μTMO 모델 기반 실시간 센서 네트워크 운영체제

(μTMO Model based Real-Time Operating System for Sensor Network)

이 재 안 † 허 신 †† 최 병 규 †
(Jae An Yi) (Shin Heu) (Byoung Kyu Choi)

요 약 센서 네트워크의 응용 범위가 점차 넓어지면서 군사 분야나 방사능 감지와 같이 실시간성을 요구하는 응용분야가 생겨나게 되었다. 하지만 기존의 센서 운영체제 연구는 효율적인 자원 활용에 초점을 두고 연구가 진행되었기 때문에 실시간성을 만족시켜 주기 어려운 구조를 가지고 있다. 본 논문에서는 정시성을 보장하는 실시간 분산 객체 TMO 모델을 센서 네트워크의 제한된 자원 환경에 알맞도록 경량화 시킨 μTMO 모델을 제안한다. μTMO 모델을 이용한 실시간 센서 네트워크 운영체제를 개발하기 위하여 한국전자통신연구원에서 개발한 센서 노드용 운영체제인 Nano-Q+를 이용하였다. Nano-Q+의 타이머 모듈을 높은 해상도를 가질 수 있도록 수정하였으며, EDF(Earliest-Deadline-First)기반의 실시간 스케줄러에 CST(Context Switch Threshold) 와 PAS(Power Aware Scheduling) 기법을 적용하여 센서 노드에 적합한 실시간 스케줄러로 변경하였다. μTMO 모델을 지원하기 위해 채널 기반의 통신 수단인 ITC-Channel을 새롭게 구현하였으며, 주기적인 스레드를 관리하는 WTMT(Watchdog TMO Management Task) 모듈을 구현하여 SpM 스레드를 주기에 맞게 실행할 수 있도록 하였다.

키워드 : 센서 네트워크, 실시간 운영체제, 센서 네트워크 운영체제

Abstract As the range of sensor network's applicability is getting wider, it creates new application areas which is required real-time operation, such as military and detection of radioactivity. However, existing researches are focused on effective management for resources, existing sensor network operating system cannot support to real-time areas.

In this paper, we propose the μTMO model which is lightweight real-time distributed object model TMO. We design the real-time sensor network operation system μTMO-NanoQ+ which is based on ETRI's sensor network operation system Nano-Q+. We modify the Nano-Q+'s timer module to support high resolution and apply Context Switch Threshold, Power Aware scheduling techniques to realize lightweight scheduler which is based on EDF. We also implement channel based communication way ITC-Channel and periodic thread management module WTMT

Key words : Sensor network, Real-time Operating Systems, Sensor Network Operation Systems

† 학생회원 : 한양대학교 컴퓨터공학과
bkchoi@cse.hanyang.ac.kr
mysaint@naver.com
†† 종신회원 : 한양대학교 컴퓨터공학과 교수
1880324@hanyang.ac.kr
논문접수 : 2007년 3월 12일
심사완료 : 2007년 11월 13일

정보과학회논문지 : 시스템 및 이론 제34권 제12호(2007.12)

1. 서 론

무선 센서 네트워크는 산재되어 있는 무선 센서 노드가 주변 환경으로부터 데이타를 수집하고 이를 가공하며, 노드들 간의 협업을 통하여 만들어진 무선 네트워크를 통하여 데이타를 전송함으로써 사용자가 이를 활용할 수 있도록 해주는 기술이다. 센서 네트워크를 구성하는 센서 노드는 극도로 제한된 자원을 가지고 있기 때문에 효율적인 자원의 활용이 매우 중요한 이슈가 된다[1].

일반적인 센서네트워크 응용 환경에서는 제한된 자원을 효율적으로 사용하기 위해 효율적으로 동작하는 운영체제가 필요하다. 이러한 범용 센서 네트워크용 운영

체제는 운영체제 자체의 크기가 매우 작아야 하며, 메모리 관리가 효율적이어야 한다. TinyOS[2]나 MANTIS [3]와 같은 범용 센서네트워크 운영체제는 자원의 효율적인 사용에 초점을 두고 연구 되었다.

센서 네트워크의 활용 분야가 넓어지면서 군사 분야에서 적을 탐지하거나 핵발전소 또는 핵 위험지역의 방사능 측정과 같은 실시간을 요구하는 분야가 나타나게 되었다. 방사능과 같은 유독성 물질을 감시하는 센서 네트워크 응용의 경우, 센서 노드가 수집한 데이타를 제한된 시간 이내에 전달하지 못할 경우 지연된 데이타의 의미가 없어질 수 있다[4]. 실시간 무선 센서네트워크 연구 분야에서 실시간 요구를 만족시키기 위해 SPEED [5]나 RAP[6]와 같은 무선 센서 네트워크용 실시간 통신 프로토콜들이 연구 되고 있다. 하지만 센서 네트워크용 실시간 운영체제에 대한 연구는 아직까지 활발하지 않다.

일반적으로 실시간 운영체제는 범용 운영체제에 비하여 스케줄링 단계에서 계산이 추가되며, 실시간 지원을 위한 부가적 기능에 의한 오버헤드가 발생된다. 센서 노드는 제한적인 자원 환경에서 동작하여야 하기 때문에 센서 노드용 실시간 운영체제를 개발하기 위해서는 실시간 지원에 따른 오버헤드를 최소화 하여야 한다.

본 논문에서는 정시성을 보장하는 실시간 분산 객체 TMO 모델을 센서네트워크의 제한된 자원 환경에 알맞도록 경량화 시킨 μTMO(MicroTMO) 모델을 제안하고, 범용 센서 노드용 운영체제에 μTMO 모델을 적용한 실시간 센서 네트워크 운영체제를 설계 및 구현하였다. 2장에서는 관련 분야로 실시간 센서 네트워크 응용 분야를 설명하고 분산 실시간 객체인 TMO에 대해 설명한다. 3장에서는 TMO를 적용 시 발생 가능한 문제점을 살펴보고 이를 해결하기 위해 TMO 모델을 경량화 시킨 μTMO 모델을 제안한다. 4장에서는 μTMO 모델을 적용한 실시간 센서 네트워크 운영체제인 μTMO-NanoQ+의 설계 및 구현을 기술한다. 5장에서는 실험을 통하여 새롭게 구현된 μTMO-NanoQ+를 검증한다. 마지막 6장에서는 결론을 내리고 향후 연구 방향에 대해 논의한다.

2. 관련연구

2.1 Nano-Q+ 운영체제

Nano-Q+는 한국전자통신연구원(ETRI)에서 개발한 센서 노드용 운영체제로써 다음과 같은 특징을 가진다.

- 에너지 소모를 최소화하기 위해 네트워크를 구성하는 노드 사이에 동기화된 시간을 기반으로 주기적으로 휴면과 활성화 모드를 반복하면서 메시지를 효율적으로 처리
- 제한된 메모리의 사용을 최소화하기 위해 스레드 사이의 스택 공유
- 멀티 스레드 스케줄러

그림 1은 Nano-Q+의 구조를 보여준다. Nano-Q+는 기존의 유닉스 시스템과 유사한 계층적 구조를 가진다. 기존의 클래식한 운영체제 형태를 유지하면서 불필요한 모듈을 제거하고, 운영체제의 각 기능들을 경량화 하였다. 전체적인 운영체제의 형태는 기존의 운영체제와 유사하지만 센서 네트워크에 최적화하기 위하여 모듈화를 통한 재구성이 가능하다[7-9].

