TinyOS의 태스크 결합을 통한 비선점형 실시간 스케줄러 구현 방안

손치원*, 탁성우**
*부산대학교 컴퓨터공학과
**부산대학교 정보컴퓨터공학부
e-mail: sonchiwon@gmail.com

Non-preemptive Real-time Scheduling in TinyOS Using TinyOS Task Combination

Chiwon Son*, Sungwoo Tak**
*Department of Computer Science, Pusan National University
**School of Computer Science and Engineering, Pusan National University

요 약

TinyOS는 현재 가장 널리 사용되는 센서 노드용 운영체제이지만, 태스크의 실시간성을 지원하지 않는다는 단점이 있다. 이에 TinyOS에 실시간성을 부여하기 위한 다양한 연구가 진행되었다. 그러나 이들 연구는 TinyOS의 사용자 태스크에 대한 실시간성만을 고려하여, TinyOS 플랫폼이 제공하는 태스크가 포함된 실제의 센서 노드 작업에 대해서는 실시간성을 만족시키지 못한다는 문제점이 있다. 따라서 본 논문에서는 TinyOS에서 센서 노드 작업의 실시간성을 지원하는 새로운 스케줄링 기법을 제안하고자 한다. 이를 위해 기존 연구의 스케줄링 기법을 센서 노드 작업에 적용했을 때 나타나는 작업 중첩 현상과 우선순위 조정 현상을 분석하고, 이를 효율적으로 해결하는 비선점형 EDF(Earliest Deadline First) 작업 스케줄링 기법을 구현하였다. 그리고 제안한 스케줄링 기법은 TinyOS의 이벤트 기반 비선점형 속성을 유지하여 제한된 하드웨어 자원을 가지는 센서 노드에 적합하다는 것을 확인하였다.

1. 서론

센서 노드는 운용 중 전력 공급이 불가능하고, 소용량의 메모리를 사용하는 특징이 있다. 이와 같이 제한된 하드웨어 자원을 가진 센서 노드를 효율적으로 운용하기 위해 지금까지 다양한 센서 노드용 운영체제가 개발되었다[1]. 그 중 TinyOS는 센서 노드의 전력 및 메모리 사용량을 최소화함으로써, 현재 가장 널리 사용되는 센서 노드용 운영체제로 자리 잡았다[2].

이와 같은 TinyOS의 저전력 및 초소형 장점은 TinyOS의 이벤트 기반 비선점형 스케줄링 정책을 통해 구현된다. TinyOS는 실행 중인 태스크가 없을 경우 프로세서를 수면 상태로 천이하고 이벤트(인터럽트) 발생 시 구동을 재개함으로써 최적의 전력 효율을 보인다. 또한 TinyOS의 태스크들은 서로 선점하지 않기 때문에 현재 프로세서의 상태(컨텍스트)를 저장하는 메모리를 사용하지 않는다.

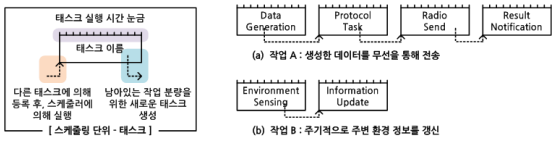

그림 1. TinyOS의 작업 시나리오

그러나 TinyOS는 *비선점형 선입선출 스케줄링 기법*을 사용함으로써, 우선순위가 높은 태스크에 대한 지원을 고려하지 않는다. 이로 인한 문제점을 파악하기 위해 **[그림 1]**과 같은 시나리오를 분석하였다. TinyOS가 설치된 센서 노드는 두 개의 *작업 A*와 *작업 B*를 수행하고 있다. 그리고 작업 B는 주기적으로 실행되어 작업 A보다 높은 우선순위를 가진다고 가정한다. 각 작업은 이벤트 핸들러를 통해 호출된 *사용자 태스크*로 시작된다. 사용자 태스크는 센서 노드의 I/O를 수행하는 TinyOS의 *플랫폼 태스크*를 순차적으로 호출하여 전체 작업을 수행한다.

