논문 2006-02-82

재구성 가능한 고성능 센서 운영체제를 위한 소프트웨어 아키텍처 설계

(A Software Architecture for Highly Reconfigurable Sensor Operating Systems)

김 태 환, 김 희 철*
(Tae-Hwan Kim, Hie-Cheol Kim)

Abstract : Wireless sensor networks are subject to highly heterogeneous system requirements in terms of their functionality and performance due to their broad application areas. Though the heterogeneity hinders the opportunity of developing a single universal platform for sensor networks, efforts to provide uniform, inter-operable and scalable ones for sensor networks are still essential for the growth of the industry as well as their technological advance. As a part of our work to develop such a robust platform, this paper presents the software architecture for sensor nodes with focus on our sensor node operating system and its configuration methodology. Addressing principle issues in its design space which includes programming execution, task scheduling and software layer models, our architecture is highly reconfigurable with respect to system resources and functional requirements and also highly efficient in supporting multi-threading under small system resources.

Keywords : wireless sensor network, operating system, software architecture, configuration

Ⅰ. 서 론

최근 센서노드는 집적회로 제조 기술의 발달로 인해 저전력, 저비용, 고성능의 임베디드 시스템 형태로 발전하고 있다. 센서 네트워크의 응용 분야는 기존의 사무실, 공장, 자연환경에 대한 감시 및 제어 분야에서 u-시티, u-농촌, u-헬스케어, u-국방 분야로 확대되고 있다. 센서 노드에 탑재되는 센서 운영체제는 응용분야의 특성에 따라 상이한 소프트웨어 및 하드웨어적 요구사항을 갖는다. 이로 인해 센서노드의 임베디드 소프트웨어 개발 시에 요구되는 비용, 시간, 안정성 등이 큰 이슈가 되고 있다.

일반적으로, 센서노드는 제한된 메모리, 전력, 컴퓨팅 능력을 갖기 때문에 센서 노드에 탑재되는 센서 운영체제는 작은 코드 사이즈, 효율적인 자원관리, 재구성 능력 등을 고려하여 설계하여야 한다. 특히, 프로그래밍 모델, 태스크 수행제어 모델, 스케줄링 모델, 소프트웨어 아키텍처 모델은 센서 운영체제의 성능을 결정하는 핵심요소라 할 수 있다.

현재, 센서 네트워크를 환경을 고려한 다양한 센서 운영체제들이 소개되고 있다[1-8]. 이러한, 센서 운영체제들은 다양한 플랫폼 지원, 효율적인 자원관리, 쉬운 응용 프로그램 환경 등을 제공한다. 그러나 이들 센서 운영체제들은 센서노드 응용모델의 요구사항 및 하드웨어 사양이 변경될 경우 OS 레벨에서 많은 소스코드 수정이 요구되며, 소프트웨어 모듈의 재사용성과 이식성이 저하되는 문제점이 있다.

본 논문에서는 기존의 센서 운영체제들이 가지고 있는 문제를 개선하기 위하여 재구성 가능한 고성능 센서 운영체제를 위한 소프트웨어 아키텍처를 제안

* 교신저자(Corresponding Author)
논문접수 : 2007. 12. 09. 채택확정 : 2008. 02. 14.
김태환 : 경북 유비쿼터스 신기술 연구센터
김희철 : 대구대학교 정보통신공학부
※ 본 연구는 정보통신부 및 정보통신연구진흥원의 지원을 받아 수행되었음 (08-기반-13, IT특화연구소: "유비쿼터스 신기술 연구센터" 설립 및 운영)

한다. 제안한 아키텍처는 수직적-수평적 구조의 계층적 소프트웨어 아키텍처 모델을 기반으로 한다. 또한, 계층적으로 구조화된 센서운영체제 설계 모델을 기반으로 구현된 센서 운영체제는 다양한 응용분야에 쉽게 적용 가능하도록 운영체제의 재구성 능력과 소프트웨어 모듈의 재사용 능력이 뛰어난 특징을 갖는다. 제안한 센서 운영체제는 우선순위 기반 선점형 멀티쓰레딩, 작은 코드 및 메모리, 다양한 사용자 API, 고도화된 하드웨어 추상화 계층 등을 지원한다.

본 논문의 구성은 다음과 같다. 2장에서는 센서 운영체제 설계 모델을 설명하고 기존의 센서 운영체제들의 특징을 비교 분석한다. 3장에서는 본 논문에서 제안하는 재구성 가능한 고성능 센서 운영체제의 소프트웨어 아키텍처를 설명한다. 4장에서는 실험을 통해 제안한 센서 운영체제의 성능을 평가하고, 5장에서 결론을 맺는다.