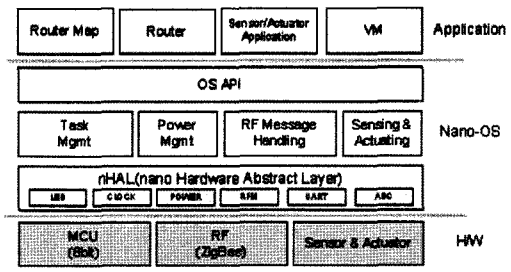

그림 1 Nano-Q+ 운영체제 구조[9]

2.2 실시간 센서 네트워크 응용

센서 네트워크의 응용분야가 점차 넓어지면서 센서 네트워크에서 실시간 지원의 필요성이 대두되고 있다. 기존의 응용 분야에서는 자연환경 감시, 동물의 이동감시와 같이 데이타 수집에 실시간성이 필요하지 않은 분야가 주를 이루었다. 하지만 방사능 탐지나 자연재해 감지, 군사 감시와 같은 응용분야에서는 수집된 데이타를 전달하기까지 오랜 시간이 소모될 경우, 데이타 자체의 의미가 없어진다. 이러한 몇몇 특수한 센서 네트워크 응용 분야에서는 실시간 지원이 필수적이다.

군사 분야에서 적을 탐지하여 베이스 스테이션으로 정보를 전달해 주는 응용분야를 살펴보자. 전장에서 활용하는 센서 네트워크는 전쟁 기간에만 사용되며, 전장의 위치가 계속적으로 변하기 때문에 단기간 사용될 가능성이 높다. 이런 응용 분야에서는 감지된 데이타를 유효한 시간 이내에 전달 해주어야 한다. 적이 지나가고 난 다음에 베이스 스테이션으로 데이타가 전달되면 데이타의 가치가 작아지거나 데이타 자체의 의미가 없어질 수 있다.

그림 2는 전장에서의 실시간 센서 네트워크 활용을 보여준다. 전장에 무작위로 배치된 무선 센서 노드들이 서로 협업을 통하여 네트워크를 구축하고 주변 환경을 감시한다. 탱크와 같은 적이 기습을 하는 경우, 수집한 정보를 바로 Sink 노드(베이스 스테이션)로 알려줌으로써 기존의 군사 통신망과 연결된 싱크가 적의 이동경로

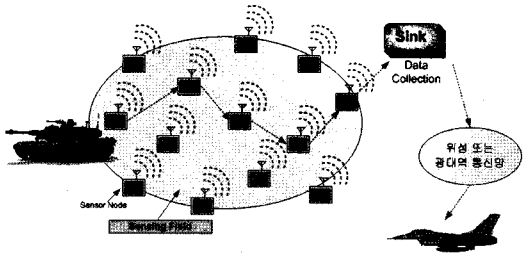

그림 2 전장(Battle Field)에서 센서네트워크의 활용

를 아군에게 알려줄 수 있다.

이러한 특수한 분야에서는 실시간성이 필요하지만 운영체제에서 실시간성을 지원해 주기 위해서는 실시간 스케줄러에서 부가적인 계산이 추가되기 때문에 운영체제 의 크기가 커지며 효율성이 떨어지게 된다.

자원이 한정적인 센서 네트워크용 운영체제에서 실시간성을 만족시켜 주기 위해서는 데드라인 기반의 실시간 스케줄러 구현과 실시간 지원을 위한 부가적인 기능 추가에 따른 오버헤드의 최소화가 필요하다.

2.3 TMO 모델

TMO 모델은 Time-triggered Message-triggered Object의 약자로써 U.C. Irvine의 Kane Kim 등에 의하여 개발된 정시보장 컴퓨팅 패러다임(Timeliness guaranteed computing paradigm)을 지향하는 분산 객체 모델이다. 여러 개의 TMO 네트워크로 설계된 시스템은 분산 환경에서 시간 조건에 의하여 수행된다[10].

그림 3은 TMO 모델의 구성 형태를 보여준다. TMO는 다음과 같이 크게 세부분으로 구성된다.

- ODS는 객체 자료 저장소(Object Data Store)로써 SpM과 SvM 메소드들이 공유하는 데이타를 가지고 있으며, 최대 유효기간(Maximum Validity Duration)을 초과하게 되면 그 데이타는 유효성을 잃는다.
- SpM(Spontaneous Method)은 주어진 시간 조건에 의해 자율적으로 구동된다. SpM에는 시작과 종료시간, 주기, 데드라인으로 구성된 시간 제약이 주어지며 커널이나 TMO 엔진에서 이러한 시간 조건에 따라 실시간 스케줄링을 제공한다.
- SvM(Service Method)은 IPC 이벤트 메시지 수신에 의해 구동이 되며 실행이 시작되면 주어진 데드라인에 의해 스케줄링 된다. SvM을 구동시키는 IPC 메시지는 분산 환경에서도 변경 없이 그대로 적용 가능한 네트워크에 투명한 IPC이다.

3. μTMO 모델의 설계

3.1 TMO 모델의 센서 네트워크 적용

분산 실시간 객체 모델인 TMO를 실시간 센서 네트

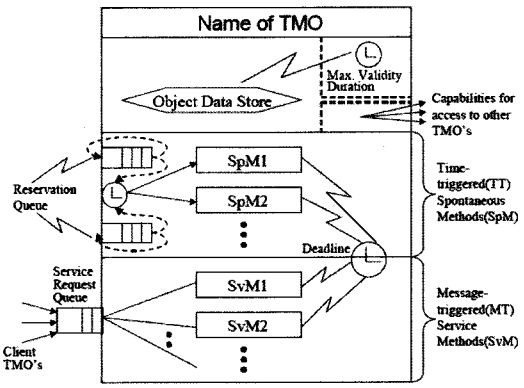

그림 3 TMO 모델[11]

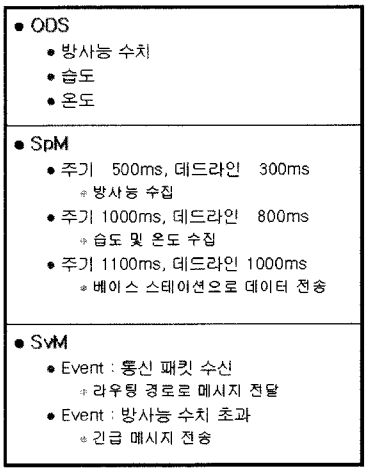

그림 4 TMO 모델을 적용한 실시간 센서 응용

워크 응용에 적용 시켜보면 그림 4와 같은 형태로 표현이 가능하다.

그림 4는 방사능 유출을 감지하는 센서 네트워크 응용을 TMO 모델을 이용하여 표현한 것이다. 방사능 정보를 수집하여 전달하는 것은 반드시 정해진 시간 이내에 처리가 완료 되어야 하는 실시간성이 요구되는 작업이다.

예제에서 주기적인 작업을 살펴보면 500ms 마다 방사능 물질 정보를 수집하고 300ms의 데드라인을 가진다. 부수적인 정보인 온도, 습도는 방사능에 비하여 상대적으로 덜 중요한 정보이므로 1000ms 마다 정보를 수집하고 800ms의 데드라인을 가진다. 매 1100ms 마다 수집된 데이타를 베이스 스테이션으로 전송한다. 이와 같은 주기적인 작업은 SpM을 이용하여 간단히 표현 가능하다.

