http://dx.doi.org/10.3745/KIPSTA.2011.18A.6.255

TinyOS에서의 선점적 EDF 스케줄링 알고리즘 설계 및 구현

유 종 선† · 김 병 곤†† · 최 병 규††† · 허 신††††

요 약

센서 네트워크는 빛, 소리, 온도, 움직임 같은 물리적 데이터를 센서 노드에서 감지하고 측정하여 중앙으로 전달하고 처리하는 구조를 가진 네트워크이다. 센서 네트워크는 여러 분야에서 활용할 수 있는 기술이다. 센서 노드가 외부에서 채취한 데이터를 실시간으로 사용자에게 전달하는 것은 매우 중요하다. 센서 네트워크의 핵심은 센서 노드인 하드웨어 플랫폼과 노드에 들어가는 초소형 운영체제라고 할 수 있다. UC 버클리에서 개발된 TinyOS는 센서 노드에서 동작하는 운영체제 중 가장 많이 사용되고 있다. TinyOS는 Event-driven 방식이며 Component 기반의 센서 네트워크 운영체제이다. 기본적으로 비선점 방식의 스케줄러를 사용한다. 만약 급한 작업이 수행되어야 하는 시점에서 다른 태스크가 수행 중에 있다면 수행 중인 태스크가 완료할 때까지 기다려야 한다. 이러한 특성으로 인해 TinyOS에서 정해진 시간안에 자신의 작업을 끝낸다고 보장하기 어렵다. 최근 연구에서 TinyOS의 빠른 반응성을 위해 Priority Level Scheduler라는 선점 기능이 제안되었다. 이것은 제한적으로 5개의 우선순위를 만들어 높은 우선순위가 낮은 우선순위를 선점할 수 있게 한다. 여기서 본 논문은 TinyOS의 실시간성을 보장함과 더불어 사용자 태스크의 평균 응답시간을 줄이고자 Priority Level Scheduler에 실시간 스케줄러인 EDF(Earliest Deadline First)를 적용한 선점형 EDF 스케줄링 방식을 제안하고자 한다.

키워드 : 센서 네트워크, TinyOS, EDF, Priority Level Scheduler

Design and Implementation of Preemptive EDF Scheduling Algorithm in TinyOS

Yoo Jong Sun† · Kim Byung Kon†† · Choi Byoung Kyu††† · Heu Shin††††

ABSTRACT

A sensor network is a special network that makes physical data sensed by sensor nodes and manages the data. The sensor network is a technology that can apply to many parts of field. It is very important to transmit the data to a user at real-time. The core of the sensor network is a sensor node and small operating system that works in the node. TinyOS developed by UC Berkeley is a sensor network operating system that used many parts of field. It is event-driven and component-based operating system. Basically, it uses non-preemptive scheduler. If an urgent task needs to be executed right away while another task is running, the urgent one must wait until another one is finished. Because of that property, it is hard to guarantee real-time requirement in TinyOS. According to recent study, Priority Level Scheduler ,which can let one task preempt another task, was proposed in order to have fast response in TinyOS. It has restrictively 5 priorities, so a higher priority task can preempt a lower priority task. Therefore, this paper suggests Preemptive EDF(Earliest Deadline First) Scheduler that guarantees a real-time requirement and reduces average respond time of user tasks in TinyOS.

Keywords : Sensor Network, TinyOS, EDF, Priority Level Scheduler

1. 서 론

센서 네트워크란 빛, 소리, 온도, 움직임 같은 물리적 데이터를 센서 노드에서 감지하고 측정하여 중앙으로 전달하고 처리하는 구조를 가진 네트워크이다[1]. 이러한 것은 군사, 과학 분야에서 널리 활용할 수 있는 신개념 기술이다. 사람이 접근하기 힘들거나 불가능한 지역에 센서 노드를 실

† 정 회 원 : 한국항공우주산업 항공ES팀 선임연구원
†† 정 회 원 : 한국건설기술연구원 건설정보연구실 수석연구원
††† 준 회 원 : 한양대학교 컴퓨터공학과 박사과정 수료
†††† 정 회 원 : 한양대학교 컴퓨터공학과 교수
논문접수 : 2011년 6월 15일
수 정 일 : 1차 2011년 9월 8일, 2차 2011년 9월 21일, 3차 2011년 9월 22일
심사완료 : 2011년 10월 23일

제 필드에 뿌려 놓아 주변 환경을 관찰하는데 사용할 수 있다. 최근 들어 센서 네트워크에 대한 연구가 많이 이루어지고 있다.

센서 네트워크의 핵심은 센서 노드인 하드웨어 플랫폼과 노드에 들어가는 초소형 운영체제라고 할 수 있다. 센서 노드는 센서 네트워크를 구성하는 하드웨어로 컴퓨터 시스템과 유사한 구조로 이루어졌지만, 극도로 제한된 자원을 가지고 있다. 예를 들어 센서노드 한 개에는 8bit MCU와 8~123 Kbyte의 플래시 메모리, 512Byte~4 Kbyte의 RAM으로 구성될 수 있다. 이와 같이 극심한 자원의 제약 때문에 센서 네트워크 운영체제의 크기가 작아질 수밖에 없지만, 사용자의 요구조건은 만족시킬 수 있는 기능도 있어야 한다.

센서 노드에 들어가는 운영체제 중에 UC버클리에서 개발된 초소형 센서 네트워크 운영체제인 TinyOS[2]가 있다. TinyOS는 크기가 4000 Byte 이하, 메모리 256 Byte 이하의 초소형 운영체제이다. 컴포넌트 기반의 구조로 이루어져 있으며 이벤트 발생에 의해 동작한다. 각각의 컴포넌트는 재사용이 가능하며, 이러한 컴포넌트들을 서로 연결함으로써 TinyOS를 구성한다. TinyOS는 C언어를 기반으로 만들어진 nesC언어[3]로 프로그래밍 된다.

TinyOS의 프로세스는 태스크와 이벤트로 나뉘며, 미룰 수 있는 계산 작업은 태스크로 사용한다. 태스크는 서로 다른 태스크에 의해 선점되지 않는다. 이러한 특성으로 인해 급한 작업이 수행되어야 하는 시점에서 다른 태스크가 수행 중에 있다면 수행 중인 태스크가 완료할 때까지 기다려야 한다. 이는 TinyOS의 반응성이 안 좋아질 뿐만 아니라 실시간성을 보장해주지 못할 수 있다.

최근 연구에서 TinyOS의 빠른 반응성을 위해 태스크 간에 선점할 수 있도록 Priority Level Scheduler[4]가 제안됐다. 이것은 기존에 한 개뿐인 태스크 큐를 5개로 늘려서 관리한다. 즉, 총 5개의 우선순위가 있으며 각 우선순위마다 따로 큐가 존재한다. 하지만 Priority Level Scheduler는 실행시 우선 순위를 정적으로 주어야하기 때문에 우선순위가 낮은 태스크는 자신의 작업을 정해진 시간내에 못 끝내는 상황이 발생할 수 있다.