II. 센서 운영체제 설계 모델

1. 설계 모델

센서 네트워크 환경은 아래와 같이 크게 세 가지 측면에서 기존의 무선 네트워크 환경과 상이한 특성을 갖는다. 첫째, 센서 네트워크를 구성하는 노드는 낮은 프로세싱 능력, 작은 프로그램 및 데이터 사이즈, 작은 배터리 용량 등과 같이 자원이 매우 제한되어 있다. 둘째, 센서 노드는 비주기적으로 발생하는 센서 데이터를 처리하고 이를 무선 네트워크를 통해 전송한다. 셋째, 이기종의 센서 노드들로 무선 네트워크가 구성될 수 있으며, 동일한 센서 노드라 할지라도 상이한 역할을 담당한다. 이러한 특징들로 인해 센서노드에 탑재되는 센서 운영체제는 다음과 같은 요구사항을 갖는다.

- 높은 프로그램 능력
- 효율적인 자원관리
- 작은 코드 이미지
- 태스크의 병행제어
- 운영체제의 재구성 능력

위의 요구사항을 만족하고 센서노드의 응용 모델 및 H/W 특성에 적합한 센서 운영체제를 설계하기 위해서는 프로그래밍, 수행제어, 태스크 스케줄링, 아키텍처 모델 등을 고려하여야 한다. 프로그래밍 모델로는 유한상태머신(Finite State Machine, FSM) 기반, 컴포넌트 기반, 전통적인 API 기반 모델 등을 고려할 수 있다. 수행제어 모델로는 이벤트 구동 모델과 멀티쓰레딩 모델이 있다. 태스크 스케줄링 모델에는 비선점형과 선점형 스케줄링 모델이 고려될 수 있다. 센서 운영체제의 아키텍처 모델에는 수직적 구조(vertical Layering)의 추상화 기반 계층적 모델과 수평적 구조(horizontal layering)의 기능 기반의 모델이 있다. 그림 1은 센서 운영체제를 위한 설계 모델들의 장단점을 보여준다.

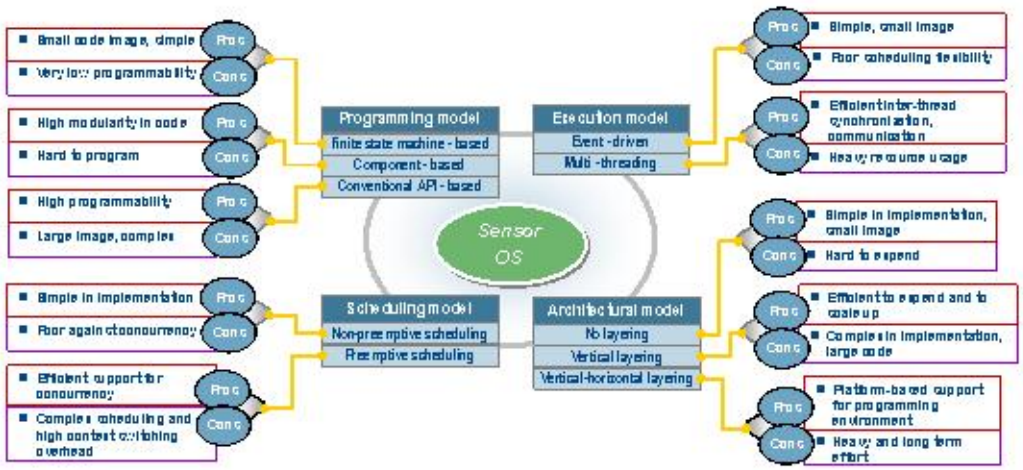


그림 1. 센서 운영체제 설계 모델의 장단점

Fig. 1. Pros and cons of design models of sensor operating systems

2. 센서 운영체제

센서 운영체제 개발 시에 고려될 수 있는 다양한 설계 모델과 함께 현재까지 많은 종류의 센서 운영체제들이 소개되고 있다. 본 소절에서는 현재까지 소개된 국내외의 센서 운영체제 중에서 대표적인 센서 운영체제를 설명한다.

TinyOS[1]는 현재까지 가장 널리 사용되는 센서 운영체제로서 미국의 버클리 대학에서 개발되었다. 컴포넌트 기반과 이벤트 구동 방식의 운영체제로서, 제한된 메모리 공간의 효율적 이용 및 에너지 효율적인 자원관리 등의 특징을 갖는다. 반면, 멀티태스킹을 지원하지 않기 때문에 한 번에 하나의 태스크만 수행되며, 응용 프로그램 개발을 위해 nesC라고 하는 새로운 프로그램 언어를 사용해야 하는 단점이 있다.

Mantis[2]는 미국 콜로라도 대학교에서 개발된 센서 운영체제로서 선점 가능한 멀티쓰레딩을 지원한다. 특히, 계층적 멀티쓰레딩과 타임 슬라이싱 기반의 선점형 스케줄링을 통해 복잡성과 시간 제약성 태스크를 효과적으로 처리한다.

