논문 2011-06-06

가변 실행 시간 태스크들을 위한 개선된 Pfair 스케줄링 알고리즘

(An Improved Pfair Scheduling Algorithm for Tasks with Variable Execution Times)

박 현 선, 김 인 국*
(Hyun-Sun Park, In-Guk Kim)

Abstract : The Pfair scheduling algorithm, which is an optimal scheduling algorithm in the hard real-time multiprocessor environments, propose the necessary and sufficient condition for the schedulability and is based on the fixed quantum size. Recently, several methods that determine the optimal quantum size dynamically were proposed in the mode change environments. But these methods considered only the case in which the period of a task is increased or decreased. In this paper, we also consider the case in which the execution time of a task is increased or decreased, and propose new methods that determine the optimal quantum size dynamically.

Keywords : Real-time scheduling, Multiprocessor scheduling, Pfair scheduling

Ⅰ. 서 론

범용 컴퓨팅 시스템에서의 각 태스크는 그 것에 주어진 논리적인 흐름에 따라 올바른 결과를 산출해 내는 것을 목표로 한다. 이에 비해 실시간 시스템은 시스템의 정확성과 안전성을 보장하기 위해 반드시 충족되어야 하는 명백한 시간적 요구사항에 중요성을 둔다. 각 태스크에 대한 시간적 요구사항을 종료시한(deadline)이라 부르며 만약 주어진 종료시한이내에 해당 태스크를 처리하지 못하는 경우에는 시스템에 심각한 문제가 초래될 수도 있다.

실시간 시스템에서 각 태스크의 종료시한을 지키는 일은 운영체제가 예측가능성을 가지고 동작할 때 보장될 수 있으며 특히 태스크들을 실시간 스케줄링하는 것이 매우 중요한 일이 된다. 단일 프로세서 환경에서는 Liu와 Layland가 제안한 Rate-Monotonic Scheduling(RMS) 알고리즘과 Earliest Deadline Scheduling(EDS) 알고리즘이 최적의 알고리즘으로 증명되었다 [4]. RMS 알고리즘은 각 태스크에게 정적(static)으로 우선순위가 부여되는 우선순위 기반 알고리즘으로서 주기가 짧을수록 더 높은 우선순위가 부여된다. EDS 알고리즘은 동적(dynamic)으로 우선순위가 부여되는 우선순위 기반 알고리즘으로서 스케줄링 시점에서 볼 때 종료시한까지의 남은 시간이 짧은 태스크일수록 더 높은 우선순위가 부여된다. 그러나 RMS 알고리즘이나 EDS 알고리즘은 다중프로세서 환경에서는 최적의 알고리즘이 되지 못한다.

Baruah 등은 다중 프로세서 환경에서 주기적 경성 실시간 태스크들에 대한 최적의 스케줄링 알고리즘인 Pfair 스케줄링 알고리즘 [12]과 이것을 개선한 PD 알고리즘 [11]을 제안하였다. Pfair 스케줄링 알고리즘에서 각 태스크는 보다 작은 실행 단위인 퀀텀 단위로 분할되어 스케줄링된다.

이때 하나의 태스크를 구성하는 여러 퀀텀들을 해당 태스크의 서브태스크(subtask)라 부른다. Pfair 스케줄링 알고리즘에서 서브태스크들은 Earliest-Pseudo-Deadline-First(이하 EPDF)를 기반으로 우선순위가 결정되며, 둘 이상의 서브태스크가 동일한 의사 종료시한을 가질 경우에는 몇 가지

* 교신저자(Corresponding Author)
논문접수 : 2010. 12. 10., 수정일 : 2011. 01. 06.,
채택확정 : 2011. 01. 12.
박현선 : 단국대학교 대학원 전자계산학과
김인국 : 단국대학교 컴퓨터과학과 교수
※ 이 연구는 2009년도 단국대학교 대학연구비의 지원으로 연구되었음.

규칙을 사용하여 최종 우선순위를 결정한다.