다른 노드로부터 메시지를 수신하거나 방사능 수치가 급격히 변하는 경우에는 이벤트가 발생된다. 이러한 이

벤트에 의해 수행되는 동작은 SvM 스레드를 통해 표현 가능하다.

TMO 모델에서는 SpM 메소드를 이용하여 주기적인 작업을 처리하고, SvM 메소드를 이용하여 비주기적인 이벤트성 작업을 처리한다.

센서 네트워크 응용의 경우 주기적으로 데이타를 수집하고 이벤트가 발생하는 경우 지정된 작업을 수행함으로써 센서 노드의 동작 특성이 결정된다. TMO 모델을 이용하는 경우 이러한 동작 특성을 설계 단계에서 높은 추상화 수준으로 표현 할 수 있다.

기존의 멀티 스레드 기반 센서 네트워크 운영체제에서는 주기적인 작업 표현을 위해 실행 코드 중간에 sleep()과 같은 시간 대기 함수를 이용하였다. sleep()과 같은 시간 대기 함수가 복잡한 순환문이나 분기문에 적용되는 경우 코드의 동작을 이해하는데 어려움이 있었으며, 타이밍 문제가 발생할 경우 디버깅이 어려운 단점이 있다. 또한 멀티 스레드 환경에서는 타이머 API를 쓴다 하더라도 정확한 주기를 맞추기 힘들다.

3.2 TMO 모델 적용 시 발생 가능한 문제점

3.1절에서 서술하였듯이 TMO 모델을 센서 응용에 적용하는 경우 높은 추상화 수준에서 시간 제약에 대한 표현이 가능하며, 실시간 지원이 가능하다. 이러한 장점이 있는 TMO 모델을 센서 네트워크에 그대로 적용하는 경우 다음과 같은 문제점이 발생할 수 있다.

객체지향 프로그래밍 패러다임은 캡슐화, 상속성, 다형성의 특징을 가진다. 객체지향 방식을 사용하는 경우 복잡한 프로그램을 높은 추상화 수준에서 표현할 수 있는 장점이 있다.

객체를 사용하는 경우 클래스의 특성 표현을 위한 약간의 메모리 오버헤드가 있다. 캡슐화와 상속성은 컴파일 단계에서 처리되는 부분이므로 실행 중 오버헤드는 그렇게 크지 않다. 다형성을 지원하기 위해서는 가상 함수의 위치를 저장하는 가상 함수 테이블을 유지하여야 하며, 명령어의 개수가 적은 RISC 머신에서는 가상 함수를 호출하기 위한 어셈블리 명령어가 증가한다. 일반적으로 순차적 프로그래밍에 비하여 크기가 커지며 실행 시 동적으로 결정되는 동적 바인딩에 의하여 실행이 느려질 수 있다. 일반적인 컴퓨팅 환경에서는 크게 문제가 되지 않는 부분이지만, 센서 노드의 경우 8Mhz 정도의 동작 속도를 가지는 AVR과 같은 MCU를 주로 사용하므로 객체지향 방식으로 프로그래밍 하는 경우 속도 저하가 일어 날 수 있다. 센서 노드 내부에서는 일반적으로 4KB 정도의 메모리를 제공하기 때문에 메모리 사용량이 많은 객체지향 방식 프로그래밍은 센서 노드에서 사용하기에는 부담이 될 수 있다. 또한 센서 노드의 특성상 동작이 단순하기 때문에 객체지향 패러다임을 적용하여 얻을 수 있는 장점이 크지 않다.

TMO 모델은 분산 객체 개념을 기반으로 동작하기 때문에 여러 TMO 객체 사이에 IPC나 로컬 네트워크를 통한 통신이 빈번하게 일어난다. 유선 환경에서는 로컬 네트워크에 브로드캐스팅이나 멀티캐스팅을 이용한 분산 환경 구축이 큰 문제를 일으키지 않지만, 무선 네트워크를 사용하는 센서 네트워크에서는 과도한 에너지 소모 및 송수신 방해와 같은 문제가 발생할 가능성이 있다. TMO 모델 사이에 전달되는 메시지가 전체 네트워크로 전송되면 무선 대역폭을 차지하여 다른 정보의 송수신을 방해하는 것 뿐 아니라, 잦은 무선 송수신에 의해 센서 노드의 중요한 자원인 전력을 많이 소모한다. 따라서 무선 통신에 따른 전력 소모가 많은 분산 모델을 센서 네트워크에 그대로 적용하기에는 무리가 따른다.

3.3 μTMO 모델의 설계

3.2 절에서 제시한 문제점을 해결하기 위해 TMO 모델을 센서 네트워크에 알맞도록 경량화한 μTMO (micro-TMO) 모델을 제안한다.

제한적인 자원을 가지는 MCU를 사용하기 때문에 객체지향 모델을 절차적 프로그래밍 언어인 C언어를 통하여 TMO 모델을 지원할 수 있도록 변경하였다. SpM과 SvM 함수의 경우 TMO 객체의 멤버 함수이었던 것에 반하여 μTMO 모델에서는 각 함수를 함수 포인터를 이용하여 맵핑하는 구조를 취함으로써 C언어를 이용하여 TMO 모델을 표현할 수 있도록 하였다. 이를 위해 시스템 API에서 커널에 SpM 스레드와 SvM 스레드를 등록할 수 있는 시스템 콜을 제공한다. 이러한 등록 함수는 운영체제가 시작될 때 한번만 수행되면 된다.

기존 TMO 모델에서는 IPC를 통해 분산 환경에서 오는 클라이언트의 이벤트 메시지를 처리한다. 센서 네트워크에서 이러한 분산 처리 방식을 그대로 사용할 경우 무선 통신을 통하여 브로드캐스트 또는 멀티캐스트로 메시지를 보내야 하는데, 이는 통신 대역폭이 작은 센서 네트워크의 특성상 너무 많은 대역폭을 사용하게 되며, 빈번한 무선 전송으로 인해 에너지 소모가 많아진다.

센서 노드의 협업 동작 특성을 TMO 모델과 비교하여 보면, TMO 모델은 다수의 분산된 TMO 객체들이 IPC를 통하여 협업한다. 센서 응용의 경우 그림 4에서 볼 수 있듯이 하나의 TMO 모델만으로도 센서 노드의 동작 표현이 가능하다. 또한 노드의 자원 제약 때문에 다수의 TMO 모델을 단일 노드에서 실행하는 것은 큰 부담이 된다. 센서 네트워크에서 다른 노드와의 협업은 주로 경로 계산을 위한 라우팅 정보 전달이나 데이타 전달과 같은 간단한 형태로 일어난다. TMO 모델처럼 IPC나 로컬 네트워크를 통해 계속 메시지를 전달하기 보다는 협업이 필요한 경우 약속된 패킷을 통해 협업하

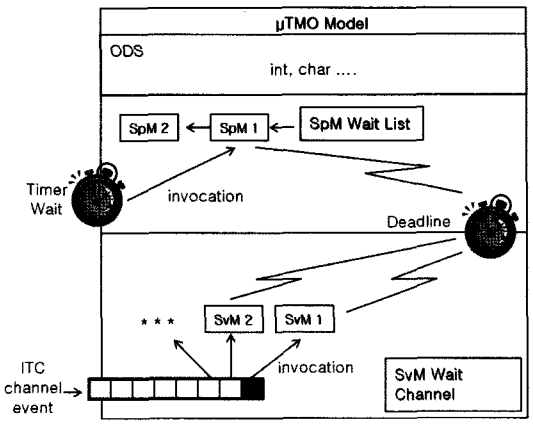

그림 5 µTMO 모델 구조

는 일반적인 센서 네트워크의 협업 방식으로 처리가 가능하다.