SOS[3]는 미국 UCLA에서 개발된 이벤트 구동 기반의 비선점형 센서 운영체제이다. SOS는 메시지 전송 메커니즘, 동적 메모리 할당 및 참조를 통해 동작중인 운영체제의 중단 없이 네트워크를 통해 바이너리 모듈의 추가 및 삭제가 가능한 특징을 갖는다.

Contiki[4]는 스웨덴 컴퓨터공학 연구소에서 개발한 이벤트 구동 기반의 비선점형 센서운영체제 이다. Contiki는 운영체제 동작에 필요한 메모리 요구량이 비교적 작고 런타임에도 동적인 프로그램 추가 및 삭제가 가능하다. 또한, 제한적으로 선점형 멀티쓰레딩을 지원한다.

µC/OS-II[5]는 JEAN J. LABROSSE에 의해 공개용으로 만들어진 선점형 실시간 커널로서 다양한 임베디드 시스템에 사용되고 있다. uC/OS-II는 CPU 의존적인 부분만 어셈블리어로 작성되어 있으며, 대부분의 코드가 ANSI C로 작성되었기 때문에 다양한 CPU로의 포팅이 용이한 특징을 갖고 있다. 또한, 응용 프로그램에서 필요한 기능만을 이미지에 포함할 수 있도록 설계되어 있으므로 코드 공간이나 데이터 공간의 낭비를 줄일 수 있다.

AUTOSAR[6]는 자동차 내 전장장치의 ECU를 위한 표준 SW 아키텍처로서 수직적/수평적 SW 아키텍처, SW의 재사용성, 강력한 하드웨어 추상화 계층을 지원한다. AUTOSAR 아키텍처는 응용계층인 컴포넌트 기반의 소프트웨어, 컴포넌트 간의 통신을 위한 RTE(run-time environment), 기본 소프트웨어(basic software, BSW)로 구성된다. 기본 소프트웨어는 상위 계층을 위한 서비스, 운영체제, 추상화 모듈로 구성되며, 운영체제로는 자동차용 실시간 운영체제 표준인 OSEK/VDX가 사용된다.

NanoQplus[7]는 한국전자통신연구원에서 개발한 선점형 멀티스레드 기반 센서 운영체제이다. NanoQplus는 확장성과 시스템 재구성이 비교적 용이하며, POSIX 기반의 표준 API 중에서 멀티 쓰레드 관련 서브셋을 지원한다. 특히, NanoQplus는 기존의 전통적인 운영체제의 형태를 갖추면서 불필요한 모듈을 제거함으로써 센서 운영체제를 경량화 하였다.

RETOS[8]는 연세대학교에서 개발한 선점형 멀티스레드 기반의 센서 운영체제이다. RETOS는 쓰레드 기반 프로그래밍 인터페이스, 시스템 탄력성,

표 1. 센서 운영체제들의 설계 모델 비교

Table 1. Design model comparison of sensor operation systems

	TinyOS	MANTIS	SOS	Contiki	µC/OS-II	AUTOSAR (OSEK)	Nano Qplus	RETOS
	UC Berkeley	Colorado Univ.	UCLA	Swedish institute	JEAN J. LABROSSE	AUTOSAR Consortium	ETRI	Yonsei Univ.
Programming model	Component based	Conventional API based	Conventional API based	Conventional API based	Conventional API based	Component based	Conventional API based	Conventional API based
Execution control	Event-driven	Multi-threading	Event-driven	Event-driven	Multi-threading	Multi-threading	Multi-threading	Multi-threading
Task scheduling	Non-preemptive	Preemptive	Non-Preemptive	Non-preemptive	Preemptive	Mixed-preemptive	Preemptive	Preemptive
Architecture model	Vertical Layering	Vertical Layering	Vertical Layering	-	Vertical layering	Vertical /horizontal layering	(Limited) Vertical Layering	Vertical Layering

커널의 동적 재구성 및 확장성, WSN 지향 네트워크 추상화 계층을 지원하는 특징을 갖고 있다.

표1은 대표적 센서 운영체제들의 프로그래밍, 수행제어, 태스크 스케줄링, SW 아키텍처 모델을 비교하였다.

Ⅲ. 재구성 가능한 고성능 센서 운영체제 설계

1. SW 아키텍처

센서 운영체제에서 소프트웨어 아키텍처 모델은 운영체제의 재구성 능력 및 소프트웨어 모듈의 재사용성에 큰 영향을 준다. 수직적 계층 모델은 운영체제의 확장이 용이한 반면 소스 사이즈가 커지고 구현이 복잡한 단점이 있다. 수직적-수평적 계층 모델은 소프트웨어 모듈의 재구성 및 재사용이 용이하며 플랫폼 기반의 프로그래밍 환경을 지원한다. 반면, 소프트웨어 모듈간의 인터페이스 및 의존성을 잘 고려하여야 하며, 소스 사이즈와 개발 기간이 늘어나는 단점이 있다.