본 논문에서는 TinyOS상에서 실시간성을 개선하기 위해서 Priority Level Scheduler상에 대표적인 실시간 스케줄링 알고리즘인 EDF(Earliest Deadline First)[5]를 사용하여 실시간성을 보장하는 선점형 EDF 스케줄링 기법을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구인 센서 네트워크, TinyOS, Priority Level Scheduler에 대하여 간략히 설명한다. 3장은 TinyOS에서 제공하는 EDF 스케줄링의 한계를 보이면서, 본 논문이 제안하는 선점형 EDF 스케줄러의 설계 및 구현에 대하여 설명한다. 4장에서는 선점형 EDF 스케줄러의 동작 실험 및 비선점형 EDF 스케줄러와의 성능 비교 실험을 보인다. 마지막으로 5장에서 본 논문의 결론 및 향후 과제에 대해 논의한다.

2. 관련 연구

2.1 센서 네트워크

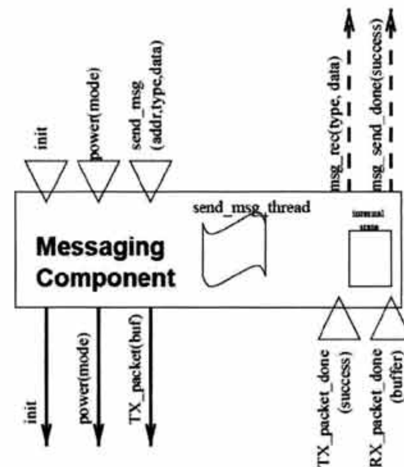
센서 네트워크란 기존 인간과 컴퓨터간의 커뮤니케이션에 사물과 물리적 대상을 추가시켜 협력 네트워크를 구성하고, 필요로 하는 모든 곳에서 센서 노드들을 통하여 자율적으로 정보를 수집, 관리 및 제어하는 시스템이다. 즉 물리 공간에 빛, 소리, 온도, 움직임 같은 물리적 데이터를 센서 노드에서 감지하고 측정하여 중앙으로 전달하고 처리하는 구조를 가진 네트워크이다[1].

2.2 TinyOS

UC Berkeley 대학에서 개발된 초소형 네트워크 임베디드 시스템 운영체제가 바로 TinyOS[2]이다.

TinyOS는 핵심 OS 코드가 4000바이트 이하, 데이터 메모리가 256바이트 이하의 초소형 운영체제이다. 매우 적은 메모리로 멀티태스킹을 지원하는 이벤트 기반 멀티태스킹 방식으로 동작하며, 이벤트 발생이 없는 시간 동안 CPU를 Sleep 모드로 전환함으로서 효율적인 CPU 사용을 통해 저전력을 실현한다.

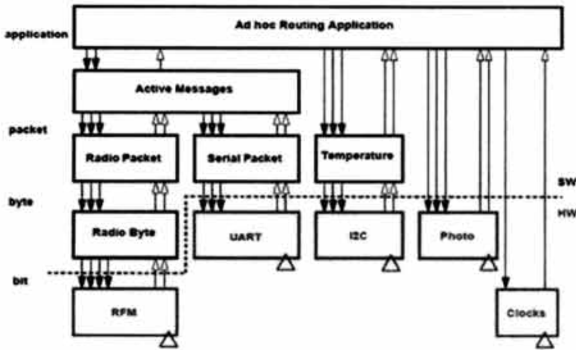
TinyOS는 컴포넌트 기반의 구조, 단순 이벤트 기반의 동시 모델과 함께 단계별로 분리되어 실행된다. 각 컴포넌트는 (그림 1)과 같이 이벤트/커맨드 구조로 하위 컴포넌트로부터 발생하는 이벤트를 상위 컴포넌트에서 이벤트 핸들러를 이용하여 이벤트에 대한 처리를 한다.



(그림 1) 컴포넌트의 예

TinyOS는 각각의 서비스를 재사용이 가능한 일련의 컴포넌트로 제공하며, 어플리케이션은 이러한 컴포넌트들 간의 연결로 구성된다. (그림 2)는 TinyOS에서 재사용 가능한 일련의 컴포넌트들을 독립적인 연결방법을 이용하여 구성한 것이다.

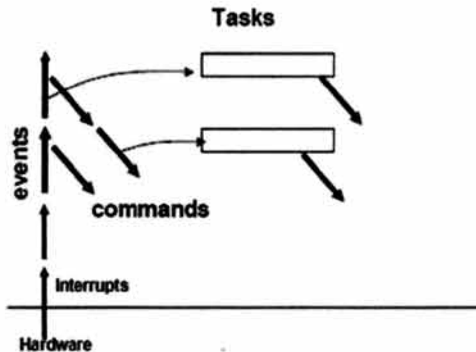
각각의 어플리케이션은 사용하고자 하는 일련의 컴포넌트를 주문형으로 만들며, 컴포넌트를 분해함으로서 사용되지 않는 서비스를 어플리케이션으로부터 제외할 수 있다.



(그림 2) 라우팅 어플리케이션의 예

TinyOS는 간단한 FIFO 구조를 가지는 소형 태스크 스케줄러와 재사용이 가능한 일련의 컴포넌트로 구성된다. 컴포넌트는 상위 컴포넌트로부터 내려온 요청을 수행하는 커맨드 핸들러, 하위 컴포넌트에서 올라온 이벤트를 처리하는 이벤트 핸들러, 고정된 메모리 영역인 프레임, 스케줄러에 의해 수행되는 태스크로 구성된다.

TinyOS의 프로세스들은 크게 태스크와 이벤트(하드웨어 인터럽트)로 나뉘며 스케줄링은 간편성을 위해 단순한 2 레벨 스케줄링 기법을 사용한다. 전체적인 TinyOS의 수행 방식은 (그림 3)과 같다. 태스크는 다른 태스크에 의해 선점되지 않지만, 이벤트에 의해서는 선점된다. 이벤트는 특정 하드웨어 인터럽트나 특정 조건을 만족했을 경우 호출되는 프로세스로서, 다른 태스크 보다 먼저 실행되는 특징이 있다.



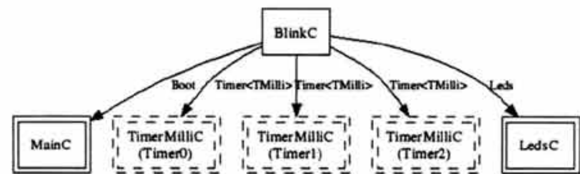
(그림 3) TinyOS의 수행 모델

2.3 nesC 언어

nesC[3]는 TinyOS의 실행 모델과 구조적인 개념을 표현하기 위하여 설계된 C언어의 확장된 언어이다.

nesC에서 어플리케이션은 정의된 컴포넌트와 양방향을 갖는 인터페이스로 만들어진다. 즉, 한 개 이상의 연결된 컴포넌트로 구성되는데, 컴포넌트는 인터페이스를 제공하고 이를 통하여 다른 컴포넌트와 연결된다. 인터페이스는 컴포넌트에 접근할 수 있는 양방향의 통로이며 인터페이스 제공자가 반드시 구현해야 하는 커맨드와 인터페이스 사용자가 구현해야 하는 이벤트라고 불리는 함수 집합을 정의한다. 하나의 컴포넌트는 여러 개의 인터페이스를 사용 및 제공할 수 있으며, 같은 인터페이스에 대해 여러 개의 인스턴스를 생성할 수 있다.

nesC에서는 모듈(module)과 컨피규레이션(configuration) 두 타입의 컴포넌트가 있다. 모듈(module)은 어플리케이션 코드를 제공하고 한 개 혹은 그 이상의 인터페이스를 구현한다. 컨피규레이션(configuration)은 특정 컴포넌트에 의해 사용된 인터페이스를 다른 컴포넌트에 의해 제공된 인터페이스에 연결하고 다른 컴포넌트를 함께 결합하는데 사용된다. 이러한 과정을 와이어링(wiring) 이라고 한다. (그림 4)는 Blink 어플리케이션의 와이어링(wiring) 모습을 보여준다.



