비선점 구간을 갖는 태스크들을 위한 저전력 실시간 스케줄링

Low Power Real-Time Scheduling for Tasks with Nonpreemptive Sections

김남진, 김인국
단국대학교 공학대학 컴퓨터과학과

Nam-Jin Kim(njkim@dku.edu), In-Guk Kim(igkim@dankook.ac.kr)

요약

RM이나 EDF에 기반을 둔 실시간 스케줄링 알고리즘들은 태스크들이 선점 가능함을 가정하고 있지만 현실적으로는 선점 불가능한 부분이 존재할 수도 있다. 또한 프로세서의 전력 소모를 줄이기 위한 기존의 스케줄링 알고리즘은 태스크 이용률을 기반으로 하여 블로킹 구간이 있는 경우와 없는 경우를 기준으로 두 가지 프로세서 속도(S_H, S_L)를 결정한다. 이 알고리즘에서 높은 속도로 동작하는 S_H 구간은 블로킹에 의한 우선순위 역전이 발생하는 구간이며 이 구간의 길이는 블로킹 구간을 포함한 태스크 종료시한까지로 설정된다. 본 논문에서는 S_H 구간의 길이를 기존의 알고리즘보다 더 짧게 하여 전력소모율을 낮추는 방법을 제안하였다. 모의실험을 통해, 제안된 알고리즘의 전력소모율이 기존 알고리즘에 비하여 최대 13%만큼 감소되었음을 볼 수 있었다.

■ 중심어 : | 실시간 스케줄링 | 저전력 | 비선점 |

Abstract

The basic real-time scheduling algorithms based on RM or EDF approaches assume that the tasks are preemptive, but the tasks may contain nonpreemptive sections in many cases. Also the existing scheduling algorithm for reducing the power consumption of the processor is based on the task utilizations and determines the processor speed S_H or S_L according to the existence of the blocking intervals. In this algorithm, the S_H interval that operates in high speed is the interval during which the priority inversion by blocking occurs, and the length of this interval is set to the task deadline that includes the blocking intervals. In this paper, we propose an improved algorithm that can reduce the power consumption ratio by shortening the length of the S_H interval. The simulation shows that the power consumption ratio of the proposed algorithm is reduced as much as 13% compared to the existing one.

■ keyword : | Real-Time Scheduling | Low Power | Nonpreemptive |

I. 서 론

최근 컴퓨터 시스템의 발전 방향과 적용 모델이 기존의 대형, 데스크톱 기준에서 휴대가 가능한 소형기기, 로봇, 센서장치등과 같이 배터리를 사용하는 기기에 집중되고 있으며 이들 시스템에 대한 주요 이슈는 소형화, 저전력화 등에 있다. 특히 저전력 부분은 시스템의 가용 시간 및 운영 시간과 밀접하게 연관되어 있기 때문에 하드

접수번호 : #091207-002
접수일자 : 2009년 12월 07일
심사완료일 : 2010년 01월 02일
교신저자 : 김인국, e-mail : igkim@dankook.ac.kr

웨어 단계에서는 물론 운영체제 단계에서도 저전력 특성을 추가하여 가용 시간을 늘리려는 연구가 많이 진행되었다.

현재 단일 프로세서 시스템에서 대표적인 실시간 태스크 스케줄링 알고리즘은 RM(Rate Monotonic) 스케줄링 알고리즘과 EDF(Earliest Deadline First) 스케줄링 알고리즘이다. 이 알고리즘들은 단일 프로세서 환경에서 실시간 태스크 스케줄링을 위한 최적의 알고리즘으로 알려져 있다[1][2]. 이러한 실시간 시스템은 시스템 특성에 따라 임베디드 시스템(embedded system) 및 휴대형 시스템에 탑재되어 운용될 수 있고, 이로 인해 전력소비를 최소화해야하는 문제가 발생되며 이를 해결하기 위한 다양한 방법들이 시도되었다.

Kim 등은 시스템에서 발생하는 슬랙 계산 방법을 Shin 등의 연구와 다른 방법을 사용하여 좀 더 효율적인 저전력 스케줄링 방법을 연구하였다[3][4]. Qadi 등은 비주기 태스크가 포함된 실시간 시스템에서 동적 전압 조절(Dynamic Voltage Scaling, DVS) 기법과 EDF 알고리즘을 사용하여 저전력 스케줄링 알고리즘 DVSST(Dynamic Voltage Scaling for Sporadic Task)를 고안하였고 이것을 임베디드 시스템에 구현하고 태스트 하였다[5]. Yao 등은 전력 소비를 최소화하기 위하여 간단한 job 스케줄링 모델을 제안하였다[6]. 또한 Aydin 등은 전력 소비 패턴이 다른 주기적인 실시간 태스크들의 종료시한을 지키기 위한 최적화된 속도 S_i를 계산하였다[7]. 이 논문에서는 전력 소비 패턴이 다른 태스크를 분류한 후 각각 다른 전력 소비 함수를 적용하였다. Hong 등은 주기적인 태스크의 종료시한을 지키면서 비주기 태스크들을 실행시키기 위한 온라인 휴리스틱 스케줄링 방법을 제시하였다[8].

이러한 저전력 스케줄링 알고리즘들은 대부분 동적 전압 조절이 가능한 프로세서에 기반을 두고 있는데 이들은 프로세서가 낮은 클록 속도로 동작할수록 더 낮은 전력이 소비되도록 설계되었다[9-12].