그림 2는 본 논문에서 제안하는 UOS(ubiquitous sensor operating system)의 소프트웨어 아키텍처를 나타내었으며, 수직적-수평적 아키텍처 모델을 기반으로 한다. 제안한 센서 운영체제의 소프트웨어 아키텍처는 크게 서비스 계층, 플랫폼 추상화 계층, MCU 추상화 계층으로 구성된다. 서비스 계층은 우선순위 기반 선점형 멀티쓰레딩 지원을 위한 커널, 시스템 서비스, 외부 메모리 서비스, 통신 서비스, 센서 제어 서비스 모듈로 구성된다. 플랫폼 추상화 계층은 다양한 센서 노드 플랫폼 지원을 위한 외부 장치 드라이버로 구성되며, 하위 계층인 MCU 추상화 계층의 장치 드라이버를 사용하여 구현된다. MCU 추상화 계층은 센서 노드에 사용되는 MCU 내부의 자원에 대한 하드웨어 추상화 계층으로 상위 계층에서 하드웨어 독립적으로 모듈을 구성할 수 있도록 한다.

센서 운영체제 설정 관리자는 사용자의 요구사항에 적합하도록 각 계층의 소프트웨어 모듈에 대한 추가, 제거, 설정 기능을 담당한다. 그림 3은 설정

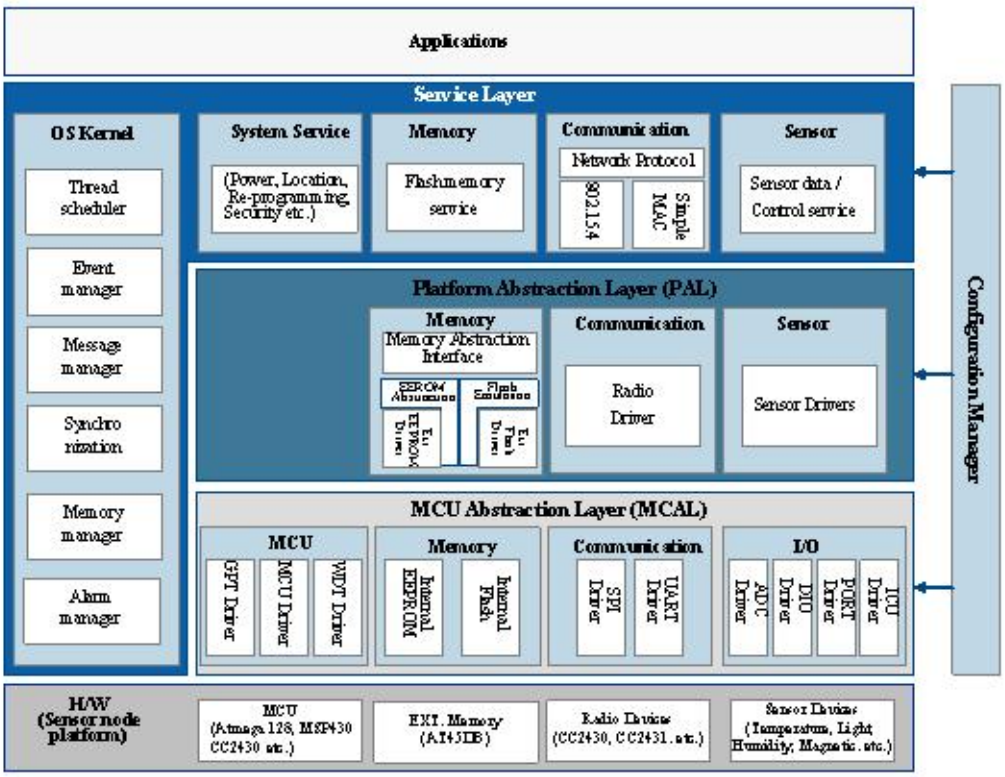


그림 2 재구성 가능한 센서 운영체제를 위한 소프트웨어 아키텍처

Fig. 2 Software architecture for highly reconfigurable sensor operating system

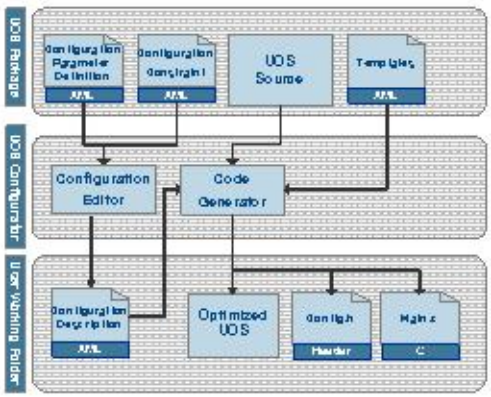


그림 3. 센서 운영체제 설정 관리자

Fig. 3. Configurator for sensor operating system

관리자의 구조를 나타낸다. 코드 생성기는 운영체제 설정 파라미터가 정의된 XML 문서와 각 모듈간의 의존성 정보가 포함된 XML 문서를 입력받아 사용자 요구사항에 적합한 센서 운영체제 코드를 생성한다.