그림 5는 센서 네트워크에 알맞도록 변형된 µTMO 모델의 구조를 보여준다. TMO 객체의 멤버 함수로 구성되어있던 SpM을 맵핑할 함수의 포인터와 주기, 데드라인으로 구성된 리스트로 관리 하고, SvM을 함수 포인터와 이벤트 종류, 데드라인을 가지는 리스트로 관리한다.

지정된 주기에 따라 동작하는 SpM 리스트의 관리는 TMO모델과 마찬가지로 WTMT(Watchdog TMO Management Task)에 의해 관리가 되며, 지정된 주기가 되면 WTMT에 의하여 깨어나서 운영체제의 실시간 스케줄러에 의하여 관리를 받게 된다.

µTMO 모델을 센서 응용프로그램에 적용하면 그림 4에서 볼 수 있듯이 디자인 단계에서 실시간성에 대한 제약을 명확하게 줄 수 있으며, 간단한 동작을 하는 함수를 시간 조건이나 특정 이벤트 메시지들과 맵핑시키는 방식을 사용하므로 실시간 프로그래밍에 대한 특별한 학습 없이 기존 멀티스레드 기반의 프로그래밍 기법을 통하여 쉽게 실시간성 지원이 가능하다.

4. µTMO-NanoQ+ 운영체제의 설계 및 구현

4.1 µTMO모델 적용 실시간 운영체제의 설계

본 논문에서 제안하는 µTMO 모델을 적용한 실시간 센서네트워크 운영체제의 구조는 그림 6과 같다.

최하위 계층에 MCU나 무선 통신을 위한 모듈, 센서 모듈과 같은 하드웨어 계층이 존재하며, 이 하드웨어를 편리하게 사용하기 위한 디바이스 드라이버의 역할을 하는 하드웨어 추상화 계층(HAL)이 존재한다.

SvM 스레드를 위한 채널 기반의 통신 수단인 ITC (Inter Thread Communication)가 존재한다.

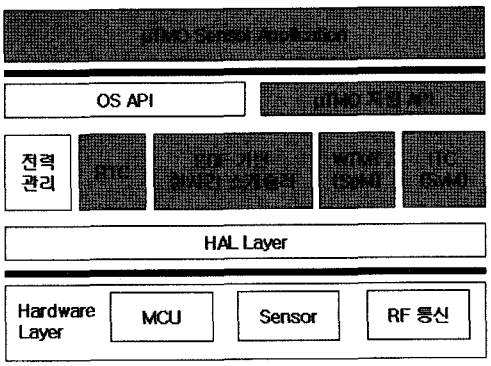

그림 6 µTMO 실시간 운영체제 구조

실시간 스케줄러는 계산 능력이 떨어지는 MCU의 효율적인 사용을 위하여 CPU 활용률이 높은 EDF[12] 스케줄러를 센서 네트워크 알맞도록 변형하여 사용한다.

고해상도 클럭인 RTC에 의하여 깨어나는 WTMT가 SpM의 주기적인 동작을 관리하게 된다. 이외의 구조는 멀티 스레드 기반의 센서 네트워크 운영체제인 MANTIS[3]나 Nano-Q+의 구조와 유사하다.

4.2 주기적 실시간 스레드(SpM)

주기적 실시간 스레드(SpM)은 타이머에 지정된 주기에 따라 활성화되어 지정된 데드라인에 맞추어 수행을 마치는 스레드를 말한다.

그림 7은 SpM 스레드의 상태 전이를 보여준다. 일반 프로세스와의 차이점은 타이머 인터럽트에 의하여 invocation되며, 수행을 완료하고 다음번 invocation을 기다릴 때 다시 SpM Invocation Wait 상태로 돌아간다.

SpM 스레드의 주기적인 특성은 WTMT(Watchdog TMO Management Task)에 의하여 관리된다. 그림 8은 WTMT에 의하여 SpM Wait List에 대기중인 SpM이 지정된 주기에 invocation되어 실시간 스케줄러로 전달되는 것을 보여준다.

WTMT는 MCU 내부 타이머인 Timer_3B에 SpM으

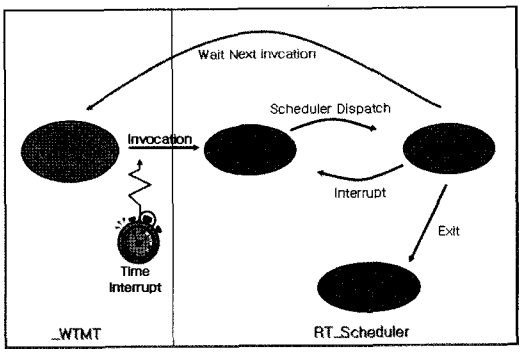

그림 7 SpM 스레드의 상태 변이

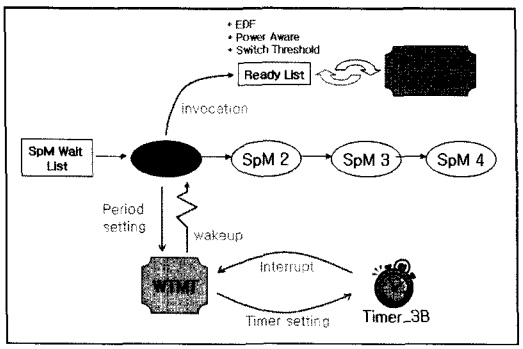

그림 8 WTMT에 의한 SpM의 주기 관리

로부터 가져온 주기 정보를 설정하고, Timer_3B 인터럽트가 발생하면 대기 중이던 SpM을 invocation 상태로 변경하여 Ready List에 입력하고 실시간 스케줄러가 관리 하도록 한다.

WTMT 모듈은 타이머 오동작에 의한 시간오차를 보정해 주기 위한 시간 보정장치를 가지고 있다.

4.3 실시간 서비스 스레드(SvM)

실시간 서비스 스레드(SvM)은 비동기적인 이벤트를 받으면 invocation 되어 데드라인에 따라 실시간 스케줄러에 의해 관리된다. 그림 9는 SvM 스레드의 상태 전이 다이어그램이다.

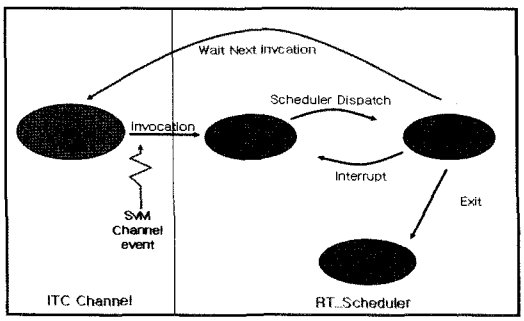

그림 9 SvM 스레드의 상태 전이도

그림 9에서 볼 수 있듯이 SpM 스레드와 유사한 상태 전이도를 가진다. SpM과 다른 점은 SpM의 경우에는 타이머 인터럽트에 의하여 invocation 되었지만 SvM 스레드는 ITC-Channel 이벤트에 의하여 invocation 된다.