RM, EDF 알고리즘에 기반을 둔 저전력 실시간 스케줄링 알고리즘들에서 모든 태스크들은 우선순위에 따라 선점 가능함을 기본 전제로 하고 있다. 그러나 태스크는 다양한 종류의 임계영역에 의해 블록 될 수 있으며 이로 인하여 우선순위 역전(priority inversion)문제를 발생시킨다[15]. 우선순위 역전 문제를 해결하기 위한 연구들에서는 특정 프로토콜들을 사용하여 높은 우선순위 태스크의 블로킹 시간이 예측 가능하도록 하였으며 모두 BPIP(Basic Priority Inheritance Protocol)와 PCP(Priority Ceiling Protocol)에 기반을 두었다. Zhang 등은 임계영역이 있는 태스크 모델에서 저전력 실시간 태스크 스케줄링을 연구하였다[14]. 알고리즘에서는 상대적으로 높은 속도로 동작하는 구간의 길이를 높은 우선순위의 태스크를 블록 하는 현재 태스크의 종료시한으로 정하였으나 본 연구에서는 높은 속도로 동작하는 구간이 가능한 짧아지도록 알고리즘을 개선하였다.

논문의 구성은 다음과 같다. 2장에서는 본 논문에서 다루고자하는 태스크 모델 및 시스템 모델을 기술하고, 3장에서는 연구의 목적과 동기를 제시한다. 4장에서는 이를 구현한 방법과 알고리즘에 관하여 기술하며, 5장에서는 개선된 알고리즘의 효율성을 증명하기 위한 모의실험을 하고 그 결과를 설명한다. 끝으로 6장에서는 결론을 제시한다.

II. 시스템 모델

본 연구에서 시스템은 단일 프로세서 환경을 가정하였으며, EDF 스케줄링 알고리즘을 사용하였다. 시스템에서 작업의 처리는 태스크 단위로 선택되어 실행되며, 하나의 태스크 T_i는 여러 개의 job을 순차적으로 실행한다. 태스크는 다음과 같은 속성을 갖는다.

- R_i : 태스크 수행 가능 시간(Release Time).
- E_i : 태스크의 최악의 경우의 실행시간.
- P_i : 태스크 주기
- D_i : 상대적인 종료시한

E_i는 최대 프로세서 속도에서 태스크 처리를 위해 요구되는 시간이다. P_i는 태스크 주기를 나타내며 태스크 주기와 종료시한은 같다고 가정한다(D_i=P_i). 실시간 태스크는 주기적으로 수행되는 순차적인 명령의 단위인 job이 수행되는 것이며, 한 태스크는 여러 개의 job들의 시퀀스로서 T_i={$J_{i,1}$, $J_{i,2}$,....,$J_{i,n}$}와 같이 나타낼 수 있다. $J_{i,j}$는

태스크와 마찬가지로 $r_{i,j}$, $e_{i,j}$ ($\leq E_i$), $d_{i,j}$, $p_{i,j}$ 속성을 갖는다.

본 논문에서 태스크들은 선점 가능한 부분과 선점 불가능한 부분을 모두 가지고 있다고 가정하며 이들은 각기 다른 처리 형태를 갖는다. 선점 가능한 부분은 기존의 EDF 알고리즘에서처럼 높은 우선순위의 태스크에 의해 언제든지 선점 가능하며, 선점 불가능한 부분은 SRP(Stack Resource Policy)[13]에 의해 실행도중 블록되지 않음이 보장된다. 이러한 선점 불가능한 부분을 B_i라고 한다.

프로세서는 동적 전압 조절이 가능하고 이것은 CMOS 제작 기술에 의해 구현된다. CMOS는 정적 또는 동적으로 전력을 소비한다. 정적 전력(static power) 소비는 바이어스(bias)와 전류누수(current leakage)등이 있으며 대부분의 회로들에서 1mW이상 소모한다. 그러나 CMOS 마이크로프로세서들에서 대부분의 전력 소비는 동적 전력(dynamic power)이다. 동적 전력은 주로 디지털 회로의 캐패시턴스(capacitance)의 충전과 방전시에 전류를 소모하기 때문에 모든 디지털 회로는 상태 전환(state transition)에 전력을 소비한다. 이때 전력 소비는 다음의 식 (1)과 같다[14].

$$P = C_L N_{sw} V_{DD}^2 f \qquad (1)$$

여기에서, C_L은 출력캐패시턴스(output capacitance)이며, N_{sw}는 클록 당 스위칭 횟수이고 f는 클록 주파수이다. CMOS 디지털 회로에서 전력 소비는 공급 전압(V_{DD})의 제곱에 비례한다. 이상의 식 (1)에서 가능한 최대 및 최소 공급 전압은 각각 V_{max}와 V_{min}으로 나타내며 프로세서 최고 및 최저 속도는 각각 S_{max}과 S_{min}으로 나타낸다. 프로세서 공급 전압은 범위 내에서 이산단계(discrete step)로 적용된다. 전압이 변경될 때 발생하는 변환 지연 시간은 매우 짧으므로 무시한다.

III. Motivation Example

저전력 실시간 스케줄링의 목표는 프로세서 이용률이 낮을 때 실시간 특성을 만족시키면서 가능한 한 프로세서 동작 속도를 낮추는 것이다. 저전력 특성을 갖도록 설계된 시스템은 낮게 유지되는 전압으로 인해 프로세서 발열이 적고 배터리를 사용하는 휴대 장치인 경우 가용시간이 늘어나 시스템 활용을 극대화 할 수 있다. 실시간 태스크들은 일반적으로 선점 가능함을 전제로 설계된 스케줄링 알고리즘에 의해 수행된다. 그러나 실제 환경에서 태스크들은 메모리 공간, 자료구조, 파일, 하드웨어 장치 등을 공유하고 또한 이들의 사용이 상호 배제적으로 임계영역내에서 수행되기 때문에 우선순위에 의해 선점 불가능한 부분이 존재한다. 이렇게 선점 불가능한 부분이 포함된 실시간 태스크 시스템에서 소비전력을 낮추기 위한 스케줄링 알고리즘들 중 Zhang 등이 연구한 DS(Dual Speed) 알고리즘이 있다[14]. 이 연구는 태스크 집합의 이용률을 기반으로 속도 S_L, S_H를 계산한 후 구간의 특성에 따라 이 두 가지 속도 중 하나를 적용하는 저전력 실시간 스케줄링 알고리즘이다.

