TinyOS에 선점형 EDF 스케줄링 적용

유종선○ 허신
한양대학교 컴퓨터공학과
ujong3@hanyang.ac.kr○, shinheu@hanyang.ac.kr

Applying Preemptive EDF Scheduling to TinyOS

Jong-Sun Yoo○ Shin Heu
Dept of Computer Science & Engineering, Han-Yang University

요 약

센서 네트워크는 여러 분야에서 활용할 수 있는 기술이다. 센서 노드가 외부에서 채취한 데이터를 실시간으로 사용자에게 전달하는 것은 매우 중요하다. 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 사용함으로써 TinyOS의 실시간성을 보장하기 어렵다. 최근 연구에서 TinyOS의 빠른 반응성을 위해 Priority Level Scheduler라는 선점 기능이 제안되었다. 여기서 본 논문은 TinyOS의 실시간성의 보장을 위해 Priority Level Scheduler에 EDF(Earliest Deadline First)를 적용한 선점형 EDF 스케줄링 방식을 제안하고자 한다.

1. 서 론

센서 네트워크란 빛, 소리, 온도, 움직임 같은 물리적 데이터를 센서 노드에서 감지하고 측정하여 중앙으로 전달하고 처리하는 구조를 가진 네트워크이다. 이러한 것은 군사, 과학 분야에서 널리 활용할 수 있는 신개념 기술이다. 사람이 접근하기 힘들거나 불가능한 지역에 센서 노드를 실제 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, 512 Byte ~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 중 대표적인 것이 UC 버클리에서 개발된 초소형 센서 네트워크 운영체제인 TinyOS [1]가 있다. TinyOS는 크기가 4000 Byte 이하, 메모리 256 Byte 이하의 초소형 운영체제이다. 컴포넌트 기반의 구조로 이루어 졌으며 이벤트 발생에 의해 동작한다. 각각의 컴포넌트는 재사용이 가능하며, 이러한 컴포넌트들을 서로 연결함으로써 TinyOS를 구성한다. TinyOS는 C언어를 기반으로 만들어진 nesC언어 [2]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 전혀 보장해 주지 못한다.

최근 연구에서 TinyOS의 빠른 반응성을 위해 태스크 간에 선점할 수 있도록 Priority Level Scheduler [3]가 제안됐다. 이것은 기존에 한 개뿐인 태스크 큐를 5개로 늘려서 관리한다. 즉, 총 5개의 우선순위가 있으며 각 우선순위마다 따로 큐가 존재한다.

본 논문은 Priority Level Scheduler의 선점 기능을 이용하여 실시간성을 보장하는데 도움이 되는 스케줄러를 보인다. 다시 말하면, TinyOS에 대표적인 실시간 스케줄링 알고리즘인 EDF(Earliest Deadline First) [4]와 Priority Level Scheduler를 사용한 선점형 EDF 스케줄링 기법을 제시한다.

본 논문의 구성은 다음과 같다. 2절에서 관련 연구를 보이고, 3절에서 TinyOS에서 제공하는 EDF 스케줄링의 한계를 보인다. 4절에서는 본 논문이 제안하는 선점형 EDF 스케줄링을 보이고, 마지막으로 5절에서 본 논문의 결론을 보인다.

2. 관련 연구

2.1. TinyOS

UC 버클리에서 개발된 TinyOS는 이벤트 발생에 의한 상태 전이 방식의 개념을 사용한 운영체제이다. 동시적인 프로세싱 및 제한된 하드웨어 메모리 공간에서의 효율적인 성능을 지원해준다. 상태머신 기반의 구조를 가지며 응용프로그램은 각 독립적인 컴포넌트를 연결하는

방식으로 이루어진다.

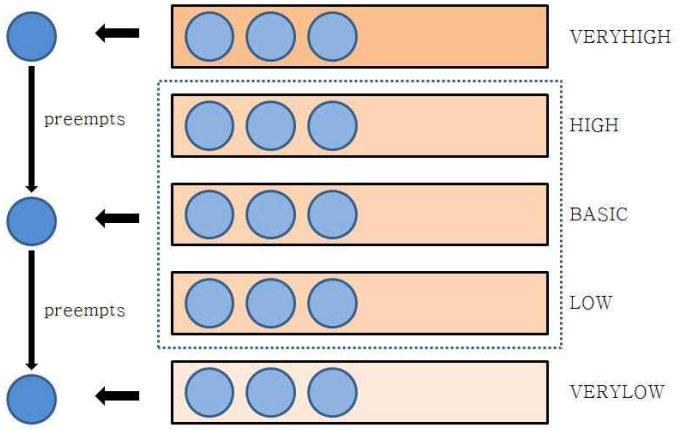
TinyOS는 간단한 FIFO구조의 태스크 스케줄러와 재사용이 가능한 컴포넌트들로 구성된다. 컴포넌트는 상위 컴포넌트로부터 내려온 요청을 수행하는 커맨드 핸들러, 하위 컴포넌트에서 올라온 이벤트를 처리하는 이벤트 핸들러, 고정된 메모리 영역인 프레임, 스케줄러에 의해 수행되는 태스크로 구성된다.