(그림 4) Blink 어플리케이션의 wiring

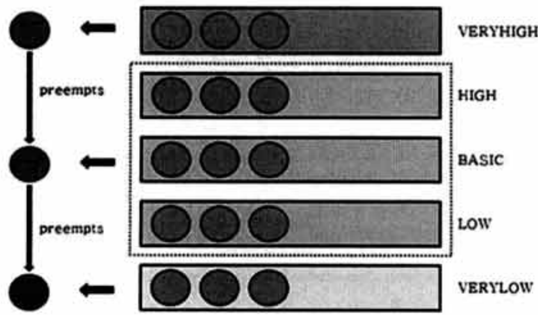
2.4 TinyOS에서 실시간성 제공 기법

TinyOS에서는 선입선출(FIFO : First-In First-Out) 방식의 비선점 방식을 이용한 태스크 스케줄링 기법을 사용하기 때문에 가장 높은 우선순위를 가진 태스크가 자신보다 낮은 우선순위의 태스크를 곧바로 선점할 수 없다. 따라서 실시간성을 보장 못 할 수 있을 뿐만아니라 평균 응답성도 커지게 된다. 이것을 보완하기 위해서 TinyOS에서 실시간성을 제공하기 위한 다양한 연구들이 제시되었다. 비선점형을 기반으로 실시간성을 제공하기 위한 스케줄링 기법은 우선순위 기반의 스케줄링 방식[6, 7]과 EDF기반을 통한 스케줄링 방식[8]이 있으며, 선점 기능을 기반으로 한 스케줄링 방식으로는 Priority Level Scheduler[4]라는 스케줄링 방식이 있다. 비선점형 스케줄링 방식은 완전하게 실시간성을 보장하기 힘들뿐만 아니라 평균 응답시간이 커지며, Priority Level Scheduler방식의 선점형 스케줄링은 비동기적으로 실행되는 태스크들에 대해 미리 지정된 정적 우선순위를 부여해야 하는 한계가 있다. 이외에 상황에 따라 우선순위를 정적또는 동적으로 줄 수 있는 two-layer priority[9] 방식이 있다. two-layer priority는 선점형 EDF에 대한 언급을 하였지만 자세한 구현에 대한 내용은 생략되었고, TinyOS에서 기본적으로 제공하는 FIFO와 비선점형 EDF 방식을 사용하여 시뮬레이션을 통해 결과를 비교하였다. 참고 논문 [10]은 TinyOS에서 응답성을 높이기 위한 방법으로 선점적인 EDF 방식을 이용하였다. 하지만 실시간성 관련 요소와 실시간성이 없는 요소를 하나의 정보로 처리함으로써 사용하지 않는 데이터의 불필요한 공간 낭비가 생길 수 있다.

2.5 Priority Level Scheduler

Cork 대학교의 Cormac Duffy가 처음으로 TinyOS에 Priority Level Scheduler[4]라는 선점 기능을 추가하였다. 이것은 일반 운영체제에서 사용되는 선점 기능하고 비교하

여 다소 독특한 구조를 가진다. 일단 TinyOS의 기본 스케줄러를 기반으로 하여 총 5개의 FIFO큐를 사용한다. VERYHIGH, HIGH, BASIC, LOW, VERYLOW로 총 5개의 우선순위가 있고, 각 우선순위마다 큐가 하나씩 존재한다.



(그림 5) Priority Level Scheduler의 구조

(그림 5)와 같이 큐가 이루어져 있으며, 파란색원은 태스크를 의미하고 각 태스크는 각 우선순위 큐에 저장되어 있다. 최상위에 있는 VERYHIGH 우선순위 태스크는 하위에 있는 모든 작업을 선점한다. 만약 하위 큐의 작업이 수행 중에 VERYHIGH의 태스크가 수행되려고 할 때 바로 선점하여 VERYHIGH 순위의 태스크가 수행된다. 중간에 있는 HIGH, BASIC, LOW 우선순위의 태스크는 기존의 TinyOS처럼 서로 선점할 수 없고 원자적으로 수행된다. 여기서 HIGH 우선순위의 큐가 먼저 수행되고, BASIC 우선순위, LOW 우선순위 순으로 수행되며, 최하위에 있는 VERYLOW 우선순위 태스크는 다른 순위에 의해 선점 당한다.

선점이 수행되면 현재 수행되는 태스크의 정보는 메모리 어딘가에 저장되고 Context Switch가 수행된다. push 함수를 사용하여 모든 레지스터 값을 메모리에 저장하고, pop 함수를 사용하여 메모리에 저장된 값을 레지스터로 복구한다. Priority Level Scheduler는 TinyOS의 반응성을 높이기 위해 추가된 선점형 스케줄링 알고리즘이지만, 우선순위를 미리 정적으로 주어야 하는 한계가 있다.

3. 선점형 EDF 스케줄러의 설계 및 구현

3.1 비선점형 EDF 스케줄러

3.1.1 TinyOS에서 제공하는 EDF 스케줄러

TinyOS의 기술문서인 TEP 106[7]은 TinyOS의 기본 스케줄러 및 태스크에 관련된 문서이다. TinyOS 2.x[11]부터 개발자가 기본 스케줄러를 다른 스케줄러로 바꿀 수 있는 메커니즘을 제공하며, TEP 106 문서는 기본 스케줄러의 동작 원리 및 스케줄러 변경 방법에 대해서 상세히 나와 있다. 이 문서에는 새로운 스케줄러에 대한 예시로 Tiny OS에 EDF 스케줄링[5]을 적용한 것이 있으나, 여기서 제시된 것은 비선점 스케줄링을 기본으로 사용됐다.

기본적인 알고리즘은 다음과 같다. 수행될 태스크가 deadline을 인자로 포스트 되면, 스케줄러가 태스크 식별자와 deadline을 확인한다. 스케줄러는 태스크 큐에 있는 태스크들의 deadline과 현재 들어온 태스크 deadline과 비교하여 deadline이 빠른 순서로 정렬 상태가 되도록 현재 들어온 태스크를 적절한 위치에 넣는다. 즉, 스케줄러가 관리하는 FIFO큐의 태스크를 deadline이 빠른 순서로 정렬시킴으로써 EDF 스케줄링을 구현하였다. 이렇게 되면 태스크 수행순서가 FIFO 이므로 deadline이 빠른 태스크가 먼저 큐에서 나와 수행된다.