위 연구에서 속도 S_L은 임계영역에서 실행중인 태스크가 없는 경우의 전체 태스크 이용률로 계산했으며, B_i에 의한 이용률 점유가 없기 때문에 속도 S_H에 비해 상대적으로 작다. 속도 S_H는 전체 태스크 집합에서 임계영역으로 인한 블록 가능한 최대 길이 B_i를 고려하여 계산한 속도이며 속도 S_L에 비하여 상대적으로 크다. 임계영역으로 인한 태스크 블로킹 구간은 스케줄링 도중 수시로 나타나며 블로킹 구간이 나타나기 전까지 프로세서 속도는 S_L로 동작한다.

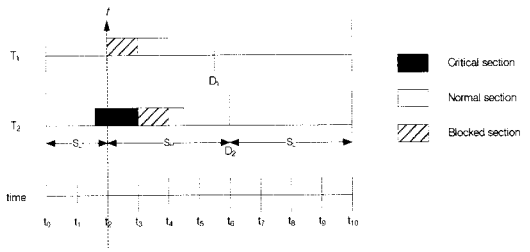

그림 1. 기존 알고리즘에서의 S_H 구간을 결정하는 방법

[그림 1] 에서 프로세서는 S_L로 수행을 시작하며 시점 t_2에서 태스크 T_2의 임계영역으로 인해 태스크 T_1이 블

록되는 경우 프로세서 속도는 태스크 T_2의 종료시한까지 S_H로 설정된다. 결과적으로 모든 구간을 V_{max}로 실행하는 일반적인 시스템보다 전력 소비가 감소하였다.

위의 설명에서 높은 우선순위 태스크가 낮은 우선순위 태스크의 임계영역 실행으로 인해 블록되는 경우에 낮은 우선순위 태스크의 종료시한까지 프로세서의 속도를 S_H로 설정하였다. 그러나 본 연구에서는 전력 소비 감소를 위하여 S_H 구간의 길이를 높은 우선순위의 태스크 종료시한으로 결정하여 [그림 2]와 같이 스케줄링 하였다.

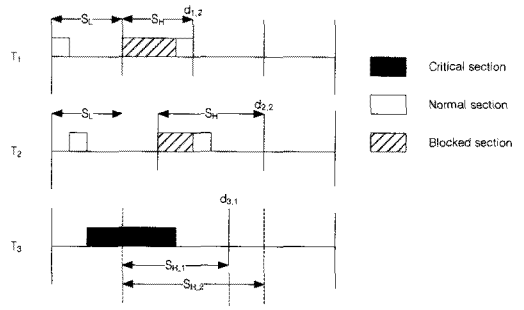

그림 2. 높은 우선순위 태스크 종료시한으로 SH 구간을 결정하는 방법

[그림 2]에서 태스크 T_1, T_2, T_3의 우선순위는 $T_1>T_2>T_3$로 가정하였다. 태스크 T_3의 임계영역은 $J_{1,1}$, $J_{2,1}$의 실행이 종료되고 $J_{3,1}$에서 시작되었다. 태스크 T_1의 $J_{1,2}$에서 두 번째 주기가 시작되고 태스크 T_3의 임계영역 실행으로 인해 블록 된다. 이때, S_H 구간의 길이는 DS 알고리즘에 의해 태스크 T_3의 $J_{3,1}$의 종료시한 $d_{3,1}$까지로 결정된다. 이상의 방법과 달리 S_H 구간의 길이를 더 줄이기 위해 태스크 T_1의 $J_{1,2}$의 종료시한 $d_{1,2}$로 S_H의 길이를 변경한다면 현재 시점에서는 더 짧아진다. 그러나 이후 태스크 T_2의 $J_{2,2}$가 다시 블록 되어 S_H 구간의 길이가 태스크 T_2의 $J_{2,3}$의 종료시한 $d_{2,3}$으로 재설정 되는 경우 기존의 방법보다 더 길어진다. [그림 2]에서 $S_{H,1}$은 DS 알고리즘에 의해 설정된 구간이고 $S_{H,2}$는 본 논문에서 가정한 방법으로 설정한 구간이다.

IV. 방 법

1. 임계영역을 고려한 정적 프로세서 속도 계산

임계영역이 있는 태스크들의 스케줄링에서는 낮은 우선순위의 태스크에 의해 높은 우선순위의 태스크가 선점당하는 우선순위 역전 현상이 발생할 수 있다. 이로 인하여 높은 우선순위 태스크의 실행이 지연되거나 최악의 경우 종료시한을 놓치는 경우가 발생할 수 있다. 이러한 이유로 실시간 스케줄링 알고리즘은 임계영역에서 실행중인 태스크를 고려하면서 높은 우선순위 태스크의 실행이 보장될 수 있도록 하여야한다. 이러한 태스크 모델에 대하여 Chen 등은 PCP를 EDF 알고리즘과 사용하여 DPCP(Dynamic PCP)알고리즘을 고안하였으며, n개의 주기적인 태스크 집합이 스케줄링 가능하기 위한 충분조건이 다음 식 (2)와 같음을 증명하였다[16].

$$\sum_{i=1}^{n} \frac{E_i + B_i}{T_i} \le 1 \qquad (2)$$

여기에서 B_i는 태스크 T_i의 job들을 블록할 가능성이 있는 임계영역의 최대 크기이다. 식 (2)는 매번 B_i 만큼의 블록될 가능성이 있는 실시간 태스크들의 전체 이용률이 1 보다 작거나 같은 경우 EDF 알고리즘으로 스케줄링 가능하다는 것을 의미한다. 이는 Chen 등이 PCP를 EDF 스케줄링 알고리즘에서 사용할 수 있도록 확장한 것이나 B_i가 매우 작아야 한다는 제한이 있다[13].