Baruah 등은 다중 프로세서 환경에서 프로세서의 수가 M일 때 태스크 집합이 스케줄링 가능하기 위한 필요충분조건이 다음과 같음을 증명하였다. 식 (1)에서 $T.e$와 $T.p$는 태스크 집합 τ에 속한 태스크 T의 실행요구 시간과 주기를 각각 나타낸다.

$$\sum_{T\in\tau}\frac{T.e}{T.p}\leqq M \tag{1}$$

Baruah 등이 Pfair 스케줄링 알고리즘을 제안한 이후 Pfair를 기반으로 하는 다수의 알고리즘들이 제안되었는데, 이러한 퀀텀 기반의 스케줄링 알고리즘들은 각 태스크의 주기 및 실행 요구 시간이 퀀텀 크기의 배수가 됨을 가정하고 있다 [5-9]. 이는 주어진 태스크 집합에 변경이 가해지지 않는 환경에서는 유효하나, 태스크 집합의 대주기에서 주어진 태스크 집합이 새로운 태스크 집합으로 변경(mode change)되는 경우 [1]나 동적 태스크 집합의 경우에는 변경되거나 새로이 삽입된 태스크들의 실행요구 시간 및 주기에 맞추어 퀀텀 크기의 변경이 요구될 수 있다.

일반적으로 태스크를 구성하는 퀀텀들의 크기가 증가하게 되면 문맥교환(context switching)을 하기 위한 시간이나 캐시 재적재 시간 등이 감소하게 되고 이는 시스템 성능이 향상되는 요인이 된다. 따라서 주어진 태스크 집합에 대한 최적의 퀀텀은 스케줄링이 실패하지 않는 한도 내에서 그 크기를 최대로 결정하여 구할 수 있다.

최근 mode change 환경에서 최적의 퀀텀 크기를 결정하는 스케줄링 알고리즘이 제안된 바 있다 [2,3]. 제안된 방법들에서는 태스크의 주기가 증가되거나 감소되는 경우를 다루고 있으며 태스크의 실행요구 시간은 감소될 수 없다고 가정하고 있다. 그러나 일반적인 태스크 모델에서 태스크의 실행요구 시간은 가능한 최소값과 최대값의 구간으로 주어지기 때문에 각 태스크의 실행요구 시간은 어떤 상황에서는 증가될 수도 있으며 또 다른 상황에서는 감소될 수도 있다 [10]. 이에 본 논문에서는 태스크의 실행요구 시간이 증가되는 경우뿐만 아니라 감소되는 경우까지도 고려하여, 보다 일반적인 태스크 모델의 최적 퀀텀 크기를 결정하기 위한 개선된 알고리즘을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 1장에서는 서론 및 관련연구를 기술하였고, 2장에서는 스케줄링 환경 및 태스크 모델이 기술되며, 3장에서는 나누어진 스케줄링 모델 각각에 대해 최적의 퀀텀 크기를 결정하기 위한 논리적인 근거와 스케줄링 알고리즘이 기술된다. 끝으로 4장에서는 결론과 향후 연구 전망 등이 기술된다.

Ⅱ. 스케줄링 모델

본 논문에서 제안하는 스케줄링 방법은 전역(global) 스케줄링을 기반으로 하고 있다. 따라서 태스크들은 모든 프로세서들이 공동으로 사용하는 단일 준비 큐에 저장된다. 스케줄링 시점이 되면 단일 준비 큐에서 가장 높은 우선순위 태스크가 선택되어 실행된다. 따라서 태스크들은 프로세서 간 이동이 허용된다.

주어진 태스크 집합은 구 스케줄(old schedule)의 대주기(major period) 끝에서 새로운 태스크 집합으로 변경됨을 가정한다. 대주기는 태스크들의 주기의 최소공배수로서 모든 태스크들은 대주기의 시작점에서 동시에 실행 준비가 됨을 가정한다.

태스크들은 실행요구 시간 및 주기의 변동 가능성을 가진다. 즉 태스크들은 대주기의 끝에서 mode change된 후 실행요구 시간이 이전보다 감소되거나 증가될 수 있으며 이에 따라 주기도 이전보다 감소되거나 증가될 수 있다.