[그림 1] 시나리오를 기반으로 TinyOS의 비선점형 선입선출 스케줄링 과정을 **[그림 2]**에 자세히 나타내었다. TinyOS의 태스크는 *태스크 큐*에 등록 후 프로세서가 유휴(Idel) 상태일 때 페치(Fetch)되는 *지연 호출(Deferred Procedure Call)*[3] 과정을 거친다. 따라서 각 작업의 최초 태스크(사용자 태스크)는 이벤트 핸들러가 완료되기까지의 지연 시간(①)을 가진다. 비선점형의 TinyOS에서도 인터럽트는 실행중인 태스크를 선점할 수 있다. 그러나 인터럽트 컨텍스트에서는 동일한 인터럽트 발생 시 해당 정보가 소실된다. 따라서 인터럽트는 즉시

※ 본 연구는 국토해양부 첨단도시기술개발사업 - 지능형국토정보기술혁신 사업과제의 연구비지원 (07국토정보C04)에 의해 수행되었습니다.

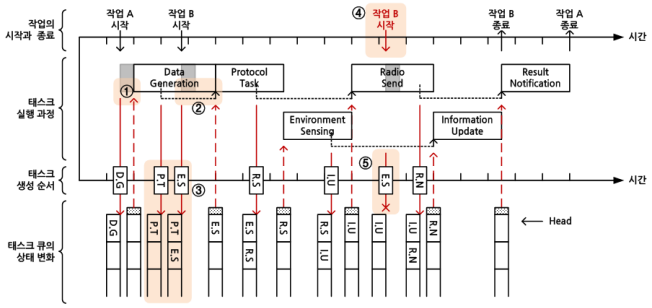

그림 2. TinyOS의 비선점형 선입선출 스케줄링 기법

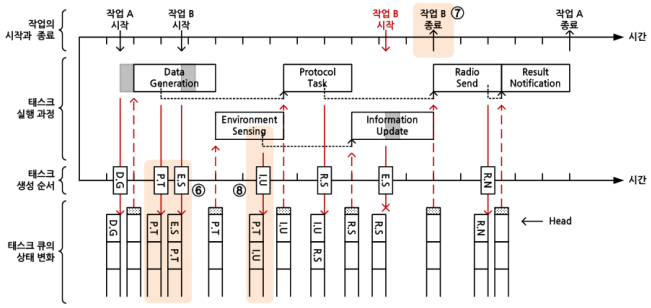

그림 3. TinyOS의 비선점형 EDF 태스크 스케줄링 기법

이벤트 핸들러를 지연 호출한 후 인터럽트 컨텍스트를 벗어난다. 이러한 이벤트 핸들러의 지연 호출 때문에, 사용자 태스크의 지연 시간에는 현재 실행중인 태스크의 완료까지 남은 시간(②)이 포함된다. 그리고 생성된 사용자 태스크는 플랫폼 태스크와 마찬가지로 자신이 속한 작업의 우선순위에 관계없이 생성 순서대로 태스크 큐에 등록(③)된다. 그러나 이러한 기법은 작업의 마감시간 (④)을 고려하지 않기 때문에, 주기적 작업이 실패하는 경우(⑤)가 발생할 수 있다.

TinyOS는 이와 같은 현상을 보완하기 위해, 작업이 완료되기 전에 동일한 작업이 다시 시작되어 이전 작업의 결과 데이터가 변경될 위험이 있는 태스크의 경우, 전역 메모리를 사용하여 해당 결과를 저장해 두고, 추후 시스템이 안정되면 저장된 데이터를 순차적으로 처리하는 방식을 제안하고 있다. 그러나 이 방식은 필요한 배열의 크기를 예측하기 어렵고, 센서 노드의 제한된 메모리를 비효율적으로 사용하게 된다. 따라서 이와 같은 문제가 발생하는 근본적인 원인을 제거할 필요가 있다. 그 방안은 TinyOS에서 태스크의 실시간 스케줄링을 지원하는 것이다. 이에 본 논문에서는 작업의 마감시간을 기반으로 태스크의 동적 우선순위를 지원하는 새로운 비선점형 스케줄링 기법을 제안하고, 해당 기능을 수행하는 TinyOS의 스케줄러를 구현하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 기존에 연구된 TinyOS 스케줄링 기법에 대한 문제점을 살펴본다. 3장에서는 본 논문에서 제안하는 TinyOS의 새로운 스케줄링 기법을 설명한다. 그리고 4장에서 제안한 스케줄링 기법의 성능을 분석하며 결론을 맺는다.