Baker 등은 n개의 주기적인 태스크들의 집합이 [정리 1]에서 주어진 조건을 만족할 때 스케줄링 가능함을 증명하였으며 이는 EDF 알고리즘에 SRP를 적용한 것이다[13]. SRP는 고정 우선순위 스케줄링과 동적 우선순위 스케줄링에서 모두에서 사용될 수 있으며 몇 가지 독특한 특성을 가지고 실시간 태스크의 공유자원 접근에 대한 동기화 프로토콜을 제공한다.

[정리 1] n개의 태스크들이 그들의 주기의 오름차순으로 정렬되어 있다고 가정한다. 이들은 식 (3)과 같을 때 EDF 알고리즘으로 스케줄이 가능하다.

$$\forall k \quad \sum_{i=1}^{k} \frac{E_i}{D_i} + \frac{B_k}{D_k} \le 1 \qquad (3)$$
$$k=1,..,n$$

최대 프로세서 속도에서 블로킹 구간이 있는 태스크 집합의 EDF 스케줄링은 높은 우선순위 태스크들의 이용률과 낮은 우선순위 태스크들의 블로킹 구간 중에서 가장 긴 것의 이용률 즉 B_i/D_i를 더한 것이 1 보다 작거나 같은 경우 스케줄 가능함을 보였다. 이것은 가장 긴 블로킹 구간에서도 스케줄링이 가능하다면 그 보다 작거나 같은 어떠한 블로킹 구간 구간에서도 스케줄링이 가능하기 때문이다.

[정리 2] n개의 태스크 집합이 그들의 주기에 오름차순으로 정렬되어 있다고 가정한다. 프로세서 속도 S_H가 식 (4)를 만족할 때 EDF 알고리즘으로 스케줄링 가능하다[14].

$$\forall k \quad k=1,..,n \sum_{i=1}^{k} \frac{E_i}{D_i} + \frac{B_k}{D_k} \le S_H \tag{4}$$

SRP에서는 위 식 (4)를 사용하여 임계영역이 있는 구간에서 스케줄링 가능한 프로세서 속도 S_H를 구하였다.

낮은 우선순위 태스크의 임계영역들 중 가장 긴 것을 L_i라 할 때 식 (5)를 만족하면 주어진 태스크 집합이 스케줄링 가능하다.

$$\forall k, 1 \le k \le n, \sum_{i=1}^{k} \frac{E_i}{D_i} + \frac{\max\{L_j | D_k < D_j\}}{D_k} \le S_H \tag{5}$$

즉, 우선순위가 높은 태스크들이 차지하는 이용률과 $L_j(1 \le j \le n)$의 이용률을 더한 것이 S_H보다 작거나 같다면 스케줄링 가능함을 보였다.

2. 임계영역을 고려한 동적 전압 스케줄링

2.1 이중 속도 전환 스케줄링 알고리즘

1절에서는 프로세서 이용률에 기반을 두어 설정된 스케줄링 가능한 정적속도(feasible static speed)를 적용하여 저전력 실시간 스케줄링을 논의하였다. 그러나 1절에서의 스케줄링 가능성 조건은 최악의 경우의 실행시간과 최대 블로킹 구간을 가정하였기 때문에 프로세서 속도를 더 줄일 수 있는 여지가 있다. 그래서 본 논문에서는 블로킹 구간이 없는 프로세서 이용률이 적은 구간에서 공급 전압을 낮추어 보다 효율적인 저전력 스케줄링 알고리즘에 관하여 논의한다.

만일 임계영역이 없고 태스크들이 완전히 선점 가능하다면 식 (6)과 같을 때 S_L의 속도로 EDF 알고리즘을 사용하여 스케줄링이 가능하다

$$\sum_{i=1}^{n} \frac{E_i}{P_i} \le S_L \tag{6}$$

식 (5)와 식 (6)을 사용하여 우선순위 역전이 발생하지 않는 구간에서는 S_L로 동작하는 DS 스케줄링 알고리즘이 제시되었다[14]. 낮은 우선순위 태스크의 임계영역에서의 실행으로 인해 우선순위 역전이 발생한다면 임계영역에서 실행중인 태스크의 종료시한까지 S_H로 동작한다. DS 알고리즘은 [그림 2]와 같다.

```
When job J_{i,j} arrives:
    if Priority(J_{i,j}) > Priority(current job)
        if Preempt Current_Job() is successful
            Execute J_{i,j};
        else
            Set_Speed(S_H);
            End_H = max(End_H, d_{current_job});
        end if
    end if

When the end of high speed interval is reached:
    End_H = -1;
    Set_Speed(S_L);
```

그림 3. DS 알고리즘

S_H와 S_L은 태스크의 진입 또는 종료 시에만 재계산된다. 초기 속도는 S_L로 설정되며, End_H는 S_H 구간의 끝을 나타내는 시점이다. 시스템에서 수행해야할 태스크 집합이 결정되면 S_L과 S_H를 계산한다. 스케줄러는 EDF 알고리즘에 의해 매 스케줄링 포인트 마다 모든 job들에

우선순위를 부여한다. 우선순위가 부여된 job은 동작 속도를 결정하기 위해 [그림 3]의 알고리즘을 수행한다. 알고리즘은 기본적으로 속도 S_L로 동작하지만 현재 실행중인 job보다 높은 우선순위 job이 도착했을 때 $(Priority(J_{i,j}) > Priority(d_{current\_job}))$ $J_{i,j}$에 의해 현재 job이 선점 가능 하다면 선점 후 실행을 계속 하고 만일 블록 된다면 프로세서 동작 속도를 $S_H(Set_Speed(S_H))$로 설정 후 End_H를 갱신한다. 이상의 프로시저는 O(1) 의 시간에 수행된다. S_H와 S_L은 자주 변경되지 않기 때문에 수행에 따른 오버헤드는 크지 않다.