각 태스크 T는 주기 p와 실행요구 시간 e를 속성으로 갖는다. $q_{\min}$은 시스템에서 표현할 수 있는 최소 퀀텀 크기로서 단위 시간 1을 의미한다. q_c는 현재 사용 중인 퀀텀 크기이고, q_n은 새롭게 선택되는 퀀텀의 크기로서 q_n이 결정되면 $q_c=q_n$이 된다.

e_c와 p_c는 q_c에 비례해서, 그리고 e_n과 p_n은 q_n에 비례해서 정의되는 실행요구 시간과 주기를 각각 나타낸다. 이때, 퀀텀 크기의 변경에 따라 태스크의 실행요구 시간과 주기는 증가되거나 감소될 수 있으므로 e_n과 p_n은 다음과 같이 표시될 수 있다.

$$e_n=\left\lceil\frac{e}{q_n}\right\rceil \text{ or } \left\lfloor\frac{e}{q_n}\right\rfloor \tag{2}$$

$$p_n=\begin{cases}\left\lfloor\frac{p}{q_n}\right\rfloor \text{ or } \left\lceil\frac{p}{q_n}\right\rceil & (\text{if}\quad q_n<p)\\ 1 & (\text{if}\quad q_n\geq p)\end{cases} \tag{3}$$

따라서 w를 $q_{\min}$에 의해 계산된 프로세서 이용률(utilization)이라 하면, q_n에 의해 변경된 태스크의 이용률 w'은 다음과 같이 표시된다.

$$w' = \frac{e_n}{p_n} = \begin{cases} \dfrac{\left\lceil \frac{e}{q_n} \right\rceil}{\left\lfloor \frac{p}{q_n} \right\rfloor} \text{ or } \dfrac{\left\lceil \frac{e}{q_n} \right\rceil}{\left\lceil \frac{p}{q_n} \right\rceil} \text{ or } \\ \dfrac{\left\lfloor \frac{e}{q_n} \right\rfloor}{\left\lfloor \frac{p}{q_n} \right\rfloor} \text{ or } \dfrac{\left\lfloor \frac{e}{q_n} \right\rfloor}{\left\lceil \frac{p}{q_n} \right\rceil} & (\text{if } q_n < p) \\ 1 & (\text{if } q_n \geqq p) \end{cases} \quad (4)$$

그러므로 $T{\cdot}w'$를 태스크 집합 τ에 속한 각 태스크 T의 이용률이라 하면 τ에 속한 모든 태스크들의 이용률의 합 U는 다음과 같이 구할 수 있다.

$$U = \sum_{T \in \tau} T{\cdot}w' \quad (5)$$

본 논문의 나머지 부분에서는 퀀텀 크기를 나타내는 q_c와 q_n대신 기호 Q를 사용하기로 한다. 또한 $p_{\max}$와 $p_{\min}$은 태스크들의 주기의 최대값과 최소값을 각각 나타낸다.

Ⅲ. 최적 퀀텀 크기 결정

1. 도달점

퀀텀 크기가 증가할 때 태스크의 이용률은 단조 증가하지는 않는다. 그러나 퀀텀 크기 Q를 계속 증가시키면 어느 시점부터는 태스크의 이용률이 계속 1의 값을 유지하게 되는데, 이렇게 태스크의 이용률이 계속 1이 되는 최초의 퀀텀 크기 Q를 실제 도달점 *RRP*라 하고, 실제 도달점을 포함한 이후의 Q 값들을 도달 구간이라 부른다.

태스크의 실제 도달점을 구하기 위해서는 $p_{\max}-1$로부터 Q값을 1만큼씩 감소시키면서 해당 태스크의 이용률이 1보다 작은 값이 나오는 지를 반복적으로 찾아야 하는데, 이렇게 태스크들의 실제 도달점을 구하는 것은 $O(p_{\max})$의 시간복잡도를 갖는다. $P_{\max}$의 값이 $n!$이 된다면 이 방법은 지수 시간복잡도를 갖게 된다. 본 논문에서는 태스크들의 실행요구 시간과 주기가 변경되는 경우에 최적의 퀀텀 크기를 구하기 위한 새로운 알고리즘을 제안하고자 한다.

