배터리 지속 시간을 보장하는 동적 작업 스케줄링 방법[1)]

김지성° 이완연
한림대학교 정보통신공학부 컴퓨터공학과
{kjs°, wanlee}@hallym.ac.kr

Dynamic Task Scheduling Mechanism Guaranteeing the Residual Time of Battery

Ji-Sung Kim° Wan Yeon Lee
Dept. of Computer Science Engineering, DIET, Hallym University

요 약

배터리를 효율적으로 사용하기 위한 기존 연구에서 제한된 용량의 배터리를 최대한 연장하여 사용하는 방법들에 관하여 다루었다. 본 논문에서는 시스템을 사용하는 정해진 시간동안 배터리가 소진되지 않도록, 시스템에서 동작하는 작업량을 동적으로 조절하는 방법을 제시한다. 제시된 방법에서는 시스템에서 동작하는 작업들을 중요성에 따라 분류하고, 시스템을 연동시키는 배터리 잔량을 주기적으로 측정한다. 그리고 측정된 배터리 잔량이 충분하면 모든 작업들을 동작시키고 배터리 잔량이 부족하면 중요성이 떨어지는 작업들부터 동작을 정지시켜서 전체 작업량을 감소시키고, 이를 통하여 배터리 지속 시간을 점진적으로 연장시키는 방법이다.

1. 서 론

최근 들어 내장형 시스템(Embedded System)은 이동기기나 로봇 등의 분야로 활용 분야가 갈수록 다양해지고 있다. 이러한 내장형 시스템은 대부분 무선 환경에서 배터리와 연동하여 동작하기 때문에 제한된 용량의 배터리를 오랜 시간동안 최대한 연장하여 사용하는 방법들에 관한 연구를 기존에 많이 다루었다. 이와 관련된 기존 연구로는 동적으로 프로세서에 제공되는 전압을 낮추는 DVS(Dynamic Voltage Scaling) 방법[1]과 주변 장치에 대한 전력을 동적으로 관리하는 DPM(Dynamic Power Management) 방법[2][3] 등이 있다. 또한 작업별로 소모되는 전력을 비교 분석한 연구[4]와 배터리의 잔량을 측정하고 배터리 특성에 맞게 동적으로 CPU의 전압을 조절하는 연구[5]가 있었다. 그러나 제한된 용량의 배터리가 일정 기간 이상 지속될 수 있도록 시스템에서 동작하는 작업량을 동적으로 관리하는 연구는 다루어지지 않았다. 예를 들어 영상 수집 기능과 이동 기능, 그리고 위치 알림 기능을 수행하는 무선 탐사 로봇을 사용할 때, 동작 중에 제한된 용량의 배터리가 소진되면 무선 탐사 로봇을 분실할 수 있다. 이처럼 사용 중에 배터리가 소진되어 발생하는 문제점을 방지하기 위하여 배터리 잔량이 충분하면 영상 수집 기능, 이동 기능, 위치 알림 기능을 모두 수행하고, 배터리 잔량이 부족하면 위치 알림 기능만을 수행하도록 동적으로 작업량을 조절하는 방법을 필요하다. 배터리 잔량에 따라 동적으로 작업량을 조절하는 방법을 사용하면 일정 기간 동안 무선 탐사 로봇이 최소한의 동작을 수행하는 것을 보장할 수 있게 된다.

본 논문에서 제안하는 배터리 지속 시간을 보장하는 동적 작업 스케줄링 방법은 시스템에서 동작하는 작업들을 선택(Optional) 작업, 기본(Mandatory) 작업, 필수(Critical) 작업 세 가지로 분류한다. 그리고 배터리 잔량을 주기적으로 측정하여 작업량 1 단계인 필수 작업만을 수행할 경우의 배터리 지속 시간, 작업량 2 단계인 기본 작업과 필수 작업만을 수행할 경우의 배터리 지속 시간, 그리고 작업량 3 단계인 모든 작업들을 수행할 경우의 배터리 지속 시간을 예측한다. 예측된 배터리 지속 시간이 시스템을 사용하고자 하는 시간보다 짧으면 시스템에서 동작하는 작업량 단계를 낮추어 시스템의 전체 작업량을 줄이고, 예측된 배터리 지속 시간이 시스템을 사용하고자 하는 시간보다 현저히 길면 작업량 단계를 올려서 시스템의 전체 작업량을 증가시킨다.

논문의 구성은 다음과 같다. 2 장에서는 배터리 잔량을 측정하는 방법과 배터리 지속 시간을 예측하는 방법을 설명하고, 3 장에서는 도출된 배터리 지속 시간을 이용하여 시스템의 작업량을 동적으로 변경하는 방법을 설명한다. 그리고 4장에는 결론 및 연구 방향을 제시한다.