본 논문에서는 제안한 센서운영체제 소프트웨어 아키텍처의 핵심 모듈인 OS Kernel의 구조에 대해 설명한다.

2. 센서 운영체제의 커널

본 논문에서 제안한 센서 운영체제의 커널은 우선순위 기반 선점형 멀티쓰레딩 실행 모델을 기반으로 한다. 각 쓰레드는 그림 4와 같이 DORMANT, READY, WAITING, SLEEPING, RUNNING 등 5가지 상태중 하나의 상태를 갖는다. 제안한 센서 운영체제 커널은 사용자에게 쓰레드 관리, 동기화, 메시지, 이벤트, 알람 API를 지원한다.

그림 5는 쓰레드 제어 정보가 담긴 쓰레드 제어 블록(TCB)의 자료구조를 보여주고 있으며, 쓰레드의 스택정보, 쓰레드 ID, 우선순위, 시간 지연 정보 등이 포함된다. 사용되지 않는 TCB는 프리 리스트로 관리되며, 쓰레드가 생성될 때 할당된다.

제안한 센서 운영체제의 스케줄러는 기본적으로 6단계의 우선순위를 지원하며, 동일 우선순위 쓰레드에 대해서 라운드로빈 방식의 스케줄링은 지원하지 않는다. 그 대신 어플리케이션 프로그래머가 해당 쓰레드의 CPU 선점을 직접 제어할 수 있도록 표 2와 같은 다양한 사용자 API를 지원한다. 그림

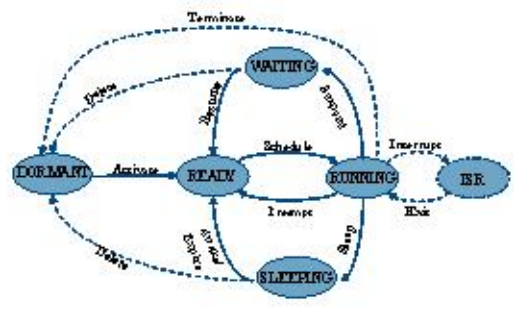


그림 4. 쓰레드 상태 천이도

Fig. 4. Thread transition state

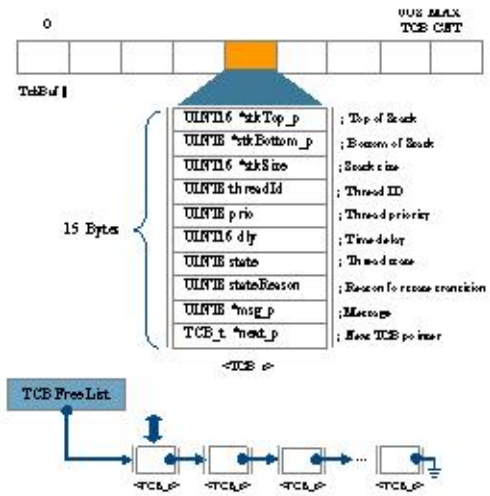


그림 5. 쓰레드 제어 블록의 자료구조

Fig. 5. Data structure of thread control block

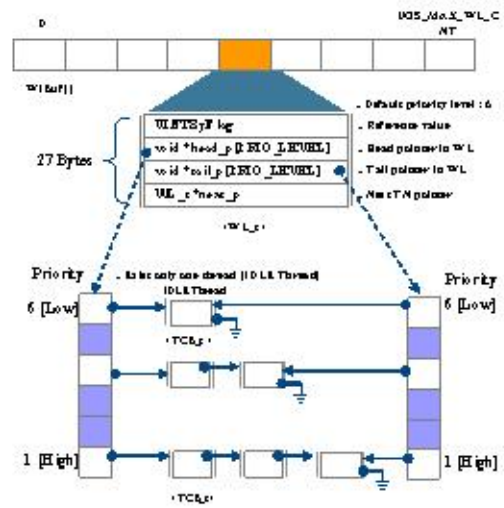


그림 6. 대기 리스트 자료구조

Fig. 6. Data structure of waiting list

표 2. 쓰레드 제어를 위한 사용자 API
Table 2. User APIs for thread control

User APIs	Description
Uos_thread_create()	• Thread 생성
Uos_thread_delete()	• Thread 삭제
Uos_thread_del_request()	• Thread 삭제 요청
Uos_thread_suspend()	• Thread 일시 중단
Uos_thread_resume()	• Thread 동작 재개
Uos_thread_terminate()	• 정상 종료된 Thread를 Dormant 상태로 천이
Uos_thread_prio_change()	• Thread 우선순위 변경

6은 스케줄링을 위한 대기 리스트의 자료구조를 보여준다. 8비트의 yFlag는 언맵(unmap) 테이블을 통해 대기 리스트에서 가장 높은 우선순위를 갖는 쓰레드(HPT)를 검색하기 위해 사용된다.