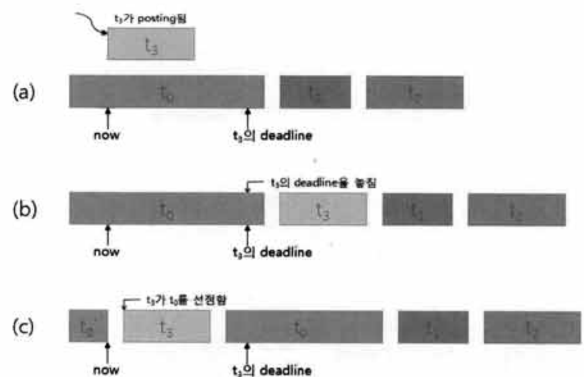
3.1.2 비선점형 EDF 스케줄러의 한계

TinyOS에서 제공하는 EDF 스케줄러는 TinyOS의 기본 개념인 비선점방식을 기반으로 구현된 것이다[6]. 이는 (그림 6)과 같은 상황에 처한 경우 비선점이기 때문에 deadline을 놓치는 일이 발생할 수 있다.

(그림 6)의 세부적인 설명은 다음과 같다. (a)에서 현재 태스크 t_0이 수행 중에 있고, 태스크 t_1, t_2가 차례로 태스크 큐에서 대기 중이다. 이 때 태스크 t_3이 포스트되지만, 태스크 t_3의 deadline은 t_0이 완료하기 전으로 되어있다. 결국 비선점 방식을 사용하는 EDF 스케줄링 방법은 (b)와 같이 될 수밖에 없다. t_3의 deadline이 가장 빨라 태스크 큐의 제일 앞쪽에 놓이게 되지만, t_0의 수행시간이 너무 길어서 t_3의 deadline을 놓치게 되어 실시간성 보장이 힘들 수 있다.

비선점 기반 EDF에서도 schedulability analysis를 통해 실시간성을 보장할 수 있지만, 시스템의 평균 응답성이 떨어지게 된다.

여기서 우리가 원하는 결과는 (c)와 같이 t_3이 t_0을 선점하여 t_3의 deadline을 충족시키고, t_0의 수행을 늦추므로써 실시간성 및 응답성을 만족시키고자 한다. 이에 본 논문은 (c)와 같이 선점할 수 있는 EDF 스케줄링을 제시하고자 한다.



(그림 6) 비선점 기반 EDF의 문제점

3.2 선점형 EDF 스케줄러의 설계

선점형 EDF는 Priority Level Scheduler[4]를 기본으로 사용하며, (그림 7)에서 Priority Level Scheduler는 5 개의 우선순위가 있다. 여기서 선점형 EDF에 맞춰서 모든 태스크의 우선순위를 (그림 7)과 같이 할당한다.

(그림 7) 태스크의 종류에 따른 우선순위 할당

우선 VERYHIGH 우선순위에 할당되는 작업은 매우 급한 태스크로써 현재 수행 중인 태스크보다 deadline이 더 빨라서 현재 태스크를 선점해야 할 때 사용한다. Priority Level Scheduler의 최고 우선순위를 사용함으로 현재 수행 중인 태스크의 정보를 메모리에 저장하고 deadline이 더 빠른 태스크를 먼저 수행시킨다.

두 번째로 HIGH 우선순위는 deadline이 있는 태스크에 할당된다. 이것은 선점형 EDF를 사용하는 모든 태스크는 HIGH에 할당되고, HIGH의 우선순위 큐는 deadline이 빠른 순서대로 정렬됨으로써 deadline이 있는 태스크를 일반 태스크보다 먼저 수행시켜 실시간성을 우선적으로 보장시킨다.

마지막으로 BASIC, LOW, VERY LOW 우선순위는 deadline이 없는 일반 태스크에 할당한다. 이렇게 해서 deadline이 있는 태스크에 좀 더 많은 수행 기회를 줌으로써 실시간성을 보장한다.

또한 VERYHIGH와 HIGH 우선순위를 구분시키는 방법은 다음과 같다.

일단 현재 수행 중인 태스크를 Tc(Current Task)라 하고, 지금 막 포스트된 태스크를 Tn(New Task)이라 하자. Tc는 현재 수행 중인 작업이므로 Tc의 deadline은 지금까지 포스트된 모든 태스크보다 빠르다. Tn이 포스트되면 deadline이 가장 빠른 Tc와 비교를 하는데, 이때 다음과 같이 두 가지 경우가 있다.

① Tn의 deadline이 Tc의 deadline 보다 같거나 늦다.
② Tn의 deadline이 Tc의 deadline 보다 빠르다.

여기서 Tc의 deadline은 Tc가 반드시 수행이 완료되어야 하는 시간이고, Tn의 deadline은 Tn이 반드시 완료되어야하는 시간이다. ①의 경우에서 Tc의 deadline이 Tn보다 빠를 때 Tn이 지금 당장 수행될 정도로 급한 것은 아니다. 이때 Tn은 HIGH 우선순위를 부여하고, HIGH 우선순위 큐 안에 있는 태스크들의 deadline과 비교하여 적절한 위치에 삽입한다. 삽입 후 HIGH 우선순위 큐는 deadline이 빠른 순으로 정렬되어 있어야 한다.

①의 경우에서 Tc와 Tn의 deadline이 같을 때 Tc가 그냥 수행될 수도 있고, Tn이 Tc를 선점하여 수행할 수 있다. 그러나 선점을 하는 것은 Context Switch 같은 오버헤드가 발생하므로 Tc가 그냥 수행되도록 하는 것이 더 효율적이므로 Tn은 HIGH 우선순위를 부여하고, HIGH 우선순위 큐에 삽입한다. 여기서 Tn은 deadline이 Tc의 deadline과 같으므로 HIGH 우선순위 큐 제일 앞쪽에 삽입하여 Tc 수행 후 Tn이 수행되도록 한다.

②의 경우는 Tn의 deadline이 더 빠른 경우로, Tn이 Tc보다 더 급한 작업이므로 선점이 필요하다. 우선 Tn은 VERYHIGH 우선순위를 부여한다. 그러면 Priority Level Scheduler가 Tc의 작업을 멈춰서 해당 태스크의 정보를 메모리에 저장한 후 Tn이 수행되도록 한다. Tn이 Tc를 선점함으로써 Tc의 수행 때문에 deadline을 놓치지 않게 되며, 결국 TinyOS의 반응성을 개선하는 효과가 있게 된다.

3.3 선점형 EDF 스케줄러의 구현

앞에서 언급한 것처럼 선점형 EDF 스케줄러는 Priority Level Scheduler를 기반으로 한다. 그러나 Priority Level Scheduler의 단점으로 우선순위를 컴파일 하기 전에 주어야 하는데 선점형 EDF 스케줄러는 매 순간의 deadline에 따라 우선순위를 판단해야 하기 때문에 Priority Level Scheduler를 바로 적용하기 어렵다. 그래서 본 논문은 Priority Level Scheduler에 실시간 스케줄러의 기능을 적용한 선점형 EDF 스케줄러를 제안하고자 한다.