2. 기반지식

배터리 잔량을 측정하는 방법은 System Technology Buckhyae 사의 Smart Battery Chip인 STB-A 칩에서 제공하는 방법을 사용한다. 이 칩에서는 배터리의 현재 사용 중인 전압(volt), 배터리 잔량(BSOC), 사용 가능시간, 전류(mA)를 I/O 포트를 통해 프로세서에게 전송할 수 있다. 구현 대상 시스템 보드의 GPIO 한 포트를 사용해서 배터리 칩과 연결하고, 정보가 필요할 때마다 칩에 요청 시그널(Request Signal)을 보내면, 칩에서는 배터리의 현재 사용 중인 전압(1 바이트), 배터리 잔량(1 바이트), 사용 가능시간(2 바이트), 전류(2 바이트)를 응답 시그널에 저장하여 시스템 보드에 전송한다. 응답된 정보 중에서 배터리 잔량이 프로세서에게 전달되어 작업 스케줄링에 활용된다.

측정된 배터리 잔량을 이용하여 배터리 지속 시간을 예측하는 방법은 다음과 같다. 다양한 배터리 잔량을 기준 값으로 삼고, 시스템에서 동작하는 작업량 단계별로 배터리 지속 시간은 충분한 실험을 통하여 측정한다. 그리고 측정된 실험값들 중에서 최악의 배터리 지속 시간을 선택하여 시스템에 사전에 입력하고 BTable이라는 테이블 형태로 관리하다. BTable에 저장된 배터리 잔량 값들 중에서 측정된 배터리 잔량에 가장 근접하는

배터리 잔량 값을 선택하여 이에 상응하는 작업량 단계별 배터리 지속 시간 실험 값을 제시하는 동적 작업 스케줄링 방법에 사용한다. 표 1은 다양한 배터리 잔량 값을 기준으로 시스템에서 동작하는 작업량에 따라 배터리가 지속되는 시간을 BTable이라는 표 형태로 시스템에서 관리하는 예를 보여주고 있다.

배터리 잔량	작업량 단계별 배터리 지속 시간(초)		
	1 단계	2 단계	3 단계
1210mA	20분 02초	16분 42초	15분 02초
1105mA	19분 08초	15분 48초	14분 11초
.	.	.	.
.	.	.	.
397mA	13분 18초	9분 57초	8분 23초

표 1. BTable: 배터리 지속 시간 실험 측정 정보

3. 제안된 작업 스케줄링 방법

제안된 방법의 목적은 시스템이 배터리 소진 현상 없이 일정 기간 동안 동작할 수 있도록 시스템에서 동작하는 작업량을 조절하는 것이다. 이를 위하여 시스템에서 동작하는 작업들의 우선순위를 선택 작업(Optional Task), 기본 작업(Mandatory Task), 필수 작업(Critical Task)의 3 단계로 분류한다. 그리고 분류된 작업들을 기반으로 배터리 잔량을 측정하여 배터리 잔량에 따라 시스템에서 동작하는 작업량을 조절한다. 시스템에서 동작시키는 작업량은 3 단계로 분류하며, 배터리 잔량이 충분하면 선택 작업, 기본 작업, 필수 작업들을 모두 동작시키는 3 단계, 배터리 잔량이 3 단계 작업을 일정기간 동안 실행하기에 부족하면 기본 작업과 필수 작업만을 동작시키는 2 단계, 배터리 잔량이 2 단계 작업을 일정기간 동안 실행하기에 부족하면 필수 작업만을 동작시키는 1 단계로 구분한다.

3.1 기본 알고리즘

제안된 방법을 구현하기 위해서는 시스템의 사용 중에 주기적으로 배터리 잔량을 측정하는 기능, 측정된 배터리 잔량을 기준으로 작업량 단계별 배터리 지속 시간을 예측하는 분석 기능, 예측된 배터리 지속 시간에 따라 시스템에서 동작하는 작업량을 변화시키는 작업 스케줄링 기능이 필요하다. 배터리 잔량을 측정하는 기능은 2 장에서 설명하였듯이 Smart Battery Chip을 사용하여 이 기능을 전담하도록 한다. 그리고 측정된 배터리 잔량을 기준으로 작업량 단계별 배터리 지속 시간을 예측하는 분석 기능은 그림 1의 Analyzer() 함수가 수행하도록 하였고, 예측된 배터리 지속 시간에 따라 시스템에서 동작하는 작업량을 변화시키는 작업 스케줄링 기능은 그림 1의 TaskScheduler() 함수가 수행하도록 설계하였다. 이 함수들에서 사용되는 수식은 다음과 같은 의미를 가진다.