제안한 센서 운영체제 커널은 자원관리 및 쓰레드간 통신 위해 표3과 같이 다양한 사용자 API를 제공한다. 세마포어와 뮤텍스는 쓰레드간 공유자원에 대한 동기화 및 상호배제를 위해 사용된다. 메시지 메일박스와 메시지 큐는 쓰레드간 메시지 교환을 통한 통신을 위해 사용된다. 이벤트는 쓰레드가 여러 이벤트 발생에 대해 동기화하고 이벤트를 효율적으로 처리할 수 있도록 한다.

표 3. 자원관리 및 쓰레드간 통신을 위한 사용자 API
Table 3. Resource management and inter-thread communication APIs

	APIs	Description
세마포어	Uos_sem_create()	세마포어생성
	Uos_sem_delete()	세마포어삭제
	Uos_sem_request()	세마포어 얻기
	Uos_sem_getNoWait()	대기상태로 전환없이 세마포어 얻기(ISR용)
	Uos_sem_release()	세마포어 반환
뮤텍스	Uos_mutex_create()	뮤텍스 생성
	Uos_mutex_delete()	뮤텍스 삭제
	Uos_mutex_request()	뮤텍스 얻기
	Uos_mutex_getNoWait()	대기상태로 전환없이 뮤텍스 얻기(ISR용)
	Uos_mutex_release()	뮤텍스 반환
메일박스	IUos_mb_create()	메시지 생성
	Uos_mb_delete()	메시지 삭제
	Uos_mb_get()	메시지 얻기
	Uos_mb_getNoWait()	대기상태로 전환 없이 message 얻기(ISR용)
	Uos_mb_send()	message 보내기
메시지큐	Uos_mq_create()	메시지큐 생성
	Uos_mq_delete()	메시지큐 삭제
	Uos_mq_get()	메시지큐에서 메시지열 받기
	Uos_mq_getNoWait()	대기상태로 전환 없이 메시지큐에서 메시지열 받기(ISR용)
	Uos_mq_send()	메시지큐에 메시지열 보내기
	Uos_mq_flush()	메시지큐 비우기
이벤트	Uos_event_init()	이벤트 제어 로직 생성
	Uos_event_delete()	이벤트 플래그 그룹 삭제
	Uos_event_set_clear()	현재 이벤트 발생여부를 이벤트 플래그 그룹에 반영
	Uos_event_wait()	쓰레드가 이벤트를 설정 후 이벤트 발생 대기
	Uos_event_get_state()	현재 event의 발생여부에 대한 상태 얻기
알람	Uos_thread_dly()	틱단위로 Thread 지연
	Uos_thread_dly_resume()	지연중인 Thread 재개
	Uos_thread_dly_HMSM()	시분초밀리 단위로 Thread 지연
	Uos_alarm_set()	Software Timer 생성 및 start
	Uos_alarm_cancel()	Software Timer stop 및 삭제
	Uos_alarm_get_time()	Remaining Time 리턴

Ⅳ. 실험 및 평가

제안한 센서 운영체제의 성능을 실험하기 위해 경북 유비쿼터스 신기술 연구센서에서 개발한 SN-100 센서 노드 플랫폼이 사용되었다. SN-100 센서노드 플랫폼은 그림 6과 같이 ATMEL사의

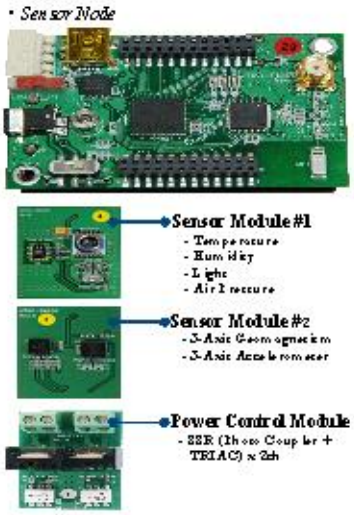


그림 6. 센서노드 플랫폼
Fig. 6. Sensor node platform

표 4. 스케줄링 관련 항목 비교
Table 4. Comparison of scheduling related items

	Mantis	μC/OS-II	OSEK	NanoQplus	UOS
Max. # of threads	Configurable (default :12)	64	8/16	8	Configurable (default :6)
Thread ID	Dynamic	Priority value	0~7/15	0~7	Dynamic
Ready queue	32bit-ready list per priority	64Entry-table with aux. data	Implementation dependent	16bit-ready list per priority	32bit-ready list per priority
Priority level	5	64	Implementation dependent	6	6
thread per priority	No limit	1	No limit	5	No limit

ATmega128 MCU, IEEE 802.15.4/Zigbee를 지원하는 TI사의 CC2420, 512K byte의 외부 플래쉬 메모리 등으로 구성되어 있다.