2.2 확장 이중 속도 전환 스케줄링 알고리즘

2.1절에서는 프로세서 이용률과 블로킹 구간의 길이를 기준으로 속도 S_L, S_H를 계산한 후 가능하면 S_L로 동작하고 우선순위 역전이 발생한 구간에 한해서 S_H로 동작하는 알고리즘에 관하여 논의하였다.

본 연구에서는 임계영역에 의한 태스크 블록이 발생하였을 때 구간 결정 방식을 변경하여 S_H 구간이 좀 더 작아지도록 하였다. 블로킹 구간을 고려한 실시간 태스크 스케줄링 알고리즘은 태스크의 이용률에 따른 속도를 식 (5)와 식 (6)을 사용하여 산출한다. 산출된 속도는 스케줄링이 진행됨에 따라 각각의 상황에 맞추어 적용된다. 속도가 정해진 후 알고리즘이 시작되면 프로세서의 속도는 S_L로 정해지며 우선순위 역전 구간이 발생할 때까지 유지된다. 우선순위 역전이 발생된다면 프로세서의 속도는 S_H로 변경되고 S_H 구간의 끝을 End_H에 저장한다. 여기에서 End_H는 현재 블록된 높은 우선순위 태스크의 종료시한과 임계영역에서 실행중인 낮은 우선순위 태스크의 종료시한을 비교하여 길이가 더 작은 구간으로 결정된다. 이상의 알고리즘을 EDS (Extended Dual Speed) 알고리즘이라 한다.

알고리즘을 구체적으로 설명하기 위하여 [표 1]과 같은 태스크 집합을 가정하고 DS 알고리즘을 적용하는 경우에는 End_H의 길이는 $S_{H,1}$로 결정되고, EDS 알고리즘을 적용시키는 경우에는 End_H의 길이가 $S_{H,2}$로 결정되는 것을 [그림 4]에서 볼 수 있다.

표 1. 예제 태스크 집합

	E	P	D	B
T_1	6	20	20	12
T_2	7	30	30	
T_3	14	70	70	

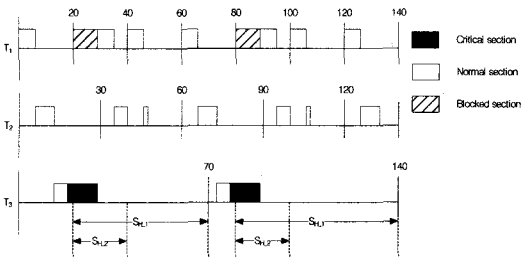

그림 4. DS와 EDS 간의 End_H 길이 비교

[그림 4]에서 태스크 T_1의 $J_{1,1}$과 태스크 T_2의 $J_{2,1}$의 실행이 종료한 후 태스크 T_3의 $J_{3,1}$이 시점 13에서 실행을 시작한다. 시점 17에서 공유자원을 사용하기 위하여 임계영역에 진입한 후 실행하는 도중 태스크 T_1의 $J_{1,2}$가 준비되어 실행을 시작 하려하지만 실행중인 태스크 T_3의 $J_{3,1}$을 SRP에 의해 선점하지 못하고 블록 된다. 이때 알고리즘은 프로세서 속도를 S_H로 설정한 후, End_H의 값을 낮은 우선순위 태스크 T_3의 $J_{3,1}$의 종료시한인 시점 70으로 결정하는 것 보다 태스크 T_1의 $J_{1,2}$의 종료시한인 시점 40으로 결정한다면 높은 속도로 동작하는 구간의 길이를 줄일 수 있다. 그러나 3장에서 제시한 문제점과 같이 기존의 End_H 길이가 더 짧을 수도 있고 이로 인해 전력 소모율은 더 나빠질 가능성이 존재한다. 이러한 문제를 해결하기 위하여 현재 블록 되는 태스크 T_i의 $J_{i,j}$의 종료시한 $d_{i,j}$를 *blocked_D*, 임계영역에서 실행중인 태스크 T_k의 $J_{k,j}$의 종료시한 $d_{k,j}$를 *current_D*라 하여 S_H 구간을 설정하기 전에 구간을 최소화 할 수 있도록 다음과 같이 비교하여 End_H를 결정하였다.

$$End\_H = max(End\_H, current\_D)$$
$$End\_H = min(End\_H, blocked\_D)$$

이상의 방법을 [그림 5]의 알고리즘으로 나타내었다.

```
When job J_ij arrives:
    if Priority(J_ij) > Priority(current job)
        if Preempt Current_Job() is successful
            Execute J_ij;
        else
            Set_Speed(S_H);
            End_H = max(End_H, d_ij);
            End_H = min(End_H, d_current);
        end if
    end if

When the end of high speed interval is reached:
    End_H = -1;
    Set_Speed(S_L);
```