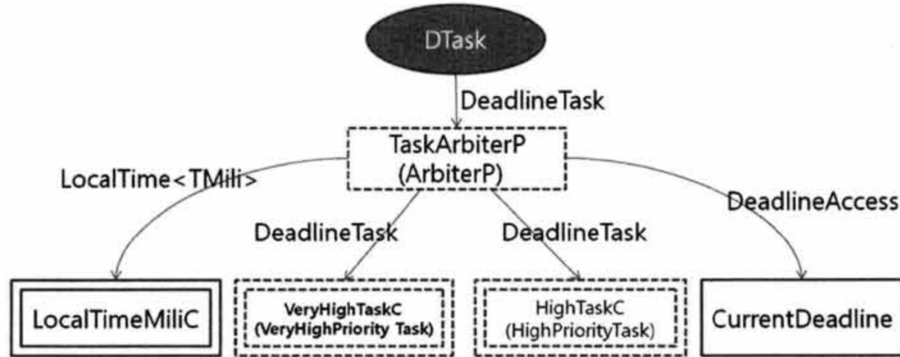
선점형 EDF 스케줄러는 크게 보면 (그림 8), (그림 9)와 같이 2개의 컴포넌트로 나눌 수 있다. (그림 9)에는 4개의 컴포넌트로 구성되어 있으며, 우측하단의 2개의 컴포넌트는 Priority Level Scheduler의 일부분이다.

(그림 8), (그림 9)는 TinyOS 2.x에서 사용하는 컴포넌트 다이어그램을 사용한다. 타원은 인터페이스를 나타내고, 한 줄로 된 사각형은 module 컴포넌트이며, 두 줄로 된 사각형은 configuration 컴포넌트이다. 또한 점선은 generic 컴포넌트를 나타내는데, generic 컴포넌트는 컴포넌트를 여러 개의 인스턴스로 생성할 수 있는 컴포넌트이다.

(그림 8) EDFTaskC의 구조

(그림 8)은 EDFTaskC 컴포넌트로 사용자에게 보이는 컴포넌트이며, 실제로 선점형 EDF 스케줄러를 사용하기 위해서는 EDFTaskC를 사용해야 된다. 응용프로그램 개발자는 최상위 컴포넌트인 EDFTaskC를 연결하여 선점형 EDF 스케줄러를 사용할 수 있다.

EDFTaskC가 제공하는 인터페이스는 DeadlineTask로 앞에서 언급한 선점형 EDF 태스크를 위한 인터페이스이다.



(그림 9) EDFTaskImplC의 구조

또한 EDFTaskC는 generic 컴포넌트로 여러 개의 인스턴스를 생성할 수 있게 한다. EDFTaskC에 연결된 EDF TaskImplC는 실질적인 구현 부분으로 제공하는 인터페이스는 EDFTaskC와 동일하다.

(그림 9)의 EDFTaskImplC는 4개의 컴포넌트로 구성되고, TaskArbiterP는 VeryHighTaskC, HighTaskC, Local TimeMilliC를 사용한다. LocalTimeMilliC는 현재 시간을 32 비트 정수로 구할 수 있는 컴포넌트이며, 시간의 단위는 밀리세컨드(ms)이다. 우측 하단의 2개의 컴포넌트는 Priority Level Scheduler의 일부분이고 VERYHIGH와 HIGH 우선순위 태스크에 해당한다. 여기서 총 우선순위가 5개인데 이 중 2개만 쓰는 이유는 선점형 EDF 스케줄러는 VERYHIGH와 HIGH만 사용하기 때문이다.

TaskArbiterP는 EDFTaskC 컴포넌트에서 온 post 커맨드를 받은 후 3.2절에 나온 정책에 따라 우선순위를 부여한 후 해당 우선순위의 컴포넌트로 post 커맨드를 보낸다. 만약 부여된 우선순위가 HIGH 태스크인 경우 HighTaskC 컴포넌트의 태스크 큐에 deadline 순서에 따라 삽입된다. 부여된 우선순위가 VERYHIGH 태스크인 경우 VeryHighTaskC 컴포넌트로 가서 현재 수행 중인 태스크를 선점하여 수행한다. 태스크 큐에 있다가 스케줄러에 의해 태스크가 수행될 차례가 오면 큐에서 event를 발생하여 DeadlineTask 인터페이스를 통해 사용자 응용프로그램까지 event가 도달하게 된다. event함수로 정의된 사용자 태스크는 event를 받고 태스크가 수행된다.

4. 실험 및 결과

4.1 실험 목적

본 실험은 선점형 EDF 스케줄러의 동작을 확인하고, TinyOS에서 제공하는 비선점형 EDF 스케줄러와 성능을 비교함으로써 선점형 EDF 스케줄러가 좀 더 많은 태스크의 deadline을 만족함을 보이는데 목적이 있다.

4.2 선점형 EDF 스케줄러의 동작 실험

본 실험은 구현된 선점형 EDF 스케줄러가 정상적으로 동작하는 것을 확인하는 실험이다. Kmote에 프로그램을 업로드 하여 콘솔 화면에서 확인을 하였다. 또한 호스트의 콘솔 화면에 출력하기 위해 TinyOS에서 제공한 printf 라이브러리를 응용프로그램에 추가시켰으며, 데드라인이 서로 다른 4개의 태스크를 같은 시간에 포스트하여 수행시켜 동작 여부를 실험하였다. 4개의 태스크 정보는 <표 1> 과 같다.

〈표 1〉 선점형 EDF 스케줄러 동작 실험의 태스크

태스크	포스트 시간	deadline	수행 시간
T1	1000	3100	500
T2	1000	2100	500
T3	1000	2600	500
T4	1000	1600	500

포스트 시간은 모든 태스크가 동일하게 1초 뒤에 수행되도록 하였다. 바로 시작하지 않고 1초 후 수행 하는 이유는 부팅 후 남아 있을 수 있는 다른 태스크가 방해되지 않도록 하기 위해서이다. 포스트되는 순서는 1초 후 T1, T2, T3, T4순으로 하였고, 데드라인을 서로 다르게 하였다. 여기서 EDF 스케줄링을 적용할 때 우리가 예상하는 결과로는 T4, T2, T3, T1 순서대로 수행하는 것이다. <표 1>에 있는 데드라인은 사용자가 인자 값으로 주는 데드라인이 아닌 실제 완료가 되어야하는 시간을 나타내는데, 사용자가 인자 값으로 주는 데드라인과 포스트 시간을 합한 값이다. 각 태스크마다 수행하는 작업은 0.5초 동안 의미 없는 루프작업을 수행하며, 결과는 (그림 10)과 같다.