구현된 센서 운영체제가 SN-100 센서노드 플랫폼에 탑재되었을 때, 커널의 스케줄링 및 문맥교환 오버헤드, 커널 및 MCU 추상화 계층의 데이터 및 코드 사이즈 측정 실험이 수행되었다. 표 4는 기존의 멀티쓰레딩 기반의 센서 운영체제와 본 논문에서 제안한 센서 운영체제의 스케줄링 관련 항목을 비교하였다. 그림 7은 실험을 통해 측정된 쓰레드간 스케줄링 오버헤드를 나타내며, 80.5μsec의 스케줄링 시간을 보였다.

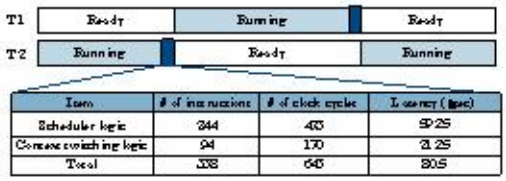


그림 7. 스케줄링 오버헤드
Fig. 7. Scheduling overhead

센서 운영체제의 커널, 플랫폼 추상화 계층, MCU 추상화 계층의 모듈이 갖는 프로그램 코드 사이즈를 그림 8에 나타내었다. 커널, 802.15.4 스택, 라디오 및 외부 플래쉬 메모리 드라이버, MCU 추상화 모듈이 탑재되었을 경우 프로그램 코드 사이즈는 약 *80 Kbyte* 이다.

커널의 동작에 요구되는 메모리 사이즈를 표 5에 나타내었다. 커널을 기본 값으로 설정하였을 경우 *439 byte*의 메모리가 요구된다. 그림 9는 커널, 플랫폼 추상화 계층, MCU 추상화 계층의 모듈에 요구되는 데이터 메모리 사이즈를 보여주며, 약 *2.2*

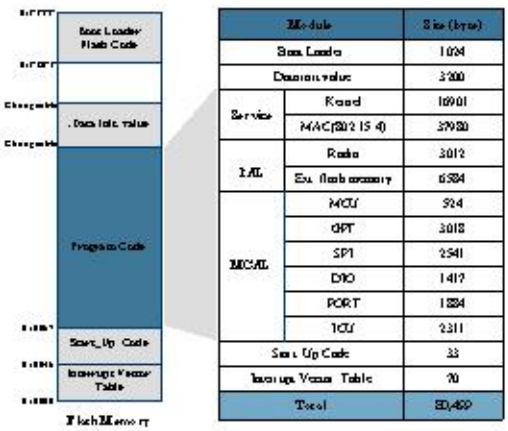
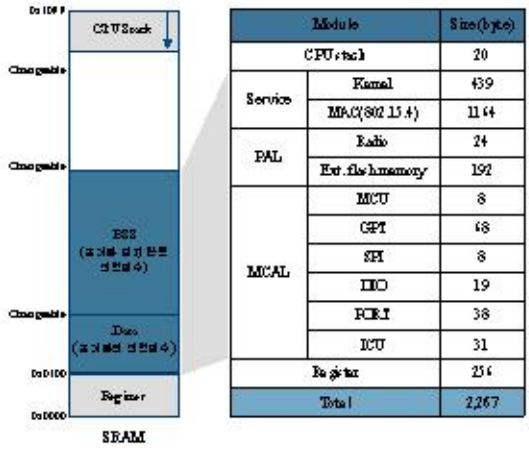


그림 8. 코드 사이즈
Fig. 8. Size of program code

표 5. 커널의 데이터 메모리 사용량
Table 5. Data memory for kernel

Value	Type		# of entity	Memory (bytes)	Description
	Name	bytes			
TcbBuf[]	TCB_t	15	6	90	- # of threads : 6
RcbBuf[]	SEM_t, MTX_t, or MB_t	5	6	18	- # of Semaphore : 2 - # of Mutex : 2 - # of MailBox : 2
MqBuf[]	MQ_t	5	2	10	- # of message queue : 2
WlBuf[]	WL_t	27	9	243	- # of ready wait list : 1 - # of semaphore WL : 2 - # of mutex WL : 2 - # of mailbox WL : 2 - # of message QWL : 2
TaBuf[]	TN_t	5	6	18	- # of threads
Global variable	UINT16	2	30	60	-
Total				439	-



Module		Size(byte)
CPU stack		20
Service	Kernel	439
	MAC(802.15.4)	1144
PAL	Radio	24
	Ext. flash memory	192
MCAL	MCU	8
	GPT	68
	SPI	8
	DIO	19
	PORT	38
	ICU	31
Register		256
Total		2,267

그림 9. 데이터 메모리 사이즈
Fig. 9. Size of data memory

*Kbyte*의 데이터 메모리가 요구된다. 본 논문에서 제안한 커널은 약 *17 Kbyte*의 코드 이미지와 *440 byte* 데이터의 메모리가 소요되었다.