TinyOS의 프로세스는 태스크와 이벤트로 나뉘며 태스크는 다른 태스크에 의해 선점되지 않지만, 이벤트에 의해서는 선점된다. 이벤트는 하드웨어 인터럽트나 특정 조건에서 호출되는 프로세스로 태스크를 선점할 수 있다.

2.2. Priority Level Scheduler

Cork 대학교의 Cormac Duffy가 처음으로 TinyOS에 Priority Level Scheduler라는 선점 기능을 추가하였다. 이것은 일반 운영체제에서 사용되는 선점 기능하고 비교하여 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 1) Priority Level Scheduler의 큐 구조

(그림 1)과 같이 큐가 이루어져 있다. (그림 1)에서 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 수행된다. 최하위에 있는 VERYLOW 우선순위 태스크는 다른 순위에 의해 선점 당한다.

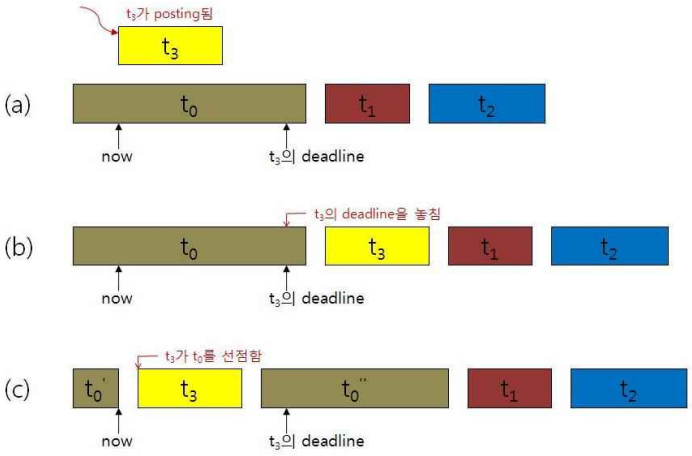
선점이 수행되면 현재 수행되는 태스크의 정보는 메모리 어딘가에 저장되고 Context Switch가 수행된다. *push* 함수를 사용하여 모든 레지스터 값을 메모리에 저장하고, *pop* 함수를 사용하여 메모리에 저장된 값을 레지스터로 복구한다.

Priority Level Scheduler는 TinyOS의 반응성을 높이기 위해 추가된 선점형 스케줄링 알고리즘이지만, 우선순위를 미리 정적으로 주어야 하는 한계가 있다.

3. TinyOS에서 제공하는 EDF 스케줄링의 한계

TinyOS의 기술문서인 TEP 106 [5]은 TinyOS의 기본 스케줄러 및 태스크에 관련된 문서이다. 이 문서에는 Tiny OS에 EDF 스케줄링을 적용한 것이 있다. 그러나 여기서 제시된 것은 비선점 스케줄링을 기본으로 사용됐기 때문에 문제가 된다.

기본적인 알고리즘은 다음과 같다. 수행될 태스크가 deadline을 인자로 포스팅 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 태스크가 deadline이 빠른 순서로 정렬시킴으로써 EDF 스케줄링을 구현하였다. 이렇게 되면 태스크 수행순서가 FIFO 이므로 deadline이 빠른 태스크가 먼저 큐에서 나와 수행된다. 수행 알고리즘을 보면 EDF 스케줄링이 제대로 동작할 것처럼 보인다.



(그림 2) 비선점 기반 EDF의 문제점

그러나 비선점 스케줄링이 기본이다 보니 (그림 2)와 같은 문제가 발생할 수 있다. (그림 2)의 (a)에서 현재 태스크 t_0가 수행 중에 있고, 태스크 t_1, t_2가 차례로 태스크 큐에서 대기 중이다. 이 때 태스크 t_3가 포스팅 된다. 그러나 태스크 t_3의 deadline은 t_0가 완료하기 전으로 되어있다. 결국 비선점 방식을 사용하는 EDF 스케줄링 방법은 (그림 2)의 (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 된다. 이렇게 되면 실시간성 보장이 힘들 수밖에

없다.