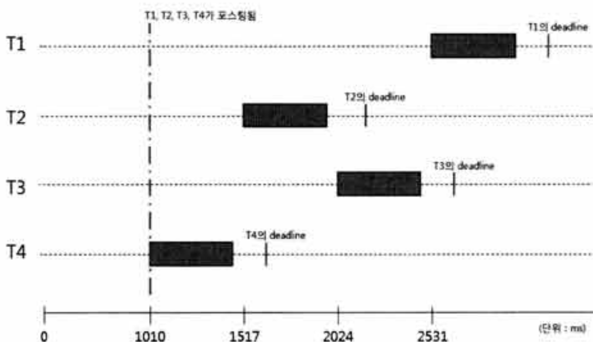
이 응용프로그램은 먼저 수행이 완료된 태스크부터 먼저 출력을 하도록 했다. 따라서 수행이 완료된 태스크의 순서는 T4, T2, T3, T1으로 예상대로 동작하였다. (그림 10)에 출력된 정보를 보면 Post는 포스트 시간을 나타내고, Deadline은 데드라인을 나타내고, Start는 그 태스크가 실제 수행을 시작하는 시간이고, Finish는 태스크가 수행을 마친 시간을 나타낸다. T1이 포스트를 제일 먼저 하고, T4가 제일 나중에 하였지만, 수행 순서는 반대가 되었다. 이

것은 포스트 시간에 상관없이 데드라인이 빠른 순서대로 수행됐음을 알 수 있다. 이 실험을 보기 쉽게 도식화하면 (그림 11)과 같다.

```
ncc -o build/telosb/main.exe -Os -O -mdisable-hwmul -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-al
l -target=telosb -fnesc-cfile=build/telosb/app.c -board= -I/opt/tinyos-2.x/tos/lib/printf -DIDENT_US
ER_ID=\"ujong3\" -DIDENT_HOSTNAME=\"ujong3-desktop\" -DIDENT_USER_HASH=0xf16be45aL -DIDENT_UNIX_TIME
=0x485e1e25L -DIDENT_UID_HASH=0x43b22effL -I%T/lib/PreemptivePriorityScheduler/Priorities -I%T/lib/P
reemptivePriorityScheduler  ExampleApp.nc -lm
    compiled ExampleApp to build/telosb/main.exe
           13168 bytes in ROM
             782 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
ujong3@ujong3-desktop:/opt/tinyos-2.x/apps/PriorityTestApps/DeadlineTest2$ make telosb reinstall bsl
,/dev/ttyUSB0
cp build/telosb/main.ihex build/telosb/main.ihex.out
    installing telosb binary using bsl
tos-bsl --telosb -c /dev/ttyUSB0 -r -e -I -p build/telosb/main.ihex.out
MSP430 Bootstrap Loader Version: 1.39-telos-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
13200 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out build/telosb/main.ihex.out
ujong3@ujong3-desktop:/opt/tinyos-2.x/apps/PriorityTestApps/DeadlineTest2$ java PrintfClient -comm s
erial@/dev/ttyUSB0:telosb
Thread[Thread-1,5,main]serial@/dev/ttyUSB0:115200: resynchronising
T4 => Post:1010,Deadline:1610,Start:1010,Finish:1510 => OK!!
T2 => Post:1010,Deadline:2110,Start:1517,Finish:2017 => OK!!
T3 => Post:1010,Deadline:2610,Start:2024,Finish:2523 => OK!!
T1 => Post:1010,Deadline:3110,Start:2531,Finish:3030 => OK!!
```

센서 노드에서 호스트 콘솔로 보내진 출력

(그림 10) 선점형 EDF 스케줄러 동작 실험 수행 결과



(그림 11) 선점형 EDF 스케줄러 수행 동작 실험 도식화

(그림 11)과 같이 모든 태스크가 1010ms 에 포스트 됐지만, 수행 순서는 데드라인이 빠른 순서로 실행됐음을 알 수 있다. 그러므로 이 실험을 통해 선점형 EDF 스케줄러가 동작하는 것을 확인 할 수 있다.

4.3 비선점형 EDF 스케줄러와 비교 실험

4.2절의 실험만 가지고는 선점형 EDF 스케줄러가 완벽하게 동작한다는 것을 보장할 수 없다. 왜냐하면 데드라인 순서대로 수행되긴 하였지만 선점이 수행되지 않는 상황이기 때문에 선점 기능에 대한 검증이 부족한 실험이었다. 이번 절의 실험은 선점 기능을 사용하는 실험으로, 선점이 발생하는 상황은 흔하게 되지 않기 때문에, 본 실험에서는 특수한 상황을 만들어 수행한다. 특수한 상황은 3.1.2 절에서 언급한 비선점일 경우의 문제가 발생하는 경우를 바탕으로 하였다. 또한, 동일한 상황 하에 비선점형 EDF 스케줄러와 비교 실험하여 선점형 EDF 스케줄러가 더 우수하다는 것을 검증하고자 한다.

수행할 태스크는 4개이고, 각 태스크에 대한 정보는 <표 2>와 같다.

이번 실험은 T1, T2, T4 가 같은 시간에 포스트되며, 데드라인의 빠르기 역시 T1, T2, T4 순서대로 빠르다.

태스크 T1이 제일 먼저 수행될 것이라는 것을 예측할 수 있다. 특이한 점은 T1의 수행 시간이 2000ms로 상당히 길게 하였는데, 먼저 수행되면서 수행 시간이 길면 다른 태스

〈표 2〉 선점형 EDF 스케줄러 동작 실험의 태스크

태스크	포스트 시간	deadline	수행 시간
T1	1000	4000	2000
T2	1000	4500	500
T3	1500	2500	500
T4	1000	5500	1000

크에 영향을 미치게 될 것이다. 다음으로 1500ms 에 T3이 포스트된다. 그러나 T3의 데드라인을 보면 다른 태스크보다 더 빠르다는 것을 알 수 있다. 이 상황은 3.1.2절의 예와 비슷한 경우라고 생각할 수 있는데, 이런 경우 비선점형 EDF 스케줄러와 선점형 EDF 스케줄러가 각각 어떻게 동작되는지 실험을 통해 알아보자.

4.3.1 비선점형 EDF 스케줄러를 사용한 경우

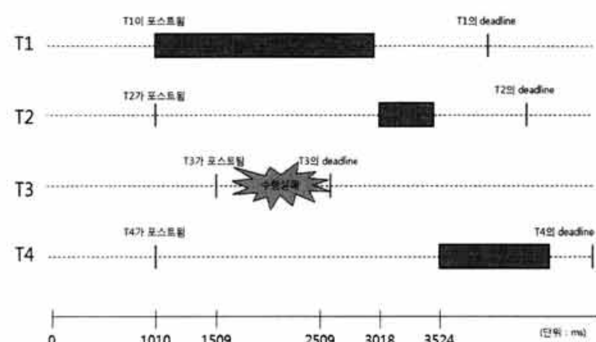
우선 실험 결과는 (그림 12)와 같이 수행되었다.

```
ncc -o build/telosb/main.exe -Os -O -mdisable-hwmul -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-al
l -target=telosb -fnesc-cfile=build/telosb/app.c -board= -I/opt/tinyos-2.x/tos/lib/printf -DIDENT_US
ER_ID=\"ujong3\" -DIDENT_HOSTNAME=\"ujong3-desktop\" -DIDENT_USER_HASH=0xf16be45aL -DIDENT_UNIX_TIME
=0x485dccd9L -DIDENT_UID_HASH=0x7143e4e6L -I%T/lib/PreemptivePriorityScheduler/Priorities -I%T/lib/P
reemptivePriorityScheduler  ExampleApp.nc -lm
    compiled ExampleApp to build/telosb/main.exe
           12760 bytes in ROM
             786 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
ujong3@ujong3-desktop:/opt/tinyos-2.x/apps/PriorityTestApps/DeadlineTest2$ make telosb reinstall bsl
,/dev/ttyUSB0
cp build/telosb/main.ihex build/telosb/main.ihex.out
    installing telosb binary using bsl
tos-bsl --telosb -c /dev/ttyUSB0 -r -e -I -p build/telosb/main.ihex.out
MSP430 Bootstrap Loader Version: 1.39-telos-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
12792 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out build/telosb/main.ihex.out
ujong3@ujong3-desktop:/opt/tinyos-2.x/apps/PriorityTestApps/DeadlineTest2$ java PrintfClient -comm s
erial@/dev/ttyUSB0:telosb
Thread[Thread-1,5,main]serial@/dev/ttyUSB0:115200: resynchronising
T1 => Post:1010,Deadline:4010,Start:1010,Finish:3006 => OK!!
T3 => Post:1509,Deadline:2509,Start:0,Finish:0 => NO~~
T2 => Post:1010,Deadline:4510,Start:3018,Finish:3517 => OK!!
T4 => Post:1010,Deadline:5510,Start:3524,Finish:4522 => OK!!
```