[2]에서는 실행요구 시간이 감소하지 않고 주기가 증가하지 않는 태스크의 도달점을 상수 시간에 구할 수 있음을 다음 정리와 같이 증명하였다.

정리 1 : 실행요구 시간이 감소하지 않고 주기가 증가하지 않는 태스크의 도달 함수 $Reach(p,\ e)$는 다음과 같이 정의한다.

$$Reach(p,\ e) = \begin{cases} \left\lfloor \frac{p}{2} \right\rfloor + 1 \left(\text{if } \frac{e}{p} \leqq \frac{1}{2} \right) \\ \left\lfloor \frac{p}{3} \right\rfloor + 1 \left(\text{if } \frac{e}{p} > \frac{1}{2} \right) \end{cases} \quad (6)$$

이때, $Q \geqq Reach(p,\ e)$이면

임의의 p와 e에 대해 $w' = \dfrac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lceil \frac{p}{Q} \right\rceil} \geqq 1$이다.

[3]에서는 실행요구 시간이 감소하지 않고 주기도 감소하지 않는 태스크의 도달점을 상수 시간에 구할 수 있음을 다음 정리와 같이 증명하였다.

정리 2 : 실행요구 시간이 감소하지 않고 주기도 감소하지 않는 태스크의 도달 함수 $Reach(p,\ e)$는 다음과 같이 정의한다.

$$Reach(p,\ e) = p \quad (7)$$

이때, $Q \geqq Reach(p,\ e)$이면

임의의 p와 e에 대해 $w' = \dfrac{\left\lceil \frac{e}{Q} \right\rceil}{\left\lceil \frac{p}{Q} \right\rceil} \geqq 1$이다.

본 논문에서는 남은 두 가지 경우에 대해 태스크의 도달점을 상수 시간에 구할 수 있음을 다음의 정리와 같이 제안하고 증명한다.

정리 3 : 실행요구 시간이 증가하지 않고 주기도 증가하지 않는 태스크의 도달 함수 $Reach(p,\ e)$는 다음과 같이 정의한다.

$$Reach(p,\ e) = p \qquad (8)$$

이때, $Q \geq Reach(p,\ e)$이면

임의의 p와 e에 대해 $w' = \dfrac{\lfloor \frac{e}{Q} \rfloor}{\lfloor \frac{p}{Q} \rfloor} \geq 1$이다.

증명 : $Q \geq Reach(p,e)$이면 $Q \geq p$이다. 이때 $\lfloor \frac{e}{Q} \rfloor$와 $\lfloor \frac{p}{Q} \rfloor$는 최소값을 갖는 경우이고 각각 1이다. 따라서 $w' = \frac{1}{1} \geq 1$이다. ■

정리 4 : 실행요구 시간이 증가하지 않고 주기가 감소하지 않는 태스크의 도달 함수 $Reach(p,\ e)$는 다음과 같이 정의한다.

$$Reach(p,\ e) = p \qquad (9)$$

이때, $Q \geq Reach(p,\ e)$이면

임의의 p와 e에 대해 $w' = \dfrac{\lfloor \frac{e}{Q} \rfloor}{\lceil \frac{p}{Q} \rceil} \geq 1$이다.

증명 : $Q \geq Reach(p,e)$이면 $Q \geq p$이다. 이때 $\lfloor \frac{e}{Q} \rfloor$는 최소값을 갖는 경우이어서 1이고 $\lceil \frac{p}{Q} \rceil$는 1이므로 $w' = \frac{1}{1} \geq 1$이다. ■

2. 예제

$\tau = \left\{ T_1 = \frac{7}{41}, T_2 = \frac{2}{4}, T_3 = \frac{29}{48}, T_4 = \frac{8}{39}, T_5 = \frac{15}{22} \right\}$가 주어진 태스크 집합이고 프로세서의 수가 M이라 가정하자. τ는 다섯 개 태스크들의 집합이며 T_1은 실행요구 시간이 7이고 주기가 41인 태스크이다.