그림 5. EDS 알고리즘

S_H와 S_L은 태스크의 진입 또는 종료 시에만 재계산된다. 초기 속도는 S_L로 설정되며, End_H는 S_H 구간의 끝을 나타내는 시점이다. 시스템에서 수행해야할 태스크 집합이 결정되면 S_L과 S_H를 계산한다. 스케줄러는 EDF 알고리즘에 의해 매 스케줄링 포인트마다 모든 job들에 우선순위를 부여한다. 우선순위가 부여된 job은 동작 속도를 결정하기 위해 [그림 5]의 알고리즘을 수행한다. 알고리즘은 기본적으로 속도 S_L로 동작하지만 현재 실행중인 job보다 높은 우선순위 job이 도착했을 때 (*Priority*($J_{i,j}$) > *Priority*(*dcurrent_job*)) $J_{i,j}$에 의해 현재 job이 선점 가능 하다면 선점 후 실행을 계속 하고 만일 블록 된다면 $J_{i,j}$와 End_H 값을 비교하여 가장 짧은 구간으로 End_H를 결정한다. 이상의 EDS 알고리즘을 적용하여 다음의 [표 2]와 같은 태스크 집합을 가정하고 [그림 6]과 같은 스케줄링이 가능하였다.

표 2. EDS 알고리즘 예제 태스크 집합

	E	P	D	B
T_1	6	15	15	10
T_2	7	40	40	
T_3	10	60	60	

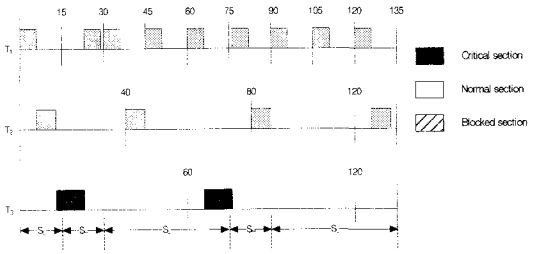

그림 6. EDS 알고리즘의 스케줄링 결과

[그림 6]에서 태스크 T_1은 시점 15와 75에서 낮은 우선순위 태스크 T_3의 임계영역에서의 실행으로 인해 블록되며 이때 속도 S_L은 S_H로 변경된다. S_H의 길이는 알고리즘에 의해 태스크 T_3의 종료시한인 시점 60과 120이 아닌 태스크 T_1의 종료시한인 시점 30과 90으로 결정되어 보다 짧은 S_H 구간이 결정됨을 알 수 있다.

이상에서 기술한 EDS 알고리즘에 대한 효율성을 입증하기 위하여 5장에서 임의의 태스크 집합들을 대상으로 모의실험을 하였으며 이를 통해 본 연구의 알고리즘으로 인한 전력 소모율이 기존 알고리즘에 비하여 효율성이 더 높다는 것을 입증하고자 한다.

V. 모의실험 및 분석

4장에서 제시한 EDS 알고리즘에 대한 저전력 효율성을 확인하기 위하여 프로그램을 작성하였다. 프로그램은 태스크 생성, 우선순위 계산, 스케줄링 정책에 따른 수행 모듈 등으로 이루어져 있다. 다음의 [그림 7]은 실험 진행 화면이다.

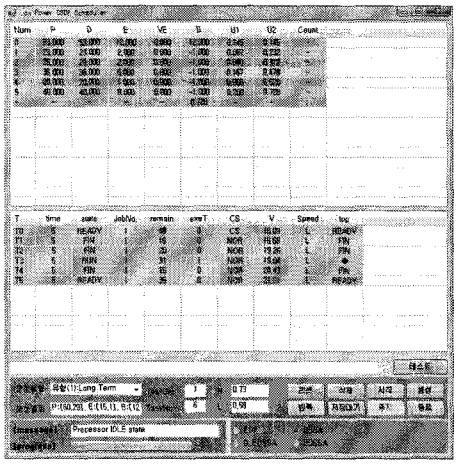

그림 7. 모의실험 프로그램 구성

공급 전압은 최대 V_{max}=1에서 최소 V_{min}=0.1이며 이는 식 (1)과 같이 공급 전압의 변화는 클록 속도의 변화로 이어진다. 모의실험을 진행하기 위하여 긴 주기, 중간 주기, 짧은 주기로 태스크 집합을 생성하였고 [표 3]와 같다.

표 3. 모의실험을 위한 태스크 집합의 생성 범주

생성 유형	주 기(P)
유형1(long)	60~20
유형2(mid)	40~20
유형3(short)	20~10

[표 3]의 범주 내에서 생성된 태스크들은 스케줄링 시작 전에 진입 테스트를 하고 전체 이용률을 넘지 않는 범위의 태스크들만 스케줄링 하였다. 각 주기는 범주 내에서 랜덤하게 결정되며 실행 시간과 블로킹 구간의 길이는 주기에 비례하여 20~30% 이내로 결정하였다.

DS와 EDS 알고리즘을 동일한 태스크 집합으로 스케줄링하여 소비되는 전력을 측정하였다. 이때 측정 시간은 각각 500sec이며 이를 각각 100회씩을 진행하였다. 각 범주별 임의의 태스크 집합이 생성된 후 스케줄러는 때 스케줄링 포인트마다 소비되는 전력량을 누적하였으며 [그림 8], [그림 9], [그림 10]은 각각 DS와 EDS 알고리즘에 의해 누적된 전력 소모량을 나타내었다. 각 실험 구간별 전력 소비 패턴을 보기 위하여 15 구간 누적 평균을 하였다.