deadline을 만족하지 못했음

(그림 12) 비선점형 EDF 스케줄러를 사용한 실험 1

비선점형 EDF 스케줄러를 사용하면 (그림 12)와 같이 태스크 T3은 데드라인을 만족하지 못함을 알 수 있다. T1의 시작시간은 1010ms, 완료시간은 3006ms 로 원래 수행시간인 2000ms 동안 선점 없이 수행됐다. T3은 T1이 수행 중에 포스트가 되었지만 T1이 마칠 때까지 수행되어 T3은 데드라인을 놓쳤다. 결국 4개의 태스크 중 3개만이 데드라인을 만족했음을 알 수 있다. 이 과정을 도식화하면 (그림 13)과 같다.



(그림 13) 비선점형 EDF 스케줄러를 사용한 경우의 도식화

(그림 13)과 같이 T3은 T1이 수행 중에 포스트 됐지만 T3의 데드라인 역시 T1이 수행하는 가운데 있다. 결국 T1이 계속 수행함으로 인해 T3은 수행되기도 전에 데드라인을 놓치는 결과가 발생한다. 이는 3.1.2절에서 지적한 것 같이 비선점을 사용하였기 때문에 위와 같은 실험 결과가 나왔다. 다음은 본 논문이 제안하는 선점 방식을 사용할 경우 결과가 어떻게 나오는지 확인한다.

4.3.2 선점형 EDF 스케줄러를 사용한 경우

위 실험과 동일한 조건 하에 선점 방식을 사용할 경우 실험 결과는 (그림 14)와 같다.

```
ncc -o build/telosb/main.exe -Os -O -mdisable-hwmul -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-al
l -target=telosb -fnesc-cfile=build/telosb/app.c -board= -I/opt/tinyos-2.x/tos/lib/printf -DIDENT_US
ER_ID=\"ujong3\" -DIDENT_HOSTNAME=\"ujong3-desktop\" -DIDENT_USER_HASH=0xf16be45aL -DIDENT_UNIX_TIME
=0x485dcd83L -DIDENT_UID_HASH=0x6dfd5b36L -I%T/lib/PreemptivePriorityScheduler/Priorities -I%T/lib/P
reemptivePriorityScheduler  ExampleApp.nc -lm
    compiled ExampleApp to build/telosb/main.exe
           13172 bytes in ROM
             782 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
ujong3@ujong3-desktop:/opt/tinyos-2.x/apps/PriorityTestApps/DeadlineTest2$ make telosb reinstall bsl
,/dev/ttyUSB0
cp build/telosb/main.ihex build/telosb/main.ihex.out
    installing telosb binary using bsl
tos-bsl --telosb -c /dev/ttyUSB0 -r -e -I -p build/telosb/main.ihex.out
MSP430 Bootstrap Loader Version: 1.39-telos-8
Mass Erase...
Transmit default password ...
Invoking BSL...
Transmit default password ...
Current bootstrap loader version: 1.61 (Device ID: f16c)
Changing baudrate to 38400 ...
Program ...
13204 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out build/telosb/main.ihex.out
ujong3@ujong3-desktop:/opt/tinyos-2.x/apps/PriorityTestApps/DeadlineTest2$ java PrintfClient -comm s
erial@/dev/ttyUSB0:telosb
Thread[Thread-1,5,main]serial@/dev/ttyUSB0:115200: resynchronising
T3 => Post:1509,Deadline:2509,Start:1509,Finish:2008 => OK!!
T1 => Post:1010,Deadline:4010,Start:1010,Finish:3512 => OK!!
T2 => Post:1010,Deadline:4510,Start:3519,Finish:4018 => OK!!
T4 => Post:1010,Deadline:5510,Start:4025,Finish:5023 => OK!!
```

deadline을 모두 만족시켰음

(그림 14) 선점형 EDF 스케줄러를 사용한 실험 1

선점 방식을 사용하게 되면 (그림 14)와 같이 모든 태스크가 데드라인을 만족함을 보여준다. 태스크 T3의 포스트 시간은 1509ms 이고, 수행 시작 시간 역시 1509ms로 포스트되자마자 바로 선점하여 수행하였다. 또한 T1의 수행 시작 시간은 1010ms 이고, 수행 완료 시간은 3517ms 로 완료하기까지 총 2507ms 가 걸렸다. T1의 원래 수행 시간은 2000ms 로 500ms 정도는 T3가 수행했음을 알 수 있다. T2는 3519ms 에서 수행을 시작한 후, 500ms 동안 실행된다. T2의 수행이 완료된 후 6ms의 시간 지연이 생기는데, 일반적으로 context switching시간이 2~3ms인 것을 감안하면 높게 나왔다. 이것은 log찍는 시간과 태스크 종료 후 처리작업으로 인한 오차범위이다. 태스크의 완료된 순서는 데드라인이 빠른 순서인 T3, T1, T2, T4로 EDF 알고리즘을 충분히 수행했다. 이 과정을 도식화 하면 (그림 15)와 같다.

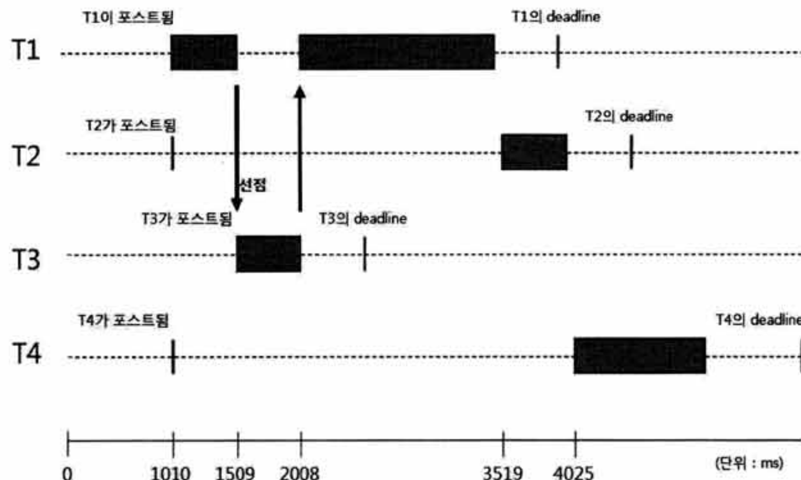
(그림 15)와 같이 T1이 수행 중에 데드라인이 T1 보다 더 빠른 T3이 포스트되면 바로 T1은 수행을 중단하고 문맥교환(Context Switch)를 실행한다. T3이 바로 수행을 시작하고 작업을 마친다. 그러고 나서 중단되었던 T1을 중단 시점부터 다시 시작한다. 그 결과 4개의 태스크 중에서 모든 태스크의 데드라인을 만족함을 이 실험을 통해서 알 수 있다.