M개의 태스크들이 이미 그들의 도달 구간에 있다면, 그들의 이용률의 합은 M이 되고 나머지 태스크들의 이용률의 합은 0보다 크므로 스케줄링은 불가능하게 된다. 따라서 최적의 Q값은 적어도 $M-1$번째 태스크 또는 그 이전에 있는 태스크의 도달점에서 구할 수 있게 된다. 이를 위해 각 태스크의 도달점을 오름차순으로 저장하는 도달 순위 리스트 $Rank[n]$을 생성한다. 또한 실제 도달점보다 작은 값으로부터 Q값을 찾아야 하므로 도달 순위 리스트의 각 원소에는 해당 태스크의 $Reach(p,e)-1$의 값이 저장된다.

식 (4)는 태스크의 실행요구 시간과 주기가 mode change 이후에 변경되는 네 가지 경우에 태스크의 이용률을 어떻게 계산해야 하는 지를 보여준다. 태스크가 변경될 수 있는 네 가지 경우는 다음과 같다.

- 실행요구 시간이 감소하지 않고 주기가 증가하지 않는 경우
- 실행요구 시간이 감소하지 않고 주기도 감소하지 않는 경우
- 실행요구 시간이 증가하지 않고 주기도 증가하지 않는 경우
- 실행요구 시간이 증가하지 않고 주기가 감소하지 않는 경우

이제 각 경우에 대하여 태스크 집합의 스케줄링 가능성을 검사하고 최적의 퀀텀 크기를 구하는 알고리즘을 분석해 본다.

2.1 첫 번째 경우

이 경우는 주어진 태스크들의 실행요구 시간이 감소하지 않고 주기가 증가하지 않는다고 가정한 경우이다. 태스크 집합 τ에 대한 도달 순위 리스트는 표 1과 같다.

표 1을 구하기 위해서는 먼저 각 태스크의 $Rank$ 값을 구해야 한다. 예를 들어, T_1은 실행요구 시간이 7이고 주기가 41인 태스크로서 퀀텀의 크기 Q가 21이면 이용률이 1이 된다. 따라서 21의 직전 값인 20이 T_1의 $Rank$ 값이 된다. 이러한 방식으로 모든 태스크들의 $Rank$ 값을 구한 후, 이들을 줄지 않는(non-decreasing) 순서로 정렬해 놓은 것이 표 1이다.

표 1. τ에 대한 도달 순위 리스트(첫 번째 경우)
Table 1. Reach rank list for τ(the first case)

i	0	1	2	3	4
Rank[i]	2	7	16	19	20
Task 번호	T_2	T_5	T_3	T_4	T_1

알고리즘 1은 [2]에서 제시한 알고리즘을 개선한 것으로서 도달 순위 리스트를 기반으로 Q를 결정한다. 알고리즘 1에서는 먼저 프로세서의 수 M에 의거하여 도달 순위 리스트에서 $Rank[M-1]$을 찾는다. 다음으로는 퀀텀 크기가 $Rank[M-1]$인 경우에 태스

크들의 이용률의 합을 계산하는데 알고리즘 1내의 Calculate_U 함수가 이 일을 수행한다. 계산된 이용률의 합이 M보다 작거나 같다면 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링 가능한 것이고 이때 최적의 퀀텀 크기는 $Rank[M-1]$이 된다.

이용률의 합이 M보다 크다면 도달 순위 리스트에서 $Rank[M-2]$를 찾고, 퀀텀 크기가 $Rank[M-2]$인 경우에 태스크들의 이용률의 합을 계산한다. 계산된 값이 M보다 작거나 같다면 주어진 태스크 집합은 스케줄링 가능한 것이고, 이 경우 최적 퀀텀 크기는 $Rank[M-2]$가 된다.

