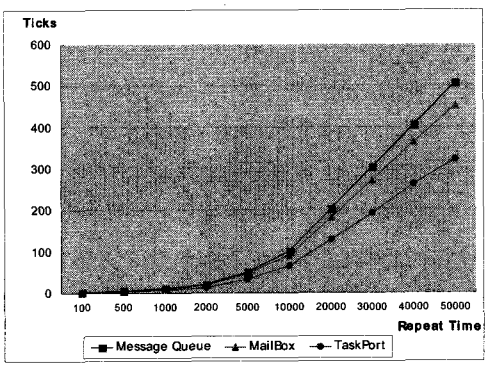


그림 13. 태스크 간 통신 도구의 성능 분석

1.2 저 전력 테스트

본 논문에서 구현한 DPM은 frequency를 제어할 수 있는 하드웨어 플랫폼이 필요하다. 하지만 현재 센서 노드용 하드웨어 플랫폼에는 이러한 기능을 지원하지 않는다. 따라서 frequency가 제어 가능한 MBA2440 (ARM920T) 보드[11]에서 DPM을 테스트 하였다. 응용 프로그램은 WAV 음악파일을 재생하는 것으로 DPM 정책에 따라서 전류의 변화를 측정하였다. [그림 14]는 Battery Good 정책을 적용할 경우로 최대 전류량이 0.35A, Battery Low 정책에서는 0.3A 그리고 Battery Critical 정책에서는 0.26A임을 보여주고 있다. 따라서 DPM 정책을 적용할 경우 소비 전류가 약 0.09A가 줄어들어, 전체 에너지 소비를 25% 이상 줄일 수 있음을 확인할 수 있다.

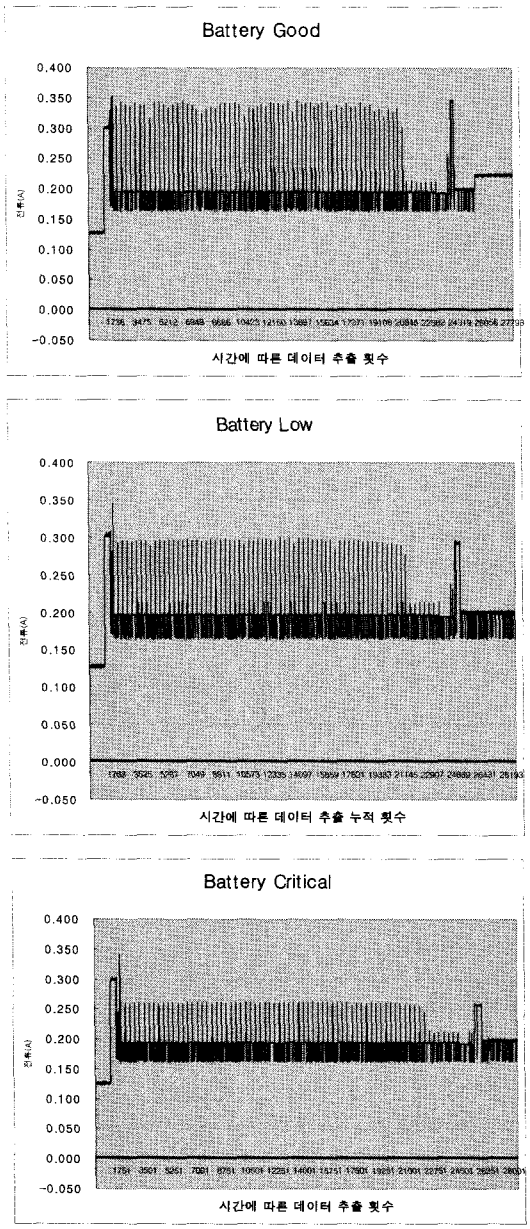


그림 14. DPM 정책에 따른 평균 전류 소모량

1.3 센서 네트워크 응용 테스트

본 논문에서 구현한 실시간 운영체제를 이용하여 [그림 16]에서는 IEEE 802.15.4 프로토콜을 이용하여 센서 노도와 싱크 노드간 RF 통신을 테스트 하였고 [그림 15]과 같은 환경에서 센서 노드간의 멀티 홉 라우팅 통신이 되는 것을 확인하였다. 라우팅 프로토콜은 (주)옥타컴에서 라이브러리 형태로 제공되는 것을 이용하였다 [10].

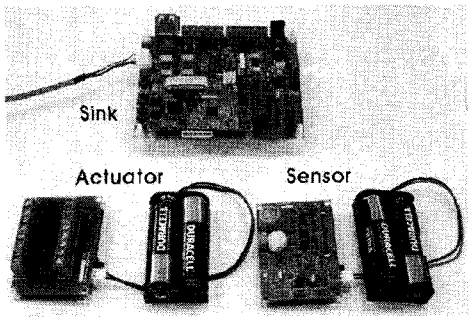


그림 15. 멀티 홉 라우팅 프로토콜 테스트 환경

[그림 16] 에서는 다음과 같은 Hex 값이 출력되는데 첫 번째 패킷을 분석해보면 Node ID가 1인 센서 노드에서 2개의 센서 값(조도, 온도)을 센서 노드에게 보내온 것을 의미한다. [그림 17]은 Zigbee 모듈을 통해 Sensor Device에서 수집된 정보가 Sink 노드로 전달되고 센서를 손으로 가리게 되면 Sink 노드가 Actuator를 제어하는 것을 확인할 수 있었다.