4.4 ITC-Channel 시스템

SvM 메소드에 전달되는 비동기 이벤트를 처리하기 위하여 Nano-Q+에는 존재하지 않는 채널 기반 스레드 간 통신 방법을 새롭게 추가 하였다. 그림 10은 채널 기반의 통신 수단인 ITC-Channel의 구조와 이벤트에 의해 SvM 스레드가 invocation 되는 것을 보여준다.

이벤트는 내부에서 발생하는 지역이벤트와 통신 패킷

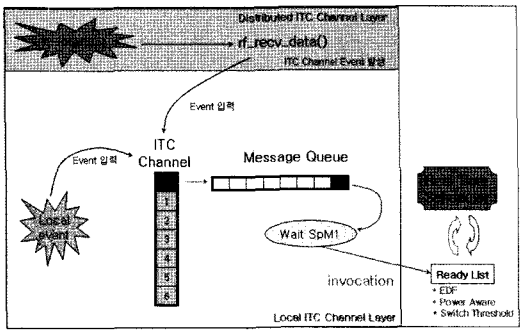

그림 10 ITC-Channel이벤트에 의한 SvM의 Invocation

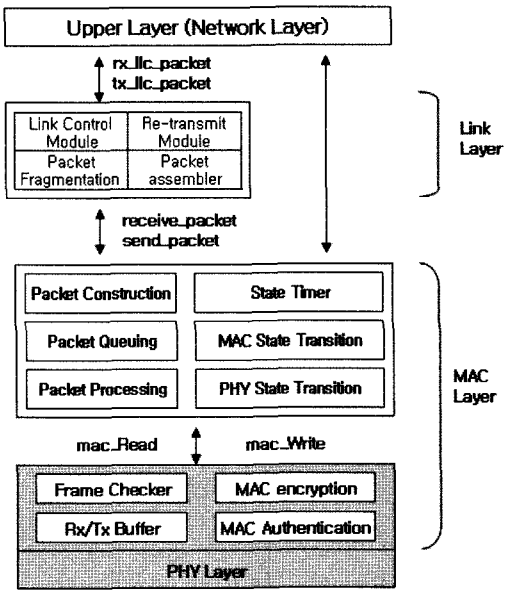

그림 11 NanoQ+의 RF 메시지 핸들링 모듈 구성[7]

을 통해 들어오는 분산이벤트가 있다. 지역이벤트는 노드 내부의 SpM 스레드나 다른 SvM 스레드에서 발생시키는 이벤트이다. 분산이벤트는 패킷의 Payload 부분에 이벤트를 발생시킬 Channel의 위치를 첨부해 주는 방식으로 이용할 수 있다.

그림 11은 NanoQ+의 RF 메시지 핸들링 모듈을 보여준다. Zigbee 무선 모듈을 통하여 메시지가 수신되면 NanoQ+의 MAC Layer와 Link Layer를 거쳐 Network Layer에 패킷이 전달된다.

사용자는 NanoQ+의 패킷 수신시 호출될 함수를 지정하는 통신관련 API인 mlme_ll_link_start() 또는 mlme_start_request()를 이용하여 rf_recv_data()를 콜백함수로 지정한다. 외부로부터 패킷이 수신되면 통신 인터럽트가 발생되고, NanoQ+의 통신 Layer 들을 거쳐 사용자가 지정한 콜백 함수인 rf_recv_data()가 자동으

로 호출되며 수신한 패킷을 처리하게 된다. 수신된 패킷의 Payload 부분을 검사하여 ITC 관련 이벤트가 필요한 경우 해당 ITC_Channel로 이벤트를 보내어, 이벤트를 대기중인 SpM을 Ready List로 입력 시킨다.

4.5 실시간 스케줄링 정책

μTMO-NanoQ+에서는 EDF(Earliest Deadline First) 스케줄링 정책을 센서 네트워크에 맞도록 변형시킨 μTMO-EDF 스케줄링 정책을 사용한다. μTMO-EDF 스케줄링 정책의 특징은 다음과 같다.

(1) EDF 스케줄링 정책

(2) Context Switch Threshold 값 적용

(3) Power Aware 스케줄링 기법 적용

센서 노드의 낮은 컴퓨팅 파워 때문에 실시간 스케줄링을 위해서 많은 계산을 할 경우 다른 작업이 방해 될 수 있다. 그렇기 때문에 실시간성 지원을 위한 스케줄링을 최소화하는 것이 필요하다. 본 논문에서는 남은 데드라인이 가장 짧은 스레드를 우선적으로 수행함으로써 최대한 많은 작업의 데드라인을 만족 시킬 수 있게 해주는 EDF 스케줄링을 기본으로 하고 여기에 센서 노드를 위한 몇 가지 최적화 기능을 추가 하였다.

많은 실시간 스케줄링 기법 중 EDF를 선택한 이유는 낮은 컴퓨팅 파워 환경에서 복잡한 스케줄링 계산 없이 MCU의 컴퓨팅 파워를 스레드를 수행하는데 최대한 활용하고자 하는데 있다.

Context Switch Threshold 및 Power Aware 기법을 적용하여 센서 노드 특성에 알맞은 스케줄링 방식으로 변경 하였다.

새롭게 들어온 스레드의 남은 데드라인이 더 짧은 경우 Context Switch가 일어나야 하지만 단순히 남은 데드라인 뿐만 아니라 미리 정해놓은 Threshold 값을 더해주어 비교한다. 현재 수행중인 스레드의 데드라인 값과 새로운 스레드의 데드라인 값과의 차이가 Threshold 값 보다 적게 차이가 나는 경우에는 Context Switch Overhead를 고려하여 Context Switch가 일어나지 않도록 한다.

μTMO-NanoQ+에서는 NanoQ+에서 제공하는 통합 개발도구인 Nano Esto의 커널 설정 메뉴에 Context Switch Threshold 값을 사용자가 ms 단위로 지정하도록 하고 있다.

Power Aware 기법은 현재 노드의 전력 상태에 따라 스레드를 선별적으로 수행할 수 있도록 해주는 기법이다. 각 스레드에 작동 가능한 파워 레벨을 미리 정해놓고 현재 노드의 전력 상태가 지정된 파워 레벨 보다 낮은 경우에는 해당 스레드는 동작시키지 않는 방법으로 동작한다.

NanoQ+에서 제공하는 Battery Monitor 모듈을 이용하여 모듈의 전력 상태를 얻어오고, 스레드 생성 시 사용자가 추가한 정보인 스레드의 수행 가능 Power Level을 이용하여 스케줄링 시 현재의 전력 상태보다 높은 Power Level의 스레드에게는 수행 권한을 주지 않는 방식으로 동작한다. 예를 들어, 사용자가 스레드 생성 시 3 Level에서 수행 하도록 지정하였다면, 노드의 파워 상태가 3 Level 보다 낮아진다면 스케줄링 시 해당 스레드는 제외가 된다.

5. 실험 및 성능 평가

5.1 SpM 스레드 실험

* 실험 환경

- SpM1(P1): 주기 500ms, 데드라인 500ms
- SpM2(P2): 주기 1200ms, 데드라인 500ms
- SpM3(P3): 주기 700ms, 데드라인 500ms

본 실험에서는 다수의 SpM 스레드를 등록하는 경우 정확한 시간에 Invocation이 일어나는지 확인하고자 하는 실험이다. 주기를 테스트하기 위함이므로 데드라인은 모두 같은 값으로 설정 하였다.