이러한 과정은 태스크들의 이용률의 합이 M보다 작거나 같게 될 때까지 반복 실행되는데 경우에 따라서는 계산된 Q가 최적이 아닐 수도 있다. Q가 최적이 아니라는 것은 퀀텀 크기가 $Q+1$일 때 주어진 집합이 스케줄링 가능한지의 여부를 보고 판단할 수 있다. 계산된 Q가 최적이 아닌 경우에는 바로 이전 단계의 $Rank$ 값으로부터 시작하여 퀀텀의 크기를 1만큼씩 감소시키면서 Q를 찾으면 된다.

```
int FindQ() {
    int Q = 1;
    int base, i;
    for(i=M-1; i>=0; i--) {
        if(calculate_U(ts, Rank[i]) <= M) {
            Q = Rank[i];
            base = i;
            break;
        }
    }
    if(calculate_U(ts, Q+1) <= M) {
        for(i=Rank[base+1]; i>0; i--){
            if(calculate_U(ts, i) <= M) {

                Q = i;
                break;
            }
        }
    }
    return Q;
}
```

알고리즘 1. 최적 퀀텀 크기 Q를 결정하는 함수
Algorithm 1. The Function that determines the optimal quantum size Q

주어진 태스크 집합 τ에 대해 M이 4이면 $Rank[3]=19$이고 이 경우 이용률의 합 U는 4보다 작거나 같으므로 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링 가능하고 최적의 퀀텀 크기는 19가 된다. 참고로 퀀텀 크기를 1만큼 증가시키면 이용률의 합 U가 4.5가 되어 스케줄링이 불가능하게 된다.

$M=3$이면 $Rank[2]=16$인 경우와 $Rank[1]=7$인 경우가 모두 이용률의 합이 3을 넘어서 스케줄링이 불가능하게 된다. 계속해서 $Rank[0]=2$이고 이 경우 이용률의 합은 2.2634이어서 스케줄링이 가능하다. 그러나 퀀텀 크기를 1만큼 증가시켜서 이용률을 계산하면 2.8012가 되어 역시 스케줄링이 가능하다. 따라서 이 경우에는 도달 순위 리스트를 이용해서 계산한 Q값이 최적이 아니므로 $Rank[1]$로부터 시작하여 퀀텀의 크기를 1만큼씩 감소시키면서 최적 퀀텀 크기를 구하면 된다. 이렇게 구해진 최적 퀀텀 크기는 5이다.

2.2 두 번째 경우

이 경우는 주어진 태스크들의 실행요구 시간이 감소하지 않고 주기도 감소하지 않는다고 가정한 경우이다. 주어진 태스크 집합 τ에 대한 도달 순위 리스트는 표 2와 같다. 표 2를 구하는 방법은 표 1의 경우와 같다. 그러나 이 경우는 주어진 태스크들의 실행요구 시간이 감소하지 않고 주기도 감소하지 않는다고 가정한 경우이므로 그 결과는 표 1의 경우와 다르다.

표 2. τ에 대한 도달 순위 리스트(두 번째 경우)
Table 2. Reach rank list for τ(the second case)

i	0	1	2	3	4
Rank[i]	3	21	38	40	47
Task 번호	T_2	T_5	T_4	T_1	T_3

태스크 집합 τ에 대해 $M=4$이면 $Rank[3]=40$이고 이 경우 이용률의 합은 4이어서, 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링 가능하고 최적의 퀀텀 크기는 40이 된다. 이 경우에 퀀텀 크기를 1만큼 증가시키면 $U=4.5$가 되어 스케줄링이 불가능하게 된다.

$M=3$이면 $Rank[2]=38$인 경우와 $Rank[1]=21$인 경우가 모두 이용률의 합이 3을 넘어서 스케줄링이 불가능하게 된다. $Rank[0]=3$이고 이 경우 이용률의 합은 2.1951이어서 스케줄링이 가능하다. 그러나 퀀텀 크기를 1만큼 증가시켜서 이용률을 계산하면 2.7152가 되어 역시 스케줄링이 가능하다. 따라서 이 경우에는 도달 순위 리스트를 이용해 계산한 값이 최적이 아니므로 $Rank[1]$로부터 시작하여 퀀텀의 크기

를 1만큼씩 감소시키면서 최적 퀀텀 크기를 구하면 된다. 이렇게 구해진 최적 퀀텀 크기는 20이다.