- α :시스템에서 현재 동작중인 작업량 단계
- β :현재 시점에서 측정된 배터리 잔량
- gt :배터리가 소진 현상 없이 시스템의 동작이 보장되어야 하는 최소 시간
- ct :시스템이 동작을 시작한 이후 현재 시간
- et :Analyzer() 함수를 통해서 계산된 배터리 지속시간
- I_t :작업 스케줄링을 동작시키는 시간 주기
- ϕ :BTable에서 β보다 작은 배터리 잔량 중 최대값
- ϕ_0 :BTable에서 β보다 큰 배터리 잔량 중 최소값
- τ :BTable에서 ϕ 값을 기준으로 검색된 α작업량 단계에서 배터리 지속 시간
- τ_0 :BTable에서 ϕ_0 값을 기준으로 검색된 α의 작업량 단계에서 배터리 지속시간

커널에서 함수 Task Scheduler()에게 I_t의 시간마다 작업량 단계(α)와 시스템에서 보장되어야 하는 최소 시간(gt)을 전달하여 호출하고, 전달한 α와 반환된 α를 비교하여 같지 않다면 현재의 작업량 단계를 변경한다. Task Scheduler()는 배터리 전담 칩에서 측정된 배터리 잔량 값(β)과 현재 시스템에서 동작되고 있는 작업량 단계(α)를 함수 Analyzer()에게 전달하고, 함수 Analyzer()로부터 예측된 배터리 지속 시간 값(τ)이 시스템의 최소 동작 시간(gt)보다 작으면 작업량 단계를 점진적으로 감소시키고, 배터리 지속 시간이 시스템의 최소 동작 시간보다 크면 작업량 단계를 점진적으로 증가시킨다.

```
1 : Function TaskScheduler( α gt )
2 :     β= 배터리 전담 칩에 전달된 배터리 잔량 값;
3 :     ct = 시스템이 동작을 시작한 이후 현재 시간;
4 :     et = Analyzer( β, α);
5 :     if( gt - ct > et )
6 :        while( gt - ct > et )
7 :           α= α- 1;
8:            et = Analyzer( β, α);
9:        endwhile
10:    else if( gt - ct ≤ et )
11:       while( gt - ct ≤ et )
12:          α= α+ 1;
13:          et = Analyzer( β, α);
14:       endwhile
15:    endif
16:    return α
17: End-TaskScheduler

18 : Function Analyzer( β, α)
19 :     φ = BTable에서 β보다 작은 배터리 잔량 중
             최대값;
20 :     τ = BTable에서 φ를 기준으로 검색된 α작업량
             단계의 배터리 지속시간;
21 :     return τ;
22 : End-Analyzer
```

그림 1. 작업 스케줄링 알고리즘

그림 1의 알고리즘은 다음과 같이 동작한다. 2번 줄에서 현재 남은 배터리의 잔량을 확인하고, 4번 줄에서는 배터리 잔량과 현재의 시스템에서 동작하는 작업량 단계를 Analyzer() 함수에 넘겨서 배터리 지속시간을 예측한다. 그리고 6-9번 줄들에서 예상 배터리 지속시간이 앞으로 동작을 보장해야할 시간

보다 작다면 작업량 단계를 한 단계 감소시킨다. 그리고 감소된 작업량 단계에서도 예상 배터리 지속시간이 앞으로 동작을 보장해야할 시간보다 여전히 적을 수도 있기 때문에, 예상 배터리 지속시간이 앞으로 동작을 보장해야할 시간보다 커질 때까지 작업량 단계를 점진적으로 감소시키는 과정을 반복한다. 또한 현재의 배터리 지속 시간이 시스템의 잔여 동작 시간을 보장한다면, 11-14번 줄들에서 작업량 단계를 한 단계 높여서도 배터리 지속 시간이 시스템의 잔여 동작 시간을 보장하는지 확인하고, 보장한다면 작업량 단계를 점진적으로 증가시키는 과정을 반복한다. 18번째 줄부터의 Analyzer() 함수는 배터리 잔량(β)과 현재 작업량 단계(α)를 전달 받는다. BTable의 배터리 잔량에서 다음의 관계가 $\phi < \beta < \phi_1$ 성립한다면, ϕ에 부합하는 작업량 단계별 배터리 지속 시간 중에서 α에 해당하는 배터리 지속시간을 반환하게 된다.

3.2 배터리 지속 시간 예측 오차 보정 방법

3.1장에서 사용된 Analyzer() 함수는 BTable에서 β보다 작은 배터리 잔량 중 최소값(ϕ)을 반환하기 때문에 입력된 배터리 지속시간과 BTable에 저장되어 있는 배터리 잔량값들과 정확하게 일치하지 않는다. 따라서 실제 배터리 지속시간과 오차가 발생할 소지가 있다. 예를 들어 스케줄링 시점의 배터리 잔량이 1130mA이고 작업량 단계가 1이라고 했을 때, 표 2와 같은 BTable이 존재한다면 기존의 Analyzer() 함수에서는 1105mA에 해당하는 배터리 지속 시간을 예측하기 때문에 1130mA - 1105mA = 25mA에 상응하는 잔여시간의 오차가 발생한다.