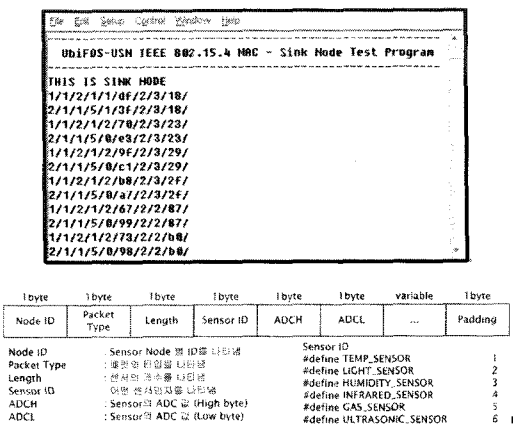


그림 16. IEEE 802.15.4 테스트

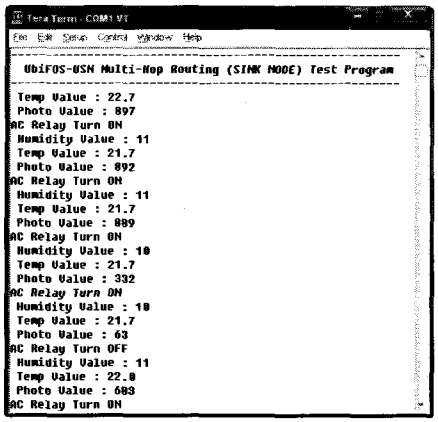


그림 17. 멀티 홉 라우팅 프로토콜 태스트 결과

1.4 커널 기능 비교

[표 3]은 관련연구에서 언급한 4가지의 센서 네트워크용 운영체제와 본 논문에서 구현한 실시간 운영체제의 비교 자료이다.

표 3. 센서 네트워크 운영체제 별 특징

특 징	TinyOS	SOS	MANTIS	NanoQplus	UbiFOS-USN
저전력 모드 지원	O	O	X	O	O
다중 모드 지원	X	O	O	O	O
동적 재프로그래밍	O	O	X	X	X
우선순위 스케줄링	X	X	O	O	O
실시간 보장	X	X	O	O	O
실시간 모델	컴포넌트 베이스	모듈 베이스	쓰레드 베이스	쓰레드 베이스	쓰레드 베이스

본 논문에서 제안한 실시간 운영체제는 기존에 센서 네트워크에 사용되었던 운영체제들의 특징인 적은 실행 이미지 크기와 저 전력 기능을 포함하고 있다. 본 논문에서는 센서 네트워크 응용에서 적은 태스크를 사용하여 응용프로그램을 작성한다는 것에 착안하여 8단계의 우선순위만을 제공하지만 보다 나은 실시간성을 제공하는 스케줄링 기법을 제안하였으며, 기존 실시간 운영체제들에서 사용되었던 APM 저전력 기능을 보완할 수 있는 DPM 저전력 기능을 추가하여 보다 적은 전력 소모를 갖는 시스템을 구성할 수 있음을 보였다. 또한, 기존의 실시간 운영체제들이 메시지 전달 방식으로 대부분 전역 변수나 메시지 큐를 통해 태스크 간 통신을 수행하였지만, 본 논문에서는 센서 네트워크 응용 프로그램의 태스크 간 통신에 태스크 포트를 사용함으로써 보다 나은 성능을 제공할 수 있음을 보였다.

V. 결론 및 향후 연구 과제

본 논문에서는 실시간성을 보장하는 센서 네트워크 운영체제를 구현하였다. 이 운영체제를 사용하면 노드에서 동시에 일어나는 작업들의 우선순위에 따라 실시간 스케줄링이 가능하며, 분산된 환경에서 다른 노드로부터 받은 메시지를 유효한 시간 내에 처리할 수 있게 해줌으로써 시스템의 실시간성을 만족 시킬 수 있다.

본 논문에서 구현한 실시간 운영체제는 4Kbytes 대의 실행 이미지를 갖는 우선순위 기반의 선점형 커널로서 센서 네트워크용 운영체제에 요구되는 기능을 고려하여 기본적으로 멀티태스킹을 위한 커널 자료구조와 태스크 관리 및 스케줄링, 태스크간 통신 및 동기화, 동적 메모리 관리 등의 서비스를 제공한다. 또한 센서 네트워크를 구성하기 위한 센싱 및 구동 기능, 통신 프로토콜인 지그비 RF 모듈을 지원하고 마지막으로 DPM, APM을 이용하여 배터리의 파워를 절약할 수 있는 기법을 구현하였다. 본 논문에서 구현한 센서 네트워크용 실시간 운영체제는 해당 특정 플랫폼에서 최적의 기능을 수행 할 수 있도록 모듈 별로 구현함으로써, 작은 이미지 사이즈를 요구하는 응용 프로그램에 만족시킬 수 있다.