2. 작업 중첩 현상과 우선순위 조정 현상

지금까지 비선점형 운영체제인 TinyOS에 실시간성을 부여하기 위한 다양한 연구가 진행되었다. 그 목적은 전체 시스템의 재개발 비용을 없애고 기존 응용과의 호환성을 유지한 채로 실시간 스케줄링을 지원하여 TinyOS의 성능을 개선하려는 것이다.

이와 같은 연구 중 가장 간단한 아이디어는 작업의 마감시간을 기반으로 태스크 큐를 재배열하는 *비선점형 EDF(Earliest Deadline First) 태스크 스케줄링 기법*[3]이다. **[그림 3]**은 비선점형 EDF 태스크 스케줄링 과정을 **[그림 1]**의 시나리오에 적용한 결과를 나타내고 있다. 새로운 태스크가 등록되면 대기 중인 태스크와 비교하여 우선순위가 높은(마감시간이 가까운) 태스크 순서대로 태스크 큐가 정렬(⑥)된다. 그 결과 높은 우선순위를 가지는 태스크가 비선점형 선입선출 스케줄링 기법에 비해 빠른 시간(⑦)에 종료된다.

그러나 이러한 태스크 큐의 재배열 작업은 TinyOS의 플랫폼이 제공하는 태스크 간에는 적용할 수 없다. 그 이유는 다음과 같다. 이 기법이 적용된 스케줄러는 TinyOS의 태스크 생성 *인터페이스*를 수정하여 응용 프로그램 영역의 이벤트 핸들러에서 사용자 태스크를 호출할 때 마감시간을 입력받을 수 있도록 구현된다. 그러나 TinyOS가 제공하는 플랫폼 태스크는 기존의 태스크 생성 인터페이스를 사용하여 마감시간 정보를 입력할 수단이 없다. 따라서 이 기법은 사용자 태스크 간의 마감시간(우선순위) 비교만 가능할 뿐이다.

그 결과, 작업에 포함된 플랫폼 태스크들은 마감시간 비교 없이 생성 순서대로 태스크 큐에 배열(⑧)된다. 따라서 비선점형 선입선출 스케줄링 기법과 마찬가지로 작업의 마감시간을 만족시키지 못하는 경우가 발생한다. 이와 같이 플랫폼 태스크들의 우선순위를 비교할 수 없기 때문에 두 개의 작업을 구성하는 플랫폼 태스크가 번갈아가며 실행되는 현상을 본 논문에서는 *작업 중첩(Overlap) 현상*이라 정의한다.

작업 중첩 현상은 플랫폼 태스크의 인터페이스를 수정함으로써 해결할 수 있다. 그러나 이 작업은 TinyOS에서 제공하는 드라이버를 모두 재작성하는 비용이 필요하므로, 본 논문에서는 이러한 개선 방법은 고려하지 않는다.

또 다른 연구에서는 TinyOS의 비선점형 스케줄링 기법이 가지는 한계를 극복하기 위해 태스크 간 컨텍스트 스위칭을 사용하는 *선점형 우선순위 스케줄링 기법*[4]을 제안하였다. 선점형 우선순위 스케줄링 기법을 **[그림 1]**의 시나리오에 적용한 결과가 **[그림 4]**에 나타나 있다. 이 연구는 TinyOS에서 하드웨어 인터럽트가 수행중인 태스크를 선점한다는 사실에 착안하여, 다음과 같은 태스크 스케줄링 동작을 설명하고 있다.