위 두 실험은 동일한 조건과 상황에서 수행되었다.

비선점 방식은 모든 태스크의 데드라인을 만족 시키지 못했지만, 선점 방식은 모든 태스크에서 데드라인을 만족함을 보였다. 이는 본 논문인 제안하고 있는 선점형 EDF 스케줄러가 올바르게 동작함을 보였을 뿐만 아니라 비선점형 EDF 스케줄러보다 우수함을 실험을 통해 알 수 있다.

4.4 선점형 EDF 스케줄러의 성능 평가 및 분석

센서 네트워크에서 센서들은 실제 물리 데이터를 가져오기 때문에 실시간성이 매우 중요한 이슈일 것이다. 그러나 센서 운영체제인 TinyOS는 실시간성을 보장할 수 없다는 문제가 있다. 이에 본 논문은 선점이 가능한 EDF 스케줄러를 제안했다. 기존 TinyOS는 실시간성을 보장하지 못하기 때문에 본 논문이 제안하는 것과 비교하는 것은 무리가 있다. 비공식적으로 TinyOS에 EDF 알고리즘을 적용한 예가 있다. 이것은 TinyOS의 기본 성질인 비선점방식을 그대로 적용하였기 때문에 실시간성을 보장하지 못할 수 있을 뿐만 아니라 평균 응답성도 커질 수 있다. 이에 본 논문은 선점방식하에 EDF 스케줄링 알고리즘을 적용하였고, 비선점방식보다 우수하다는 것을 실험을 통해서 확인하였다.



(그림 15) 선점형 EDF 스케줄러를 사용한 경우의 도식화

5. 결론 및 향후 과제

센서 네트워크는 빛, 소리, 온도, 움직임 같은 물리적 데이터를 센서 노드에서 감지하고 측정하여 중앙으로 전달하고 처리하는 구조를 가진 네트워크이다. UC 버클리에서 개발된 TinyOS는 센서 노드에서 동작하는 운영체제 중 가장 많이 사용되고 있다.

기본적으로 비선점 방식의 스케줄러를 사용함으로써 TinyOS에서는 모든 태스크들이 정해진 시간 안에 끝낼 수 있다고 보장하기 힘들다. 또한, 비선점 기반의 EDF 스케줄링은 매우 급한 태스크가 들어와도 현재 수행되는 태스크가 완료될 때까지 기다려야하기 때문에 자신의 데드라인을 넘기는 문제가 있다는 것을 예제 및 실험을 통해 확인하였다.

이에 본 논문은 TinyOS의 실시간성 및 응답성을 강화하기 위하여 Priority Level Scheduler를 기반으로 선점형 EDF 스케줄러를 제안하였고 수행 중에도 우선순위를 할당할 수 있도록 개선하였다.

본 논문에서 고려할 부분은 Priority Level Scheduler는 일반 OS에서 동작하는 선점과는 다르게 매우 제약적으로 동작하기 때문에 선점형 EDF 스케줄러 역시 제약적으로 선점기능을 사용하고 있다. 따라서 향후에는 제약적인 선점기능을 향상시키면서 TinyOS의 실시간성 보장을 위한 연구가 필요하다.

참 고 문 헌

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

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. "System architecture directions for networked sensors," In Proc. of the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.93 - 104, Cambridge, MA, Nov., 2000.

[3] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. "The nesC language: A holistic approach to networked embedded systems," In proc. ACM SIGPLAN Conference on Programming Language Design and Implementation, pp.1-11, June, 2003.

[4] Cormac Duffy, Utz Roedig, John Herbert and Cormac J. Sreenan. "Adding preemption to tinyos," In To appear in the Fourth Workshop on Embedded Networked Sensors (EmNets 2007), University College Cork, Ireland. ACM Digital Library, June, 2007.

[5] C.L. Liu and James W. LayLand. "Scheduling Real-Time Environment," J.ACM, 20, pp.40-61, 1973.

[6] V.Subramonian, H-M. Huang, S. Data, and C. Lu, "Priority scheduling in TinyOS - A case study," Technical Report WUCSE-2003-74, Washington University, 2002.

[7] Tao Lei, Xiang-mo Zhao and Fei hui. "A TinyOS scheduling strategy and its implementation," Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on, pp.216-219, May, 2011.

[8] P.Levis and C.Sharp. "Schedulers and Tasks," TinyOS 2.x Extension Proposal 106.

[9] Zhi-bin Zhao, Fuxiang Gao, "Study on Preemptive Real-Time Scheduling Strategy for Wireless Sensor Networks," Journal of information processing systems 5(3), pp.135-144, September, 2009.

[10] Liu Jianhui, "Research on TinyOS Scheduling Strategy Based on SJP," Computational Intelligence and Design (ISCID), 2010 International Symposium on, pp.144-146, Oct., 2010.

[11] P. Levis, D. Gay, V. Handziski, J.-H. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler, and A. Wolisz. "T2: A second generation os for embedded sensor networks." Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universitat Berlin, 2005.

유 종 선

e-mail : ujong3@hanmail.net
2007년 한양대학교 전자컴퓨터공학부(학사)
2009년 한양대학교 컴퓨터공학과(석사)
현 재 한국항공우주산업 항공ES팀 선임연구원
관심분야 : 운영체제, 무선 센서네트워크 (WSN), 항공 OFP

김 병 곤

e-mail : bkkim@kict.re.kr
1991년 한양대학교 전자계산학과(학사)
1993년 한양대학교 전자계산학과(석사)
2003년 한양대학교 컴퓨터공학과 박사과정 수료
1993년~현 재 한국건설기술연구원 건설정보연구실 수석연구원
관심분야 : 운영체제, 무선 센서네트워크(WSN), 건설 정보화

최 병 규

e-mail : mysaint@hanyang.ac.kr
2002년 한양대학교 전자컴퓨터공학부(학사)
2004년 한양대학교 컴퓨터공학과(석사)
현 재 한양대학교 컴퓨터공학과 박사과정 수료
관심분야 : 무선 센서네트워크(WSN), TMO(Time-triggered Message-triggered Object)

허 신

e-mail : shinheu@hanyang.ac.kr

1973년 서울대학교 전기공학과(학사)
1979년 미국 University of Southern California 전산학(석사)
1986년 미국 University of South Florida 전산학(박사)
1980년~1986년 미국 University of South Florida 연구원보
1986년~1988년 미국 The Catholic University of America 조교수
1988년~현 재 한양대학교 컴퓨터공학과 교수
관심분야 : 분산컴퓨팅, 결함허용시스템, 실시간운영체제 등