향후 연구 과제로는 센서 네트워크용 응용프로그램에서 일반적으로 발견되는 태스크 셋을 효율적으로 처리하기 위한 최적화 작업이 필요하다. 마지막으로 한번 설치가 되면 커널에 대한 수정이 힘든 만큼 센서 네트워크상에서 동적으로 커널을 재구성할 수 있는 연구가 필요하다.

참 고 문 헌

[1] Chalermak Intanagonwiwat, Ramesh govindan, and Deborah Estrin, A Scalable and Robust

Communication Paradigm for sensor networks. MOBICOM 2002.

[2] C. Chong and S. P. Kumar, "Sensor Networks: Evolution, Opportunities, and Challenges," Proc. of the IEEE. Vol.91, No.8, Aug. 2003.

[3] C. C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A Dynamic Operating System for Sensor Networks," Proc. of the 3rd Int'l Conf. on Mobile Systems, Application, and Services (MobiSys '05), Seattle, Washington, 2005.

[4] M. T. Schmitz, B. M. Alhashimi, and P. Eles, *System-Level Design Techniques for Energy-Efficient Embedded Systems*, Kluwer Academic Pub., Boston, 2004.

[5] H. Blanchard, B. Brock, M. Locke, M. Orvek, R. Paulsen, and K. Rajamani, *Dynamic Power Management for Embedded Systems*, IBM and MontaVista Software, Version1.1, Nov. 2002.

[6] M. H. Cho, J. W. Lee, H. S. Kang, and C. H. Lee, "Design and Implementation of Light-Weight Real-Time Operating System for Audio Player," KISS Autumn Conf., Vol.2, pp.328-330, 2006.

[7] http://www.tinyos.net/

[8] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, and R. Han, "MANTIS: System Support for MultimodAl NeTworks of In-situ Sensors," 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), pp.50-59, 2003.

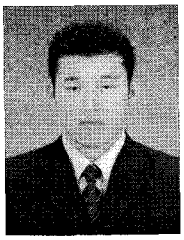
[9] C. C. Han, R. K. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava, "SOS: A dynamic operating system for sensor networks," Proc. of the 3rd Int'l Conf. on Mobile Systems, Applications, And Services(Mobisys), 2005.

[10] http://qplus.or.kr, http://octacomm.net/

[11] http://mculand.com/

저 자 소 개

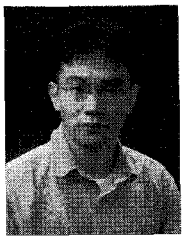
강 희 성(Hui-Sung Kang) 준회원



- 2005년 2월 : 충남대학교 컴퓨터공학과 (공학사)
- 2005년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 (석사과정) 재학 중

<관심분야> : 실시간 운영체제, 센서 네트워크

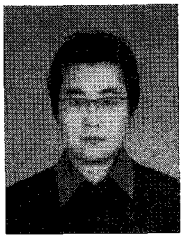
전 상 호(Shang-Ho Jeon) 준회원



- 2006년 2월 : 충남대학교 컴퓨터공학과 (공학사)
- 2006년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 (석사과정) 재학 중

<관심분야> : 실시간 운영체제, 자바가상머신

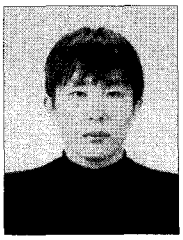
정 근 재(Geun-Jae Jeong) 준회원



- 2006년 2월 : 충남대학교 컴퓨터공학과 (공학사)
- 2006년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 (석사과정) 재학 중

<관심분야> : 실시간 운영체제, 임베디드 시스템

이 숭 열(Soong-Yeol Lee) 준회원



- 2004년 2월 : 충남대학교 컴퓨터공학과 (공학사)
- 2005년 9월 ~ 현재 : 충남대학교 컴퓨터공학과 (석사과정) 재학 중

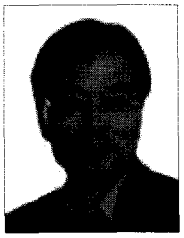
<관심분야> : 실시간 운영체제, 임베디드 시스템

김 용 희(Yong-Hee Kim) 정회원

- 2003년 2월 : 충남대학교 컴퓨터공학과 (공학석사)
- 2003년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 (박사과정) 재학 중

<관심분야> : 실시간 시스템, 실시간 운영체제, 고장허용 컴퓨팅

이 철 훈(Cheol-Hoon Lee) 정회원



- 1983년 2월 : 서울대학교 전자공학과 (공학사)
- 1988년 2월 : 한국과학기술대학교 컴퓨터공학과 (공학석사)
- 1992년 2월 : 한국과학기술대학교 컴퓨터공학과 (공학박사)
- 1983년 3월 ~ 1986년 2월 : 삼성전자 컴퓨터사업부 연구원
- 1992년 3월 ~ 1994년 2월 : 삼성전자 컴퓨터사업부 선임연구원
- 1994년 2월 ~ 1995년 2월 : Univ. of Michigan 객원연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙연구원

<관심분야> : 실시간시스템, 운영체제, 고장허용 컴퓨팅