그림 12는 이론적 계산된 각 SpM이 invocation되어야 하는 시간을 보여준다. P1, P2, P3는 실험환경에서 제시한 SpM1, SpM2, SpM3를 나타낸다. 0ms에 동시에 Ready 상태가 되어 스케줄링이 되는 경우 SpM1, SpM2, SpM3가 각각 Invocation 되어야 하는 시간을 나타내고 있으며, 그림 13은 실험을 통하여 각각의 SpM들이 실제로 Invocation된 시간을 나타낸다. 그림 12에서 제시하는 이론적인 Invocation 시간과 그림 13의 실험 데이타를 비교하여 실제 원하는 시간에 SpM이 Invocation 되었는지 비교하여 보았다.

실제 노드에서 실험한 그림 13을 보면 SpM 1이 553ms부터 시작하여 500ms 주기로 동작하는 것을 알 수 있다. 53ms가 지난 상태에서 적용이 되는 이유는 시간을 측정하는 tmr_ms가 센서 보드를 초기화한 뒤 부터 작동하기 때문이다. 실제로 보드를 초기화한 뒤 새로운 SpM 스레드를 생성, 등록하고 스레드를 실제로 시작하기 까지 53ms의 시간이 소요된 것이다. 그러므로 실제 프로그램이 시작되는 시간은 tmr_ms 53ms부터이다. SpM 2의 경우 1255ms에 수행 되었다. 실제 수행되어야 하는 시간에서 2ms 후에 수행이 되었다. SpM 3의 경우에는 755ms, 1455ms, 2155ms에 수행되었음을

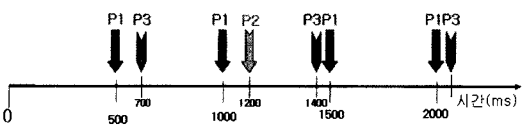

그림 12 각 SpM의 Invocation 시간

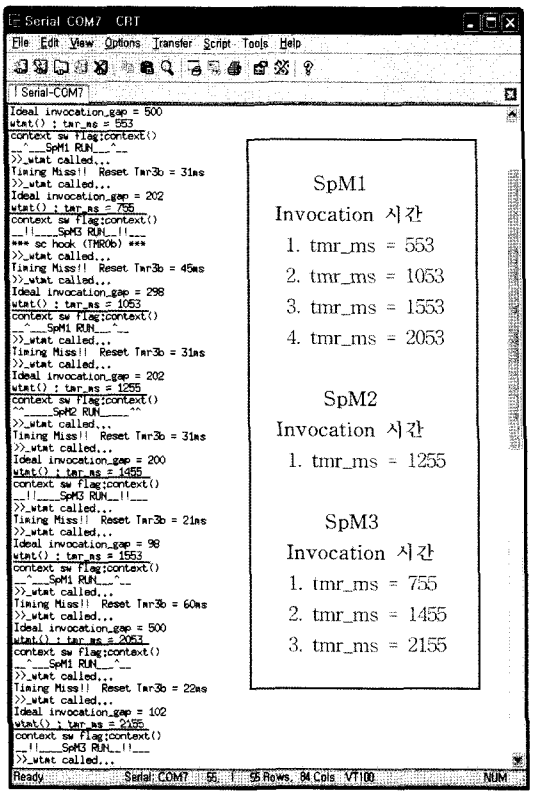

그림 13 다중 SpM 스레드 동작 테스트

볼 수 있다. 이 결과를 이론적인 결과인 그림 12와 비교해 보면 커널에서 허용하는 2ms 내외의 오차 범위 내에서 수행이 진행되는 것을 알 수 있다.

5.2 실시간 서비스 스레드(SvM) 동작 실험

본 실험은 ITC-Channel을 통해 보내진 이벤트에 반응하여 등록된 SvM 스레드가 정상적으로 invocation 되는지 확인하기 위한 실험이다. 이 실험을 통해 ITC-Channel에 이벤트를 보내는 send_event_to_channel() 함수와 SvM 스레드 모델이 정상적으로 수행됨을 확인한다.

* 실험 환경
  - SpM1 : 주기 500ms, 데드라인 500ms
  - SvM1 : 채널 1번, 데드라인 500ms
  - SpM 스레드가 3회 반복 수행될 때마다 ITC-Channel로 이벤트를 보내서 SvM을 invocation 시킨다.

주기적인 SpM 1에서 ITC-Channel로 이벤트를 보내주는 System API를 사용하여 실제로 ITC-Channel로 이벤트가 전달되는지 확인 할 수 있도록 하였다. SpM 1 스레드는 500ms마다 깨어나서 동작을 하며, SpM 1이 3회 수행될 때 마다 채널 1번으로 이벤트를 보내도록 하였다.

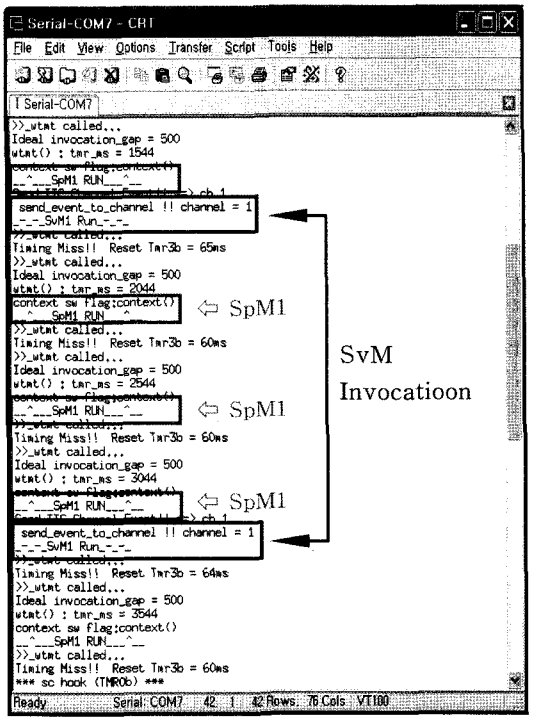

그림 14 SvM 스레드 동작 테스트

그림 14는 실험 결과를 보여 준다. 1544ms 시간에 Invocation된 SpM 1이 채널 1번에 이벤트를 전송하여 SvM 1 이 수행되었다. 3044ms에 Invocation SpM 1이 채널 1에 이벤트를 보내어 SvM 1이 활성화 되었다.

이 실험을 통하여 ITC-Channel 보낸 이벤트에 의하여 SvM 스레드가 정상적으로 동작함을 알 수 있다.

5.3 스케줄링 정책 실험

본 실험은 EDF를 기반으로 한 스케줄링 정책이 정상적으로 작동하는지 알아보기 위한 실험이다.

* 실험 환경
  - SpM1 : 주기 - 1500ms , 데드라인 - 1500ms
  - SpM2 : 주기 - 1600ms , 데드라인 - 500ms
  - SpM1의 실행 프로그램은 100ms마다 메시지를 출력, 1000ms동안 10개의 실행 메시지를 출력하는 프로그램이다.