2.3 세 번째 경우

이 경우는 주어진 태스크들의 실행요구 시간이 증가하지 않고 주기도 증가하지 않는다고 가정한 경우이다 주어진 태스크 집합 τ에 대한 도달 순위 리스트는 표 3과 같다. 표 3을 구하는 방법은 표 1의 경우와 같다. 그러나 태스크들에 대한 가정이 다르므로 그 결과는 표 1의 경우와 다르다.

표 3. τ에 대한 도달 순위 리스트(세 번째 경우)
Table 3. Reach rank list for τ(the third case)

i	0	1	2	3	4
Rank[i]	3	21	38	40	47
Task 번호	T_2	T_5	T_4	T_1	T_3

$M=4$이면 $Rank[3]=40$이고 이 경우 이용률의 합은 5이어서 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링이 불가능하다. 계속해서 $Rank[2]=38$인 경우와 $Rank[1]=21$인 경우도 이용률의 합이 4를 넘으므로 스케줄링이 불가능하다. $Rank[0]=3$이고 이 경우 이용률의 합은 2.5846이어서 Pfair 스케줄링이 가능하다. 이 경우에 퀀텀 크기는 3이다.

여기서 퀀텀 크기를 1만큼 증가시켜 이용률을 계산하면 2.5055이어서 역시 스케줄링이 가능하다. 따라서 3은 최적의 퀀텀 크기가 아니다. 최적 퀀텀 크기는 $Rank[1]$로부터 시작하여 퀀텀의 크기를 1만큼씩 감소시키면서 구하게 된다. 이렇게 구해진 최적 퀀텀 크기는 20이다.

2.4 네 번째 경우

이 경우는 주어진 태스크들의 실행요구 시간이 증가하지 않고 주기는 감소하지 않는다고 가정한 경우이다. 주어진 태스크 집합 τ에 대한 도달 순위 리스트는 표 4와 같다. 표 4를 구하는 방법은 표 1의 경우와 같다. 그러나 태스크들에 대한 가정이 다르므로 그 결과는 표 1의 경우와 다르다.

$M=4$이면 $Rank[3]=40$이고 이 경우 이용률의 합 U는 4보다 작거나 같으므로 주어진 태스크 집합은 Pfair 스케줄링 알고리즘에 의해 스케줄링 가능하고 최적의 퀀텀 크기는 40이 된다. 퀀텀 크기를 1만큼 증가시키면 $U=4.5$가 되어 스케줄링이 불가능하게 된다.

표 4. τ에 대한 도달 순위 리스트(네 번째 경우)
Table 4. Reach rank list for τ(the fourth case)

i	0	1	2	3	4
Rank[i]	3	21	38	40	47
Task 번호	T_2	T_5	T_4	T_1	T_3

$M=3$이면 $Rank[2]=38$이고 이 경우 이용률의 합 U는 3.5이어서 스케줄링이 불가능하다. 계속해서 $Rank[1]=21$이고 이 경우 이용률의 합은 2.8333이어서 스케줄링이 가능하게 된다. 이때 퀀텀 크기를 1만큼 증가시키면 $U=3.3333$이 되어 스케줄링이 불가능하게 된다. 따라서 최적 퀀텀 크기는 21이다.

Ⅳ. 결 론

일반적으로 스케줄링 가능한 퀀텀의 크기가 증가하게 되면 문맥 교환을 하기 위한 시간이나 캐시 재적재 시간 등이 감소하게 되어 시스템 성능이 향상되는 요인이 된다. 따라서 Pfair 스케줄링에서 최적의 퀀텀 크기는 스케줄링 가능한 최대의 퀀텀 크기이다. 본 논문에서는 Pfair 스케줄링에서 태스크들의 실행요구 시간이 가변적일 때 최적 퀀텀 크기를 동적으로 결정하기 위한 개선된 방법을 제안하였다.