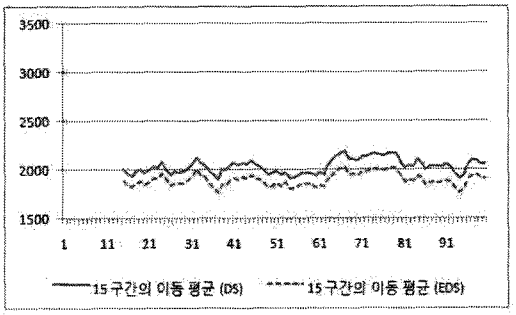

그림 8. Long term period 태스크 집합에 대한 전력 소모량

[그림 8]은 상대적으로 긴 주기로 생성된 임의의 태스크 집합에 대한 전력 소모량이다. DS 알고리즘과 EDS 알고리즘에 의한 전력 소모량을 각각 실선과 점선으로 표시하였다. 그래프에서 긴 주기 태스크 집합의 경우 최저 1700부터 최고 2000의 전력을 소모하였다. 긴 주기 태스크 집합의 경우 짧은 주기 태스크 집합 보다 프로세서 이용률이 적어 그만큼 블로킹 확률이 떨어진다. 이것은 비교적 많은 구간에서 프로세서 속도가 S_L 상태에 있었다는 것을 의미한다. 이는 다음의 결과 그래프들을 통하여 알 수 있다.

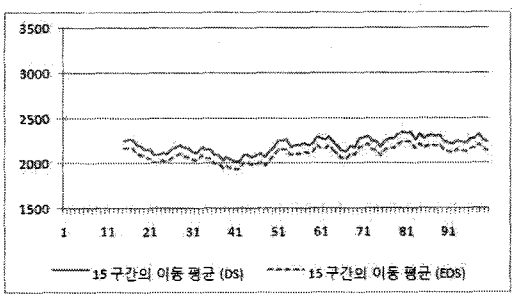

그림 9. Midterm period 태스크 집합에 대한 전력 소모량

[그림 9]는 중간 크기의 주기를 갖는 태스크 집합에 대한 전력 소모량을 나타낸다. 그래프에서 전력 소모량은 최저 1900부터 최고 2200이며 이는 긴 주기 태스크 집합

보다 좀 더 증가한 것을 알 수 있다. 이는 긴 주기 태스크 집합에 비교하여 프로세서 이용률이 증가한 결과이다. 즉, 주기의 감소로 인하여 태스크들 간에 복잡도가 증가했기 때문이다.

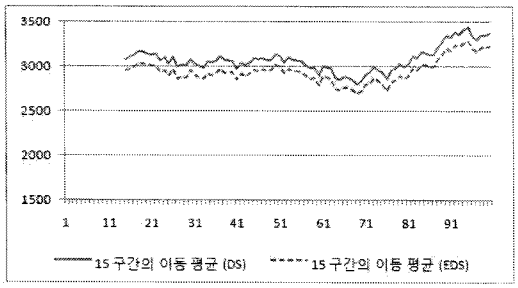

그림 10. Short term period 태스크 집합에 대한 전력 소모량

[그림 10]은 짧은 주기 태스크 집합에 대한 전력 소모량 그래프이다. 그래프에서 전력 소모량은 최저 2700부터 최고 3800이며 이는 지금까지의 전력 소모량 보다 많다. 이상의 전력 사용량 그래프들은 주기가 변화함에 따라 프로세서 이용률이 증가하였고 이는 블로킹 확률을 증가시키기 때문에 프로세서는 주로 S_H 상태로 동작하여 전력의 소비가 [그림 8]과 [그림 9]에 비해 증가한 것을 알 수 있다. 이상에서와 같이 블로킹 확률의 변동은 태스크의 소비전력과 밀접한 관계가 있으며 이를 검증하기 위하여 블로킹 구간의 변동에 따른 전력 소비율을 알아보았다.

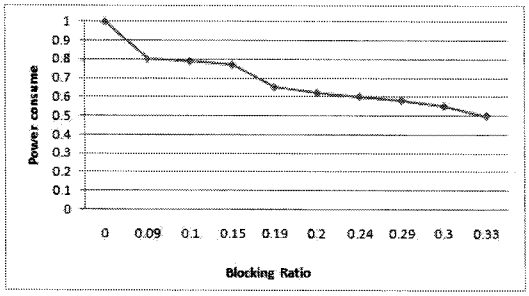

그림 11. 높은 프로세서 이용률에서 블로킹 확률과 전력 소비와의 관계

[그림 11]은 상대적으로 프로세서 이용률이 높은 환경에서 블로킹 구간의 증가에 따른 전력 소비율을 나타낸다. 그래프에서 보는바와 같이 블로킹 구간의 비율이 적을수록 전력 소비는 많으며 그렇지 않은 경우 전력 소비가 적어지는 것을 볼 수 있다. 또한 높은 프로세서 이용률은 블로킹 확률을 증가 시키며 이로 인해 프로세서는 주로 속도 S_H 로 동작할 확률이 증가한다. 때문에 [그림 11]의 전력 소비율은 프로세서 이용률이 적은 경우보다 더 증가한다. [그림 11]과 같은 조건에서는 프로세서 이용률이 상대적으로 낮은 경우 블로킹 확률이 높아지며 따라서 속도 S_L 상태로 동작할 가능성이 높다. 이는 다음의 [그림 12]에서 확인 할 수 있다.

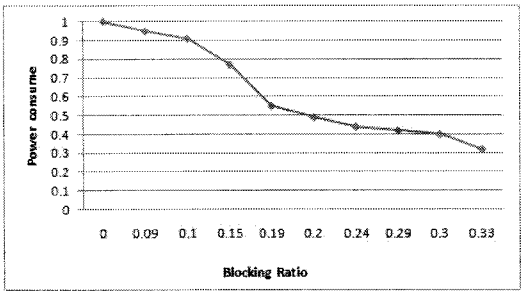

그림 12. 낮은 프로세서 이용률에서 블로킹 확률과 전력 소비와의 관계

[그림 11]에서 블록킹 확률이 0.19인 경우 전력 소비율이 0.65이나 [그림 12]에서는 0.55로 더 낮아지는 것을 볼 수 있다. 또한 [그림 11]에서 블로킹 확률이 0.33인 경우에는 전력 소비율이 0.5이지만 [그림 12]에서는 0.3으로 프로세서 이용률의 차이에 따라 소비전력이 달라진다. 결론적으로 프로세서 이용률의 증가는 블로킹 확률을 증가 시키고 블로킹 확률의 증가는 프로세서가 S_H로 동작할 확률을 높여 전력 소비를 증가시킴을 알 수 있다.