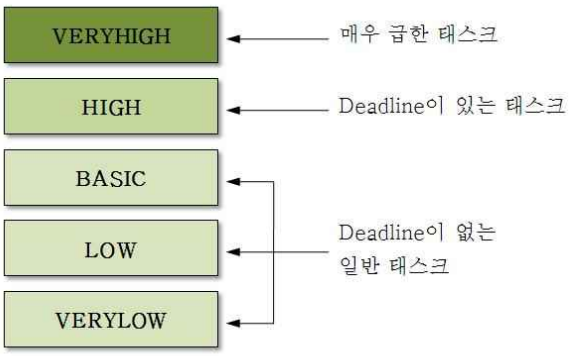
여기서 우리가 원하는 결과는 (그림 2)의 (c)와 같이 t_3가 t_0를 선점하여 t_3의 deadline을 만족시키고 t_0의 수행을 늦추면 된다. 본 논문은 (그림 2)의 (c)와 같이 선점할 수 있는 EDF 스케줄링을 제시하고자 한다.

4. 선점형 EDF 스케줄러

이번 절은 본 논문에서 제시하고자 하는 선점형 EDF 스케줄러에 대하여 알아본다. 선점형 EDF의 기본 정책, 구조, 사용방법에 대해 자세히 살펴본다.

4.1. 기본 정책

선점형 EDF는 Priority Level Scheduler를 기본으로 사용한다. (그림 3)에서 Priority Level Scheduler는 5개의 우선순위가 있다. 여기서 선점형 EDF에 맞춰서 모든 태스크의 우선순위를 (그림 3)과 같이 할당한다.



(그림 3) 태스크의 종류에 따른 우선순위 할당

우선 VERYHIGH 우선순위에 할당되는 작업은 매우 급한 태스크이다. 이것은 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 정렬되어있다. 이는 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.

그러면 VERYHIGH와 HIGH 우선순위를 구분시키는 방법에 대해 알아본다. 일단 현재 수행 중인 태스크를 T_c(Current Task)라 하고, 지금 막 포스팅된 태스크를 T_n(New Task)라 하자. T_c는 현재 수행 중인 작업이므로 T_c의 deadline은 현재 포스팅된 모든 태스크보다 빠르다. T_n이 포스팅되면 deadline이 가장 빠른 T_c와 비교를 한다. 이때 다음과 같이 세 가지 경우가 있다.

① T_n의 deadline이 T_c의 deadline 보다 늦다.
② T_n의 deadline과 T_c의 deadline은 같다.
③ T_n의 deadline이 T_c의 deadline 보다 빠르다.

여기서 T_c의 deadline은 T_c가 반드시 수행이 완료되어야하는 시간이고, T_n의 deadline은 T_n이 반드시 완료되어야하는 시간이다. ①의 경우를 보면 T_c의 deadline이 T_n보다 빠르다. 이것은 T_n이 지금 당장 수행될 정도로 급한 것은 아니다. 이때 T_n은 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.

②의 경우는 T_c와 T_n의 deadline이 같다. 이것은 T_c가 그냥 수행될 수도 있고, T_n이 T_c를 선점하여 수행할 수 있다. 그러나 선점을 하는 것은 Context Switch 같은 오버헤드가 발생하므로 T_c가 그냥 수행되도록 하는 것이 더 효율적이다. T_n은 HIGH 우선순위를 부여하고, HIGH 우선순위 큐에 삽입한다. 여기서 T_n은 deadline이 T_c의 deadline과 같으므로 HIGH 우선순위 큐 제일 앞쪽에 삽입하여 T_c 수행 후 T_n이 수행되도록 한다.

③의 경우는 T_n의 deadline이 더 빠르다. 이런 경우는 T_n이 T_c보다 더 급한 작업이므로 선점이 필요하다. 우선 T_n은 VERYHIGH 우선순위를 부여한다. 그러면 Priority Level Scheduler가 T_c의 작업을 멈춰서 해당 태스크의 정보를 메모리에 저장한 후 T_n이 수행되도록 한다. T_n이 T_c를 선점함으로써 T_c의 수행 때문에 deadline을 놓치지 않게 된다. 결국 TinyOS의 실시간성을 좀 더 높이는 효과가 있게 된다.