이전 방법에서는 mode change되는 태스크 집합에 속한 각 태스크의 주기가 감소하거나 증가하는 경우만을 고려하고 실행요구 시간은 감소하지 않는다는 가정 하에서 최적의 퀀텀 크기를 결정하였는데 본 논문에서는 실행요구 시간이 감소하는 경우까지도 고려하여 태스크의 도달 함수를 정의하였다. 또한 고려된 경우에 대하여 도달점을 구하기 위한 함수를 제안하고 그 결과를 증명하였으며, 최적의 퀀텀 크기를 동적으로 결정하는 방법을 제안하였다.

본 논문에서 제안된 알고리즘은 이전의 연구에서 제안된 알고리즘을 부분적으로 개선하였다. 즉, 도달 순위 리스트에 의해서 최적 퀀텀 크기를 구하지 못한 경우에도 보다 빠르게 최적 퀀텀 크기에 접근할 수 있도록 설계되었다.

본 논문에서 정의된 도달 함수는 많은 경우에 있어서 상수 시간에 최적의 퀀텀 크기를 구할 수 있었으나 그렇지 못한 경우도 존재하였다. 이를 개선하기 위해서는 일부 도달 함수를 수정하고 이를

증명하는 것이 필요할 것이다.

본 논문에서 제안된 방법은 향후 퀀텀 크기 증가에 따른 스케줄러 동작 횟수의 감소나 CPU 클럭 감소에 따른 퀀텀 크기의 결정 등을 통해서 저전력 설계에도 응용될 수 있을 것이다.

참고문헌

[1] 김인국, 흐름 공정 모델의 효율적인 실시간 스케줄링, 아주대학교 박사학위 논문, 1995.

[2] 김인국 외, “Mode Change 환경에 적합한 동적 퀀텀 크기 스케줄링”, 콘텐츠학회논문집, 제6권, 제9호, pp. 28-41, 2006.

[3] 김인국 외, "개선된 동적 퀀텀 크기 Pfair 스케줄링의 구현", 한국산학기술학회논문지, 제10권 제10호, pp. 2760-2765, 2009.

[4] C.L. Liu and J.W. Layland, "Scheduling algorithm for multiprogramming in a hard real-time environment", JACM, Vol.20, pp. 46-61, 1973.

[5] D. Zhu, D. Mosse, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?", Real-Time Systems Symposium, Proceedings of the 24th IEEE Real-time Systems Symposium, pp. 142-151, Dec. 2003.

[6] J. Anderson and A. Srinivasan, A New Look at Pfair priorities, Technical report, Dept of Computer Science, Univ. of North Carolina, 1999.

[7] J. Anderson and A. Srinivasan, "Early-release fair scheduling", Proceedings of the 12th Euromicro Conference on Real-time Systems, pp. 35-43, June. 2000.

[8] J. Anderson and A. Srinivasan, "Pfair scheduling: beyond periodic task systems", Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications, pp. 297-306, Dec. 2000.

[9] J. Anderson, A. Block, and A. Srinivasan, "Quick-release fair scheduling", Proceedings of the 24th IEEE Real-time Systems Symposium, pp. 130-141, Dec. 2003.

[10] W. Jane, S. Liu, Real-time Systems, Prentice Hall, Apr. 2000

[11] S. Baruah, J. Gehrke, and C.G. Plaxton. "Fast scheduling of periodic tasks on multiple resource", Proceedings of the 9th International Parallel Processing Symposium, pp. 280-288, Apr. 1995.

[12] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel, "Proportionate progress: a notion of fairness in resource allocation", Algorithmica, Vol.15, pp. 600-625, 1996.

저 자 소 개

박 현 선

2000년 : 단국대학교 전자계산학과 학사.
2003년 : 단국대학교 전자계산학과 석사.
2006년 : 단국대학교 전자계산학과 박사수료.

관심분야 : 운영체제, 실시간시스템, 임베디드시스템.
Email : hyunnys@dankook.ac.kr

김 인 국

1982년 : 단국대학교 학사.
1985년 : 미국 에모리대학교 석사.
1995년 : 아주대학교 박사.
2001~2003년 : 미국 뉴멕시코 공과대학 방문교수.

현재, 단국대학교 컴퓨터과학과 교수.
관심분야 : 운영체제, 실시간시스템, 임베디드시스템.
Email : igkim@dankook.ac.kr