태스크는 5단계의 우선순위 중 하나로 지정된다. 우선순위가 같은 태스크 간에는 선입선출 방식을

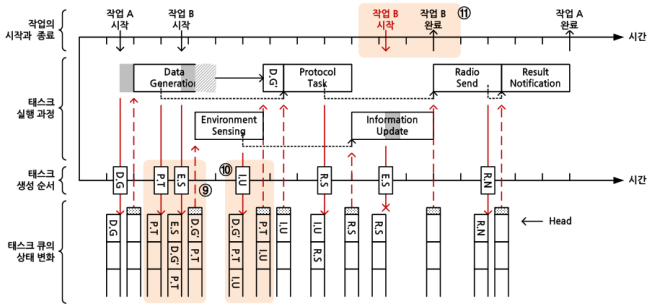

그림 4. TinyOS의 선점형 우선순위 스케줄링 기법

사용한다. 생성된 태스크가 실행 중인 태스크보다 우선순위가 낮으면, 자신보다 높은 우선순위를 가지는 태스크들이 모두 완료될 때까지 실행이 지연된다. 만약 생성된 태스크가 실행중인 태스크보다 우선순위가 높으면, 프로세서의 모든 레지스터 값(컨텍스트)을 메모리에 저장한 후, 새로운 태스크를 페치(⑨)한다. 이 때 컨텍스트를 저장하기 위해 스택 자료구조를 사용한다. 그 결과 선점 당한 순서대로 태스크 큐가 배열(⑩)되어, 우선순위가 높은 태스크의 완료 이후에 적법한 순서로 남은 태스크들이 수행된다.

그러나 선점형 우선순위 스케줄링 기법에서도 작업 중첩 현상 때문에 마감시간 내에 작업을 완료하지 못하는 경우가 발생(⑪)한다. 또한 이 기법은 플랫폼 태스크에 대한 호환을 위해 플랫폼 태스크를 *기본(Basic)* 우선순위로 지정하고, 사용자 태스크의 우선순위는 플랫폼 태스크에 상대적인 우선순위를 지정하는 방식을 사용한다. 그러나 사용자 태스크는 이벤트 핸들러에서 호출되는데, 이벤트 핸들러가 플랫폼 태스크이기 때문에, 기본보다 높은 우선순위를 가지는 사용자 태스크의 경우, 기본 우선순위를 가지는 이벤트 핸들러가 페치될 때까지 실행이 지연된다. 이와 같이 사용자 태스크가 플랫폼 태스크의 우선순위에 의해 영향을 받는 현상을 본 논문에서는 *우선순위 조정 현상*이라 정의한다.

우선순위 조정 현상을 해결하는 방법은 사용자 태스크를 이벤트 핸들러를 거치지 않고 하드웨어 인터럽트 통지 루틴에서 비동기적으로 태스크 큐에 등록하는 것이다. 그러나 이 방법은 TinyOS의 응용이 하드웨어 인터럽트에 밀접한 구조를 가지게 한다. 이는 사용자가 응용 프로그램 작성 시 TinyOS가 제공하는 하드웨어 추상화 컴포넌트를 사용할 수 없음을 의미한다. 무엇보다 이와 같은 정적 우선순위에 따른 스케줄링 기법은 마감시간과 같이 동적으로 변화하는 작업 우선순위를 가지는 태스크들의 실시간 처리에는 적합하지 않다는 문제가 있다.

3. 태스크 결합을 통한 작업 단위 스케줄링 기법

지금까지 살펴 본 TinyOS의 실시간 스케줄링 기법에 관한 연구는 공통적으로 TinyOS가 가지는 *컴포넌트 기반*의 구조적 특징을 고려하여 TinyOS의 스케줄러 컴포넌트를 보완하는데 초점을 둔다. 따라서 이들 연구에서 제안하는 TinyOS의 스케줄링 기법은 이벤트 기반 구동 방식을 유지하고 스케줄러에 의한 오버헤드를 최소화하여 TinyOS의 저전력 및 소형성 장점을 유지한다. 또한 이와 같이 구현된 스케줄러는 기존 스케줄러 컴포넌트의 인터페이스를 보존한 채 확장 기능을 제공하기 때문에 이미 개발된 응용과의 *연속적 결합(Seamless integration)*을 지원한다. 즉, 이들 연구는 TinyOS의 제약사항을 만족시키며 TinyOS의 스케줄러를 개선하는 타당한 방법을 보여주고 있다.

그러나 이들 연구는 공통적으로 TinyOS의 단일 태스크에 대한 실시간성을 고려함으로써, 실제 상황에서 센서 노드의 작업은 다수의 태스크 결합으로 구현된다는 사실을 간과하고 있다. 따라서 관련 연구에서 제안한 스케줄링 기법들은 응용 프로그램에서 작성되는 사용자 태스크에 대해서는 만족할 만한 실시간성 성능을 보여주지만, **[그림 1]**의 시나리오에 적용한 결과에서 알 수 있듯이, 센서 노드의 의미 있는 작업에 대해서는 실시간성을 만족시키지 못한다.