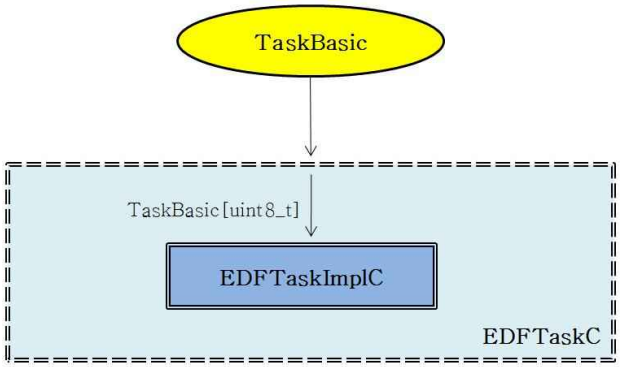
4.2. 구조

앞에서 언급한 것처럼 선점형 EDF는 Priority Level Scheduler를 기반으로 한다. 그러나 Priority Level Scheduler의 단점으로 우선순위를 컴파일 하기 전에 주어야 하는 문제가 있다. 선점형 EDF는 매 순간의 deadline에 따라 우선순위를 판단해야 하기 때문에 Priority Level Scheduler를 바로 적용하기 어렵다. 그래서 본 논문은 Priority Level Scheduler를 약간 수정해서 선점형 EDF를 만들고자한다.

선점형 EDF의 기본적인 컴포넌트 구조는 (그림 4), (그림 5)와 같다. 총 5개의 컴포넌트로 이루어 졌으며, (그림 5)에서 제일 하단의 2개의 컴포넌트는 Priority Level Scheduler의 일부분이다. (그림 4), (그림 5)는 TinyOS 2.0에서 사용하는 컴포넌트 다이어그램을 사용한다. 타원은 인터페이스를 나타내고, 한 줄로 된 사각형은 module 컴포넌트이며, 두 줄로 된 사각형은 configuration 컴포넌트이다. 또한 점선은 generic 컴포넌트를

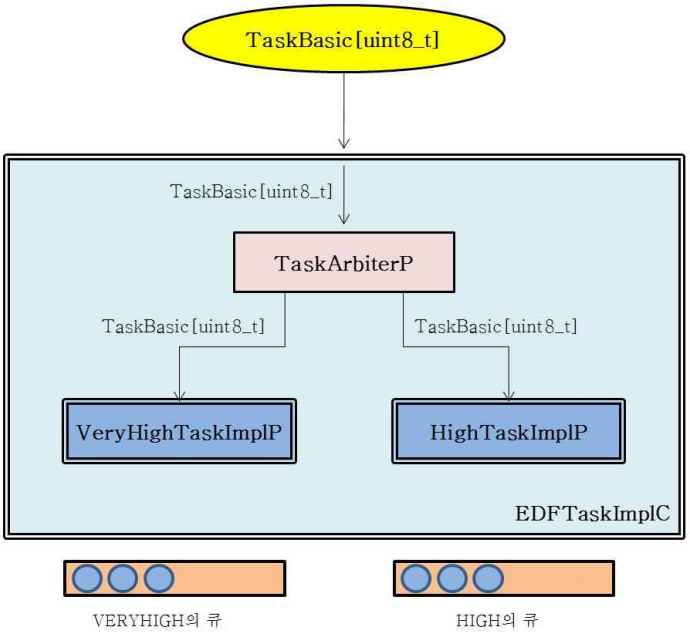
나타낸다.

응용프로그램 개발자는 최상위 컴포넌트인 (그림 4)의 *EDFTaskC*를 연결하여 선점형 EDF를 사용할 수 있다.



(그림 4) 선점형 EDF 스케줄러의 컴포넌트 구조

*EDFTaskC*가 제공하는 인터페이스는 *TaskBasic*으로 기본적으로 TinyOS에서 제공해준다. 또한 *EDFTaskC*는 generic 컴포넌트로 여러 개의 인스턴스를 생성할 수 있게 한다. *EDFTaskC에 연결된 EDFTaskImplC*는 실질적인 구현 부분으로 제공하는 인터페이스는 *EDFTaskC*와 동일하다. 그러나 태스크를 구분하기 위해 *TaskBasic[uint8_t]*와 같이 8bit 정수 값을 식별자로 사용한다.