각 스케줄링 알고리즘에서 긴 주기 태스크 집합은 DS와 비교하여 감소율이 1~13% 가량의 분포를 나타내어 최대 13%의 전력 감소 효과가 나타났다. 또한 전력 소비율 그래프 [그림 8][그림 9][그림 10]에서와 같이 DS대비 EDS의 그래프가 낮은 전력 소비가 지속적으로 이루어짐을 알 수 있으며 또한 어떤 태스크 집합도 DS 스케줄링 알고리즘으로 소비한 전력보다 크지 않으므로 수정된

알고리즘에 효율성이 있다는 것을 알 수 있다. 전력의 소모율은 S_H 구간의 길이가 얼마나 길어지느냐가 결정적인 요소인데 EDS 알고리즘의 최악의 경우는 End_H의 길이가 DS 알고리즘의 End_H와 같아지는 경우이다.

VI. 결론

본 연구에서는, 기존의 DS 알고리즘을 개선한 EDS 알고리즘을 통하여 임계영역이 포함된 태스크 집합에 대한 스케줄링 알고리즘을 제시하였고 이에 대한 효율성을 입증하였다. EDS 알고리즘에서는 기존의 연구에서 임계영역으로 인해 높은 우선순위의 태스크가 블록 되는 시점에 결정되는 S_H 구간의 길이를 좀 더 작게 하는 방법이 제시되었다. 알고리즘에서는 높은 우선순위 태스크를 블록하는 낮은 우선순위 태스크의 종료시한으로 속도 S_H 길이가 결정되는 방법과 달리 우선순위 역전이 발생하는 시점에 실행중인 태스크들에 대한 종료시한을 고려하여 S_H 구간이 비교적 짧게 결정될 수 있도록 하였다. 알고리즘이 기존의 DS 알고리즘에 비하여 효율적임을 보이기 위하여 모의실험을 통해 전력 소모율을 실험 하였으며 최대 13%의 전력 감소가 있었음을 확인하였다. 본 연구의 알고리즘은 태스크내의 블로킹 구간을 고려하였으나 태스크 동기화에 대해서는 고려하지 않았다. 향후 이러한 문제의 해결과 연구가 요구된다.

참 고 문 헌

[1] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," In Proceedings of RTSS, pp.166-171, 1989.

[2] 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.46-61, 1973.

[3] W. Kim, J. Kim, and S. Min, "A Dynamic Voltage Scaling Algorithm for Dynamic- Priority Hard Real-Time Systems Using Slack Time Analysis," Proceedings of the conference on Design, p.788, 2002.

[4] Y. Shin, K. Choi, and T. Sakurai, "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors," ICCAD, p.365, 2000.

[5] A. Qadi, and S. Goddard, "A Dynamic Voltage Scaling Algorithm for Sporadic Tasks," Proceedings of the 24rd IEEE Real-Time Systems Symposium, 2003.

[6] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in Proc. IEEE Annual Foundations of Computer Science, pp.374-382, 1995.

[7] H. Aydin, R. Melhem, and D. Mosse, "Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics," Proceedings of the 13th Euromicro Conference on Real-Time Systems 2001, p.225, 2001.

[8] I. Hong, M. Potkonjak, and M. B. Srivastava, "Online Scheduling of Hard Real-Time Tasks on Variable Voltage Processor," ICCAD1998, 1998.

[9] W.C. Athas, and J. G. Koller, "An energy-efficient CMOS line driver using adiabatic switching," 1994 IEEE Great Lakes Symposium on VLSI, 1994.

[10] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen , "Low-power CMOS digital design," IEEE Journal of Solid State Circuits, 1995.

[11] K. Govil, E. Chan, and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," International Conference on Mobile Computing and Networking, 1995.

[12] M. Weiser, B. Welch, and A. Demers, S.

Shenker, "Scheduling for reduced CPU energy," in Proc. USENIX Symposium on Operating Systems Design and Implementation, pp.13-23, 1994.

[13] T. P. Baker, "Stack-base Scheduling of Real-Time Processes," The Journal of Real-Time Systems, pp.67-99, 1991.

[14] F. Zhang, and S. Chanson, "Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections," In Proceedings of RTSS'02, 2002.

[15] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols, An Approach to Real-Time Synchronization," Technical report CMU-CS-87-181, 1987.

[16] M. I. Chen, and K. J. Lin, "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems," The Journal of Real- Time Systems, Vol.2, No.4, pp.325-346, 1990.

저 자 소 개

김 남 진(Nam-Jin Kim) 정회원

- 1996년 6월 : 단국대학교 물리학과(이학사)
- 2000년 2월 : 단국대학교 전자계산학과(이학석사)
- 2003년 2월 : 단국대학교 전자계산학과(박사수료)
- 2006년 12월 : 휴대형진단치료기기 개발센터

<관심분야> : 실시간 스케줄링, 임베디드시스템,

김 인 국(In-Guk Kim) 정회원

- 1982년 : 단국대학교(학사)
- 1985년 : 미국 에모리대학교(석사)
- 1995년 : 아주대학교(박사)
- 1986년 ~ 현재 : 단국대학교 전자컴퓨터학부 컴퓨터과학전공 교수
- 2001년 ~ 2003년 : 미국 뉴멕시코 공과대학 방문교수

<관심분야> : 운영체제, 실시간시스템, 임베디드시스템