배터리 잔량	작업량 단계별 배터리 지속 시간(초)		
	단계 1	단계 2	단계 3
1210mA	20분 02초	16분 42초	15분 02초
1105mA	19분 08초	15분 48초	14분 11초

표 2. 배터리 지속 예측 시간

따라서 다음과 같은 방법으로 배터리 지속 시간의 오차를 줄인다. 먼저 배터리가 1105mA 근처에서 1mA가 감소하는 시간 비율을 다음과 같이 계산한다. 1mA가 감소하는 시간 = 1200sec - 1150sec)/(1220mA - 1090mA) = 0.514 sec/mA. 그리고 계산된 시간 비율을 이용하여 배터리 잔량이 1130mA인 경우의 배터리 지속 시간은 다음과 같이 구한다. 1130mA에서의 배터리 지속시간 = 1105mA에서 배터리 지속시간 + (1130mA - 1105mA) * 1mA가 감소하는 시간(0.514초) = 19분 08초 + 12.85초 = 19분 20.85초. 이와 같은 오차 보정 방법을 사용하여 3.1장의 Analyzer() 함수를 개선하여 배터리 지속 시간을 예측하는 새로운 Analyzer() 함수를 그림 2에서 기술하였다.

그림 2의 3번과 5번 줄들에서 ϕ의 배터리 지속시간(τ)과 ϕ_1의 배터리 지속시간(τ_1)을 구한 후, 6번째 줄에서 배터리 지속시간의 하한값에 1mA가 감소하는데 걸리는 예측시간에 남은 배터리의 잔량과 BTable에서 찾은 배터리 잔량의 차를 곱하여 오차값을 보정한 배터리 지속시간이 구해진다. 구해진 배터리 지속시간에 ϕ 을 합하여 예측된 배터리 지속시간을 반환한다.

```
1: Function Analyzer( β α)
2:    φ = BTable에서 β보다 작은 배터리 잔량중 최대값;
3:    τ = BTable에서 φ를 기준으로 검색된 α의 작업량
          단계에서 배터리 지속시간;
4:    φ1 = BTable에서 β보다 큰 배터리 잔량중 최소값;
5:    τ1 = BTable에서 φ1를 기준으로 검색된 α의 작업량
          단계에서 배터리 지속시간;
6 :  return  φ + ( β- φ) * ( τ1 - τ) / ( φ1 - φ);
7 : End-Analyzer
```

그림 2. 개선된 Analyzer() 함수

4. 결 론

본 논문에서는 시스템에서 동작하는 작업들에게 우선 순위를 부여하고, 배터리 잔량을 측정하여 배터리 지속시간을 예측하고, 예측된 배터리 지속 시간에 근거하여 시스템의 잔여 운영 시간을 보장하도록 시스템에서 동작하는 전체 작업량을 조절하는 방법을 제시하였다. 제시된 작업 스케줄링 방법은 실제 임베디드 시스템으로 구현하고 있다. 구현중인 시스템에서는 Hybus사의 Hyper-255B를 메인 보드로 사용하고, 이 보드에 Omi-Web 카메라, Buffalo 802.11b 무선랜 카드와 DC 모터를 연결하여 사용한다. 운영체제로는 ARM용으로 패치된 리눅스 2.4.1 커널을 적재하고, 시스템에서 동작하는 작업들로는 현재 위치 알림 기능, 영상 획득 기능, 이동 기능을 세 단계의 우선순위로 분류된 실행시킨다. 또한 System Technology Byuckhae사(www.stbchip.co.kr)의 STB-A 칩을 사용하여 배터리 잔량을 측정하며 배터리 지속 예측 시간을 기반으로 시스템의 전체 작업량을 동적으로 변경한다.

5. 참조문헌

[1] K. Govil, E. Chan, H. Wasserman,"Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," International Conference on Mobile Computing and Networking, pp. 13-25, Nov. 1955.

[2] Benini, et. al., "A Survey of Design Techniques for System-Level Dynamic Power Management", IEEE Transactions on VLSI, pp. 219-316, Jun. 2000.

[3] L. Benini, G. Castelli, A. Macii, R. Scarsi, "Battery-Driven Dynamic Power Management", IEEE Design & Test of Computers, vol. 18, no. 2, pp. 53-60, March-April 2001.

[4] 강경태, 심호준, 박성수, 성민영, 신현식, 장래혁, "uC/OS-II 운영체제의 부시스템별 CPU전력 소비 분석", pp. 94-96, 정보과학회 추계학술대회, 2001년 10월.

[5] 최석원, 차호정, "배터리 잔량에 기반한 동적 전력 관리 기법", pp. 106-108 , 정보과학회 춘계학술대회, 2004년 4월.