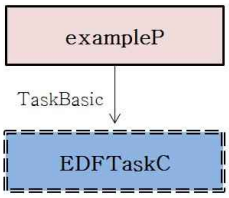
(그림 5) EDFTaskImplC의 구조

(그림 5)의 *EDFTaskImplC*는 3개의 컴포넌트로 구성되고, *TaskArbiterP*는 *VeryHighTaskImplP*와 *HighTaskImplP*를 사용한다. 최하단의 2개의 컴포넌트는 Priority Level Scheduler의 일부분이고 VERYHIGH와 HIGH 우선순위 태스크에 해당한다. 여기서 총 우선순위가 5개인데 2개만 쓰는 이유가 4.1절에서 언급했듯이 선점형 EDF는 VERYHIGH와 HIGH만 사용하기 때문이다.

*TaskArbiterP*는 *EDFTaskImplC* 컴포넌트에서온 *post command*를 받은 후 4.1절에 나온 정책에 따라 우선순위를 부여한 후 해당 우선순위의 컴포넌트로 *post command*를 보낸다. HIGH 태스크인 경우 *HighTaskImplP* 컴포넌트의 태스크 큐에 deadline 순서에 따라 삽입된다. VERYHIGH 태스크인 경우 *VeryHighTaskImplP* 컴포넌트로 가서 현재 수행 중인 태스크를 선점하여 수행한다. 스케줄러에 의해 태스크가 수행될 차례가 오면 큐에서 *event*를 발생하여 *TaskBasic* 인터페이스를 통해 사용자 응용프로그램까지 *event*가 도달하게 된다. *event* 함수로 정의된 사용자 태스크는 *event*를 받고 태스크가 수행된다.

4.3. 사용

앞 절에서는 선점형 EDF의 구조에 대해 알아보았다. 이번 절은 실제 개발자가 선점형 EDF를 어떻게 사용하는지 알아본다. 예제 컴포넌트는 *exampleP*이고 *TaskBasic* 인터페이스를 사용한다.



(그림 6) 선점형 EDF 사용예시 (wiring 부분)

컴포넌트 연결은 (그림 6)과 같이 *EDFTaskC*를 *TaskBasic* 인터페이스로 연결한다.

```
module exampleP{
    . . .
    // 태스크 인터페이스 사용
    uses interface TaskBasic as EDFTask;
        . . .
}
implementation{

                . . .

        // deadline을 상수 DEADLINE으로 post함
        call EDFTask.postTask(DEADLINE);

                . . .

    // 사용자 태스크의 구현 부분
    event void EDFTask.runTask(){
                . . .
                // 태스크 코드
                . . .
    }

                . . .

}
```

(그림 7) 선점형 EDF 사용예시 (모듈 부분)

또한, 개략적인 소스 구조는 (그림 7)과 같다. 태스크를 *post* 할 때는 *TaskBasic*의 *postTask()*를 호출하면 되고, 인자 값으로 deadline 값을 넣어준다. 태스크 구현은 *TaskBasic*의 *runTask()*에 구현하면 된다.

5. 결 론

본 논문은 TinyOS의 실시간성을 높이기 위해 선점형 EDF 스케줄링을 제시하였다. 기존에 있던 비선점 기반의 EDF 스케줄링은 매우 급한 태스크가 들어와도 현재 수행되는 태스크가 완료될 때까지 기다려야하는 문제가 있다. 최근 연구에서 TinyOS에 선점기능을 추가한 Priority Level Scheduler가 제안되었다. 본 논문은 이것을 사용하여 선점 가능한 EDF 스케줄러를 제안한다. 그러나 Priority Level Scheduler는 컴파일 이전에 우선순위를 미리 할당해야 하는 문제가 있다. 본 논문은 이것을 변형하여 수행 중에도 우선순위를 할당할 수 있도록 한다.

향후에는 선점형 EDF 스케줄러를 사용하여 실시간성이 필요한 응용분야에서 널리 활용될 것으로 기대한다. 또한, 이것을 더욱 발전시켜 TinyOS의 실시간성을 더욱 증가시킬 것으로 기대한다.

참고문헌

[1] 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), pages 93-104, Cambridge, MA, Nov. 2000.

[2] 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, pages 1−11, June 2003.

[3] 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.

[4] C.L. Liu and James W. LayLand. "Scheduling Algorithms for Multiprogramming in a Hard Real−Time Environment." J.ACM, 20, pages 40−61, 1973

[5] TEP 106 : “Schedulers and Tasks” http://www.tinyos.net/tinyos−2.x/doc/html/tep106.html