센서 노드의 제한적인 하드웨어 자원으로 인해 기존의 센서 운영체제들은 코드 및 데이터 메모리 요구량이 하나의 성능척도로 사용되었다. 그러나 최근에 발표되는 운영체제들은 센서 하드웨어의 발전에 따라 운영체제의 다양한 기능의 제공 여부가 성능 척도로 사용되고 있다.

Ⅴ. 결론

무선 센서 네트워크는 소형 MCU, 무선 통신칩, 센서 등으로 구성된 저가의 센서 노드를 대량으로 배치하여 센싱 데이터를 수집하고 활용하는 기술이다. 최근에는 u-시티, u-농촌, u-헬스케어, u-국방, u-환경 등의 다양한 분야에 적용되어 그 중요성이 증가하고 하고 있다.

센서 노드에 탑재되는 센서 운영체제는 센싱 데이터의 획득, 처리, 전송 등을 지원하는 핵심 요소기술 이다. 기존의 센서 운영체제들은 센서 노드가 매우 제한적인 하드웨어 자원을 가지기 때문에 이러한 자원에 대한 효율적 사용에 추점을 두어 개발되어 왔다. 센서 운영체제는 응용분야의 특성에 따라 상이한 소프트웨어 및 하드웨어적 요구사항을 갖기 때문에 응용분야에 적합한 센서 운영체제를 개발하기 위해서는 많은 시간과 비용이 소요된다. 이러한 문제점을 개선하기 위해서는 센서 운영체제가 고도의 재구성 능력과 소프트웨어 모듈의 재사용 능력을 갖춰야한다.

본 논문에서는 재구성 가능한 고성능 센서 운영체제를 위한 소프트웨어 아키텍처를 제안하였다. 제안한 아키텍처는 수직적-수평적 구조의 계층적 소프트웨어 아키텍처 모델을 기반으로 설계되었다. 또한, 계층적으로 구조화된 센서운영체제 설계 모델을 기반으로 구현된 센서 운영체제는 다양한 응용분야에 쉽게 적용 가능하도록 운영체제의 재구성 능력과 소프트웨어 모듈의 재사용 능력이 뛰어난 특징을 갖는다. 특히, 제안한 센서 운영체제의 커널은 우선 순위기반 선점형 멀티쓰레딩, 다양한 사용자 API, 작은 코드 및 메모리 등의 특징을 갖는다.

향후 연구과제로는 구현된 센서 운영체제의 성능 개선을 위해 효율적인 전력관리, 위치인식, 원격 소프트웨어 업데이트, 동적 메모리, 경량 보안 서비스 등을 지원하는 것이다.

참고문헌

[1] E. Trumpler and R. Han, "A systematic framework for evolving TinyOS," in IEEE Workshop on Embedded Networked Sensors, pp. 61--65, May 2006.

[2] 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), Vol. 10, No. 4, pp. 563-579, Aug. 2005.

[3] C. C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes", Proceedings of the 3rd international conference on Mobile systems, applications, and services, pp. 163-176, 2005.

[4] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors", Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, pp. 455-462, 2004.

[5] J. J. Labrosse, "MicroC/OS-II The Real-Time Kernel 2/E", CMPBOOKS, Aug. 2003.

[6] AUTOSAR specification, http://www.autosar.org

[7] Sensor Network OS Research Team, "NanoQplus 2 (Nano OS) Specification", Electronics and Telecommunication Research Institute, pp. 1-76, Aug. 2007.

[B] H. Cha, S. Choi, I. Jung, H. Kim, H. Shin, J. Yoo, C. Yoon, "RETOS: Resilient, Expandable, and Threaded Operating System for Wireless Sensor Networks", The Sixth International Conference on Information Processing in Sensor Networks (IPSN 2007), pp. 148-157, Apr. 2007.

저 자 소 개

김 태 환(Tae-Hwan Kim)

2002년 2월 : 대구대학교 전자공학과 학사
2004년 2월 : 대구대학교 정보통신공학과 석사
2007년 8월 : 대구대학교 정보통신공학과 박사
2007년~현재 : 경북 유비쿼터스 신기술 연구센터 선임연구원

관심분야 : 센서 운영체제, 무선 센서네트워크, 임베디드 시스템

Email : tkim@utrc.re.kr

김 희 철(Hie-Cheol Kim)

1983년 2월 : 연세대학교 전자공학과 학사
1990년7월 : University of Southern California 컴퓨터공학 석사
1995년8월 : University of Southern California 컴퓨터공학 박사
1997년~현재 : 대구대학교 정보통신공학부 부교수

관심분야 : 센서 미들웨어, 운영체제, 무선 센서네트워크

Email : hckim@daegu.ac.kr