이에 본 논문에서는 센서 노드가 수행하는 작업 단위의 실시간성을 지원하기 위해 작업을 구성하는 태스크들을 논리적으로 결합하여 태스크 그룹 간의 우선순위를 비교하는 새로운 TinyOS의 스케줄링 기법을 제안한다. 본 논문에서 제안하는 *비선점형 EDF 작업 스케줄링 기법*은 메모리 효율을 위해 비선점형으로 설계되었고, 마감시간 기반의 동적 우선순위를 지원한다. 이와 같은 스케줄링 기법을 구현하기 위해 먼저 센서 노드 작업의 특징을 분석하였다.

센서 노드 작업을 위한 TinyOS의 태스크 호출은 **[그림 5]**와 같은 태스크 그래프[5]로 생각할 수 있다. 본 논문에서는 이러한 태스크 그래프의 노드 집합을 *TinyOS 작업*이라 정의한다. 즉, TinyOS 작업은 인터럽트 처리 루틴을 제외한 모든 동기적 태스크의 집합을 뜻한다.

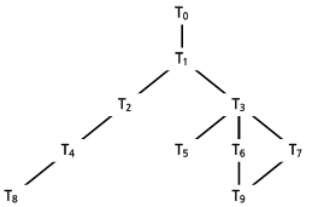

그림 5. 태스크 그래프

[그림 5]는 TinyOS 작업의 태스크 호출 관계를 정적으로 나타내며, 동적인 TinyOS 작업의 실행 과정은 **[그림 6]**에 나타나 있다. 비동기적인 하드웨어 인터럽트는 플랫폼을 거쳐 동기화 되어 응용 영역에서는 이벤트 핸들러(T_0)로 인식된다. 이벤트 핸들러에서 응용 프로그램은 사용자 태스크를 마감시간 정보와 함께 태스크 큐에 등록한다. 이와 같은 태스크를 TinyOS 작업의 *구동 태스크*(T_1)라 한다. 최초 마감시간을 가지는 구동 태스크가 페치되면 해당 태스크는 작업의 목적을 위한 플랫폼 태스크들을 호출한다. 이 때 다수의 플랫폼 태스크를 호출하더라도 호출 순서에 따라 동일한 태스크 큐에 정렬된다. 그 중 I/O를 수행하는 플랫폼 태스크는

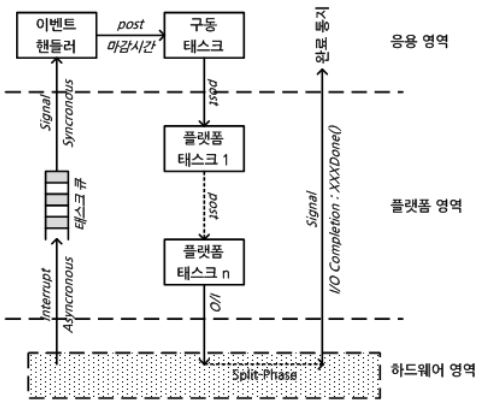

그림 6. TinyOS 작업의 실행 과정

프로세서에 독립적인 동작을 수행한 후 완료를 통지하는 인터럽트를 발생시키는데 이와 같은 과정을 *Split-Phase*[3]라 한다. 이 인터럽트는 다시 플랫폼 영역을 거쳐 동기화된 후 응용 영역에 이벤트 핸들러 형식으로 전달된다. 응용 프로그램은 이러한 완료 통지를 통해 하나의 TinyOS 작업이 완료되었음을 인지한다.

이와 같이 TinyOS 작업을 태스크들의 그룹으로 정의함으로써, 비선점형 EDF 작업 스케줄링 기법은 태스크 간의 비선점 환경에서도 작업 간의 선점을 구현할 수 있다. 그 방법을 설명하기 위해 **[그림 1]**의 시나리오에 비선점형 EDF 작업 스케줄링 기법을 적용한 결과를 **[그림 7]**에 나타내었다. 스케줄러는 구동 태스크가 페치되면 해당 TinyOS 작업에 대한 식별 정보를 저장한다. 이 정보는 해당 TinyOS 작업이 완료 통지 이벤트를 인식할 때까지 유지된다.