그림 15를 살펴보면 6048ms에 SpM1이 먼저 Invocation 되고 있다. SpM1은 100ms마다 문자열을 출력하면서 CPU 자원을 소모한다. 6450ms에 SpM2가 WTMT에 의해 Invocation 된 뒤 Ready 리스트로 입력이 되어 스케줄링을 기다리게 된다. 이때, SpM 2의 남아 있는 데드라인은 500ms인데 비해 SpM1의 남은 데드라인은 1098ms이다. SpM 2의 남은 데드라인이 더 짧기 때문에 SpM2는 SpM 1을 선점 하여야 한다.

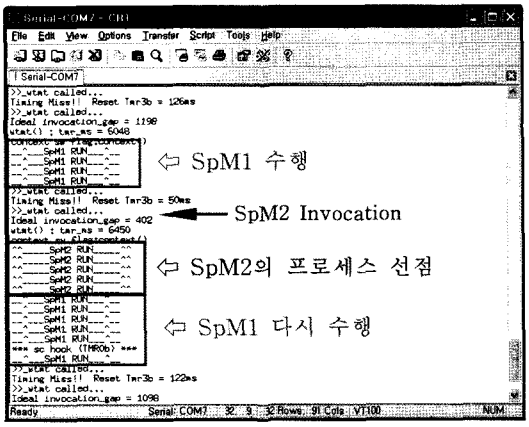

그림 15 EDF 기반 스케줄러 테스트

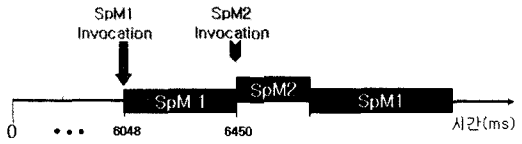

그림 16 EDF 기반 스케줄러 테스트 재구성

그림 16은 그림 15의 결과를 보기 편하도록 그림으로 표현한 것이다.

결과를 보면 4개의 SpM1 메시지가 출력되어 약 400ms의 시간이 흐른 뒤에 WTMT에 의하여 SpM2가 Invocation 된 것을 알 수 있다. SpM2의 남은 데드라인이 SpM1의 남은 데드라인 보다 더 짧기 때문에 SpM2가 선점하여 작업을 하였음을 볼 수 있다.

위 결과를 토대로 보면 EDF 기반의 스케줄링 정책이 정상적으로 동작하고 있음을 알 수 있다.

5.4 μTMO-NanoQ+ 성능 평가 및 분석

표 1은 다른 센서네트워크 운영체제와 μTMO-NanoQ+를 비교한 표이다. TinyOS와의 비교하면, TinyOS는 컴포넌트에 기반한 이벤트 드리븐 방식을 사용하는 것에 반하여 MANTIS, NanoQ+, μTMO-NanoQ+는 멀티 스레드 방식을 채택하고 있다. TinyOS의 경우 nesC(Network Embedded System C) 언어를 이용하여 컴포넌트 기반의 이벤트 드리븐 방식의 프로그램을 작성한다. 이는 이벤트 기반 동시성 모델을 제공하여 주지만 프로그래머가 새로운 언어와 프로그래밍 방법을 배워야 하는 단점이 있다.

MANTIS, NanoQ+, μTMO-NanoQ+에서 채택한 멀티 스레드 방식은 이미 많은 프로그래머가 사용하고 있는 일반적인 환경으로 프로그래머가 별도의 학습 없이 쉽게 이해가 가능하다는 장점이 있다.

NanoQ+와 μTMO-NanoQ+의 차이점을 살펴보면 NanoQ+의 경우 멀티 스레드 방식을 지원하는데 반하여, μTMO-NanoQ+는 TMO 모델을 지원하고 있다. 3.1절에서 살펴보았듯이 TMO모델을 이용하면 센서 네트워크의 일반적인 응용을 쉽게 모델링 할 수 있다. 커널에서 TMO모델의 수행환경을 제공함으로써 TMO 모델을 이용하여 손쉽게 실시간 응용프로그램을 작성할 수 있으며, 실시간 스케줄링 기법의 제공으로 실시간성을 지원할 수 있게 되었다.

TinyOS는 이벤트 드리븐 방식으로 동작하는 운영체제로 이벤트에 반응한 컴포넌트 모듈간의 동작에 의하여 응용이 만들어 진다. μTMO-NanoQ+는 TMO 모델의 지원을 통하여 이벤트에 대한 반응은 SvM을 통하여 지원하며, 주기적으로 일어나는 동작은 SpM을 이용하여 지원하도록 구성되어있다. TinyOS의 경우 주기적인 동작을 기술하기 위하여 사용자가 타이머 컴포넌트를 활용하여 주기적인 이벤트에 대한 동작을 기술하여 주어야 하지만, μTMO-NanoQ+에서는 TMO 모델의 지원을 통하여 커널 단에서 이러한 주기적인 모델을 지원하고 있다. 사용자가 주기적인 동작을 표현하기 위하여 필요한 작업은 단지 SpM 스레드 등록 시, 실행 주기를 기록 하는 과정 이외에 추가적인 작업을 필요하지 않는다. 3.1절에서 살표 보았듯이 센서 네트워크 응용에서 대부분의 센싱 작업은 주기적으로 일어나기 때문에 TMO 모델의 SpM과 SvM을 활용하면 센서 노드의 동

표 1 센서 네트워크 운영체제 비교[13]

OS	동적모듈 로드	실행모델	실시간성	네트워크	관련도구
TinyOS	미지원	컴포넌트 베이스 이벤트 드리븐	X	B-mac, Flooding	TOSSIM, TOSVIS, TinyDB, TinySEC
SOS	지원	모듈베이스 이벤트드리븐	X	UBMAC, Tree Routing	-
MANTIS	미지원	멀티 스레드	△	MAC, X-MAC	-
NanoQ+	미지원	멀티 스레드	△	Zigbee, EAMR	NanoEsto, NanoMON
μTMO-NanoQ+	미지원	TMO모델 기반 멀티스레드	○	Zigbee, EAMR	NanoEsto

작을 매우 손쉽게 모델링 할 수 있는 장점이 있다.

센서 네트워크용 운영체제에서는 실행 중에 사용 가능한 RAM의 크기가 4KB에 불과 하기 때문에 커널 크기는 중요한 비교 요소가 된다.

동일한 동작을 하는 응용프로그램을 각각 preemption_rr 스케줄러와 μTMO-EDF 스케줄러를 이용하여 작성한 뒤 비교 하였다. 스케줄러 이외의 모듈 선택사항은 동일하게 구성하였다.

표 2 Nano-Q+의 스케줄러와 크기 비교

	Preemption_rr	μTMO-EDF	비 교
스케줄러 크기	3,376 Byte	5,128 Byte	+ 1.7 KB
응용1 포함 크기	40.2KB	43.2KB	약 7.4 % 증가
응용2 포함 크기	47KB	50.1KB	약 6.6 % 증가

비교 하였다. 응용1은 μTMO-EDF 스케줄러의 테스트를 위한 3개의 SpM 스레드를 이용한 간단한 메시지 출력을 보여주는 응용프로그램이며, 응용2는 실제 센서 노드를 부착하여 센서가 감지한 값을 읽어들여 출력해 주는 응용프로그램이다. 응용2에서는 ADC 관련 모듈들이 함께 사용되므로 용량이 더욱 커진다.