태스크 실행 중 새로운 작업이 발생하여 이벤트 핸들러가 구동 태스크를 마감시간과 함께 태스크 큐에 등록하면 스케줄러는 식별 정보를 통해 현재 수행 중인 작업의 마감시간을 구한 후, 두 마감시간을 비교하여 선점 여부를 판단한다. 만약 새로운 작업의 마감시간이 더 가깝다면 작업 식별 정보를 새로운 작업으로 지정하고, 다른 방식으로 태스크 큐를 관리한다. 그 방식은 선점 발생 후부터 생성되는 태스크(⑫)를 선점 발생 시 태스크 큐에서 대기하고 있는 태스크(⑬)의 앞으로 배열하는 것이다. 이와 같은 태스크 큐 조작을 통해 플랫폼 태스크는 우선순위를 상속(⑭)받아 결국 선점한 작업이 먼저 완료된다. 그 후, 작업 식별 정보가 선점 이전으로 복원됨과 함께 대기하고 있던 플랫폼 태스크가 수행(⑮)된다. 제안한 스케줄링 기법은 이러한 과정을 통해 기존의 스케줄링 기법과 달리 우선순위가 높은 작업의 마감시간을 만족시키고 있다.

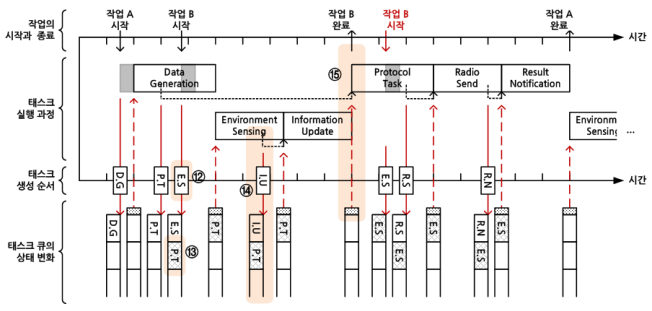

그림 7. TinyOS의 비선점형 EDF 작업 스케줄링 기법

4. 결론

지금까지 살펴본 바와 같이 본 논문에서 제안한 TinyOS의 비선점형 EDF 작업 스케줄링 기법은 기존의 스케줄링 기법에서 해결할 수 없었던 센서 노드 작업의 마감시간을 만족시킨다. 제안된 스케줄링 기법이 이러한 향상된 실시간성을 나타내는 이유는 다음과 같다.

제안된 스케줄링 기법은 플랫폼 태스크에 우선순위를 지정하여 작업 중첩 현상을 해결한다. 이를 위해 플랫폼 태스크를 구성 요소로 가지는 TinyOS 작업을 정의하고, TinyOS의 우선순위를 플랫폼 태스크에 상속시키는 방법을 사용하였다. 또한 동기적 태스크를 통해 우선순위 조정 현상을 해결한다. 이는 플랫폼 태스크를 사용자 태스크(구동 태스크)를 에 비해 높은 우선순위를 가지도록 함으로써 간단히 구현되었다.

그리고 이 스케줄링 기법은 TinyOS의 이벤트 기반 비선점형 특징을 그대로 유지한다. 따라서 본 연구를 통해 기존의 자원 효율성 장점에 실시간성이 추가된 TinyOS는 추후 적용 분야가 더욱 확대될 것으로 기대된다.

참 고 문 헌

[1] 박승민, “센서 네트워크 노드 플랫폼 및 운영체제 기술 동향,” 전자통신동향분석, 제21권 제1호, 2006년

[2] 서창수, 이철희, 김형준, “유비쿼터스 센서 네트워크 시스템”, 한백전자 기술연구소, 2006년

[3] Philip Levis and Cory Sharp, “TEP:106 Schedulers and Tasks,” www.tinyos.net

[4] Cormac Duffy, Utz Roedig, John Herbert, and Cormac J. Sreenan, “Adding Preemption to TiinyOS,” EmNets'97, 2007

[5] Cormac Duffy and John Herbert, “Achieving Real-Time Operation in TinyOS,” www.tinyos.net