μTMO-EDF 스케줄러 모듈은 기존의 Preemption_rr 스케줄러에 비하여 약 1.7KB 정도의 스케줄러 크기 증가가 있다. 이는 실시간 지원을 위해 시간정보를 저장하고 데드라인과 주기를 계산하는 코드의 증가와 WTMT, ITC-Channel 등 μTMO 모델을 지원하기 위한 추가적인 기능을 모두 포함하기 때문이다. 하지만 응용프로그램과 커널의 다른 모듈까지 포함할 경우에는 3KB의 크기 증가를 보여준다. μTMO- EDF 스케줄러를 채택함으로써 늘어나는 코드의 크기는 3KB로 동일하기 때문에 응용프로그램의 크기가 커질수록 μTMO-EDF 스케줄러를 사용함으로써 추가되는 오버헤드의 비율은 작아진다.

기존의 범용 스케줄러는 단순 우선순위 기반 선점 스케줄링을 하기 때문에 주기적인 동작을 모델링하기에는 무리가 있다. 실제 응용프로그램에서 halWait()를 이용하여 시간을 멈추어야 하는데 halWait()가 호출되어 아무런 작업을 하지 않는 동안에도 실제로 어셈블리 레벨에서는 무한 루프를 순환하면서 CPU를 점유하기 때문에 다른 작업에게 시간을 넘겨주지 않는다. 이는 μTMO-EDF 스케줄러가 비동기 타이머를 이용하여 대기하는 동안 다른 작업을 수행하는 것과는 많은 차이가 있다. preemption_rr 스케줄러는 우선순위 기반의 스케줄링만 해줄 뿐이기 각 스레드 사이의 타이밍 제약을 명확하게 표현하기 어렵다.

μTMO-EDF 스케줄러는 전체 코드 대비 약 7% 정도의 크기 증가를 통하여 완전한 타이밍 제약을 표현할 수 있으며, SpM 스레드가 주기를 기다리는 동안 다른 스레드를 수행 할 수 있기 때문에 MCU를 더 효율적으로 사용할 수 있다. 또한 기존 스케줄러에서는 지원하지 못하는 데드라인 기반의 실시간 스케줄링 기능을 제공하여 준다. 사용자의 입장에서는 실시간 프로그래밍 할 때 높은 추상화 수준에서 표현이 가능하다는 장점이 있다.

6. 결론 및 향후 과제

센서 네트워크의 응용분야가 점차 넓어지면서 실시간성에 대한 요구가 생겨나기 시작하였다. 하지만 기존의 센서 네트워크용 운영체제는 자원의 효율적인 운영 측면에 중점을 두고 개발되었기 때문에 실시간 요구사항을 만족시키기 어려웠다. 실시간 커널은 기존의 커널에 비하여 스케줄링 단계에서의 계산량이 늘어나고 많은 자원을 소모하게 된다. 센서 노드는 자원이 매우 제한되어 있는 동작 환경이므로 센서 노드에서의 실시간 운영체제는 실시간 지원에 따른 오버헤드를 최소화 하여야 한다.

본 논문에서는 군사분야와 같이 특수한 센서 네트워크 응용환경에서의 실시간 필요성에 대해 알아보고, TMO 모델을 실시간 센서 응용프로그램에 적용하여 얻을 수 있는 장점에 대하여 설명하였다. TMO 모델을 그대로 센서 네트워크에 적용 시 예상되는 문제점을 진단하고. 센서 네트워크에 알맞게 경량화 시킨 μTMO 모델을 제안하였다. μTMO 모델을 ETRI에서 개발한 멀티 스레드 기반의 센서 노드용 운영체제인 Nano-Q+에 적용시킨 μTMO-NanoQ+를 구현 하였다. 본 논문에서 제안하고 설계한 μTMO-NanoQ+는 Nano-Q+ 라이센스 규칙에 따라 한양대학교 운영체제 연구실(http://osnn.hanyang.ac.kr) 홈페이지에서 오픈 소스로 공개하고 있다.

향후 과제로, μTMO-NanoQ+에 적용시킨 실시간 스케줄러인 μTMO-EDF를 더욱 경량화시켜 센서 노드에 부담을 줄일 수 있는 방향으로 연구를 진행하여야 한다. 또한, 스레드간 공유자원에 의한 데드락 방지를 위한 프로토콜의 적용이 필요하며, 분산 이벤트를 더욱 효과적으로 처리할 수 있는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, Erdal Cayirci, "A Survey on Sensor Networks," IEEE Communications Magazine, pp. 102-114, August 2002.

[2] TinyOS, (Web Page) www.tinyos.net

[3] S. Bhatti et al., "MANTIS OS: An Embedded

Multithreaded Operating System for Wireless Micro Sensor Platforms," ACM/Kluwer Mobile Networks and Applications (MONET), Special Issue on Wireless Sensor Networks, Vol.10, No.4, August 2005.
[4] John A. Stankovic, "Research challenges for wireless sensor networks," ACM SIGBED vol1, Issue 2, pp. 9-12, Jul. 2004.
[5] T. He, J. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A Stateless Protocol for Real-Time Communication in Ad Hoc Sensor Networks," IEEE ICDCS, May 2003.
[6] C. Lu, B. Blum, T. Abdelzaher, J. Stankovic, and T. He, "RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor-Networks," RTAS, June 2002.
[7] Nano-Qplus, (Web Page) http://qplus.or.kr
[8] Seungmin Park, Jin Won Kim, Kee-Young Shin, Daeyoung Kim, "A nano operating system for wireless sensor networks," Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference, Vol.1, pp. 20-22, Feb. 2006.
[9] Seungmin Park, Jin Won Kim, Kwangyong Lee, Kee-Young Shin, and Daeyoung Kim, "Embedded Sensor Networked Operating System," ISORC, 2006.
[10] K.H. Kim and Kopetz, "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials," Proc. 18th IEEE Computer Software and Applications Conference, pp. 392-402, November 1994.
[11] 박지강, 서한석, 김정국, "분산 실시간 객체 TMO를 위한 MicroC / OS - II 실시간 스케줄러의 설계 및 구", 한국정보과학회 한국컴퓨터종합학술대회 2005 논문집(A), 2005. 7, pp. 835-837.
[12] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," Journal of the ACM, Vol.20, No.1, pp. 46-1, 1973.
[13] 송준근, 마평수, 박승민, "센서 네트워크 초소형 OS", 한국통신학회지(정보와통신), 제24권 7호, 2007. 7, pp. 26-35.

이 재 안

2005년 2월 한양대학교 전자컴퓨터공학부 학사 졸업. 2007년 2월 한양대학교 컴퓨터공학과 석사 졸업. 관심분야는 Sensor-network, Real-Time Operating System

허 신

1973년 2월 서울대학교 전기공학 학사 졸업. 1979년 5월 미국 University of Southern California 전산학 석사 졸업. 1986년 5월 미국 University of South Florida 전산학 박사 졸업. 1980년~1986년 University of South Florida 연구원보. 1986년~1988년 The Catholic Uni versity of America 조교수. 1988년~현재 한양대학교 컴퓨터 공학과 교수. 관심분야는 Distributed Computing Systems, Fault-Tolerant System, Real-Time Operating Systems

최 병 규

2002년 2월 한양대학교 전자컴퓨터공학부 학사 졸업. 2004년 2월 한양대학교 컴퓨터공학과 석사 졸업. 현재 한양대학교 컴퓨터공학과 박사과정 수료. 관심분야는 Sensor-network, TMO(Time-triggered Message-triggered Object)