論文2000-37CI-4-2

최대여유시간 제공 연성 비주기 실시간 태스크 스케줄링 알고리즘
(A Soft Aperiodic Real-Time Task Scheduling Algorithm Supporting Maximum Slack Time)

林 德 周 *, 朴 成 漢 **

(Duk-Joo Lim and Sung-Han Park)

요 약

본 논문은 고정 우선순위 실시간 시스템에서 연성 비주기 실시간 태스크의 on-line 응답시간을 줄이기 위한 연산의 최소화에 목적이 있다. 제안하는 알고리즘은 온라인 시의 오버헤드를 줄이면서, 동시에 비주기 태스크에 할당할 수 있는 최대의 여유시간을 제공함으로써 목표를 만족시킨다. 제안하는 알고리즘은 고정우선순위 비주기 실시간 태스크 스케줄링에서 off-line시 최적의 응답시간을 내는 알고리즘인 Slack Stealing에 비해 응답시간의 손실이 거의 없으면서 오버헤드 측면에서 7배 가량의 좋은 성능을 나타낸다.

Abstract

The purpose of this paper is to minimize the a slack computation time of the scheduling of a soft aperiodic real-time tasks in a fixed priority real-time system. The proposed algorithm reduces the computation overhead at on-line time and supports the maximum slack time assigned for aperiodic real-time tasks. The proposed algorithm has 10~20% more response time for aperiodic real-time tasks than that of Slack Stealing Algorithm that offers optimal response time in fixed priority real-time system. However, the performance of the proposed algorithm is seven times better in a scheduling overhead.

I. 서 론

실시간 시스템이 일반 시스템과 구별되는 점은 시스템 동작 상에 있어서 신속, 정확한 작업의 처리 뿐 아니라 주어진 종료시한 내에 작업을 끝마쳐야 한다는 시간적 정확성을 제공해야 한다는 것이다. 실시간 시스템은 사용되려는 목적에 따라 시스템 내의 태스크들의 특성이 달라진다. 그러므로 어떠한 스케줄링 알고리즘을 선택할 것인가는 곧 어떠한 태스크를 스케줄링 할 것인가의 판단으로 결정될 수 있다.

실시간 시스템에서의 태스크들은 우선순위, 주기성, 종료시한 등의 존재 여부에 따라 분류될 수 있다. 우선 순위가 할당된 태스크 집합은 우선 순위에 따라 선점(preempty)이 발생하여 수행 순서가 바뀌어 지며, 미리 할당된 태스크의 우선 순위가 온라인 시에 바뀌지 않는 방식을 고정 우선순위 스케줄링이라 하고 온라인 시에도 우선 순위를 계속 변화시키는 방식을 동적 우선순위 스케줄링이라 한다. 주기성에 따라서는 정해진 시간마다 발생하는 주기 태스크(periodic task)와 일정한 주기 없이 발생하는 비주기 태스크(aperiodic task)로 나뉘어 진다. 종료시한에 따라 종료시한을 만족시키지 않으면 시스템에 치명적인 영향을 미치는 경성 실시간 태스크(hard real-time task)

* 正會員, (주) 케이디이컴 소프트웨어 개발실

** 終身會員, 漢陽大學校 컴퓨터工學部

(Dept. of computer Eng., Hanyang University)

接受日字:2000年 1月 24日, 수정완료일:2000年 7月 12日

와 시간적 제약에 내성을 지녀 실행 도중 지연이 생겨도 시스템의 성능에 별다른 영향을 미치지 않는 연성 실시간 태스크(soft real-time task)로 나뉜다. 이러한 예로써 원자력발전소나 자동 비행 항법 장치 등의 핵심 태스크들은 경성 실시간 태스크 집합이라 할 수 있고, 예약 시스템이나 멀티미디어 서버 등은 연성 실시간 태스크 집합이라 할 수 있다.

수많은 실시간 운영체제와 특별한 목적을 지닌 운영체제의 개발 연구는 동적 우선 순위 실시간 스케줄링 알고리즘을 적용하고 있다. 하지만 상업적인 실시간 운영체제 제품은 동적 실시간 스케줄링 알고리즘을 적용하지 않는다. 그 이유는 고정 실시간 시스템이 종료시한에 대해서 엄격한 정책을 시행할 수 있으므로 동적 실시간 시스템에 비해 실세계에 더 적응력이 있기 때문이다. 또한 어떤 응용에도 적합한 동적 스케줄링 알고리즘을 수용하기 위해서는 동적 실시간 시스템에 오버헤드 증가가 불가피하기 때문에 고정 우선 순위 스케줄링 알고리즘이 사용된다[1].

본 논문은 고정 실시간 시스템에서 태스크를 스케줄링함에 있어 비주기적으로 발생하는 태스크들의 응답시간을 줄이기 위한 온라인 시의 연산을 최소화하는데 그 목적이 있다[2],[3]. 주기 태스크의 경우 각각의 종료시한을 가지고 있어 그 종료시한만 만족을 시켜 준다면 스케줄링에 아무 문제가 없다는 점에서 비주기 태스크의 성능 향상에 초점을 두게 된다. 즉, 주기 태스크는 경성 실시간 태스크이며 비주기 태스크는 연성 실시간 태스크로 간주된다. 비주기 태스크의 응답시간을 줄이기 위한 알고리즘은 크게 두 가지 방식으로 나뉠 수 있다. 하나는 비주기 태스크의 실행을 위한 주기 서버를 두어 주기 태스크와 동등한 수준으로 스케줄링해 주는 것으로 Deferrable Server[4] 및 Polling Server[5] 등이 있다. 그러나 이 방법들은 알고리즘이 단순하여 구현은 쉽지만 시스템의 유휴시간이 많아 비주기 태스크의 응답시간이 좋지 않다. 다른 방법은 주기 태스크의 실행 시 생기는 여유시간(slack time)을 계산하여 이 시간을 비주기 태스크 실행에 제공하는 방식이다. 이 방식 중 최적으로 알려진 Slack Stealing[6],[7] 방식은 비주기 태스크의 스케줄링 시에 최소의 응답시간을 제공하는 가장 좋은 알고리즘이다. 하지만 오프라인 시 계산되는 여유시간 테이블의 크기가 크고, 온라인 시의 자료구조 갱신 횟수 및 여유시간 계산을 위한 연산횟수가 많아 온라인 시 오버헤드가 많다는 단점 때문에 실제 시스템에 적용되기에는 어려움이 따른다. 본 논문에서는 이러한 문제를 해결하기 위해 오버헤드가 적으면서도 비주기 태스크의 응답시간이 Slack Stealing 알고리즘에 비해 큰 차이가 나지 않는 알고리즘을 제안한다.

제안한 알고리즘은 Slack Stealing과는 달리 off-line시에 여유시간을 계산하고, 이의 사용기간을 명시함으로써 알고리즘이 단순하면서도 비주기 태스크 발생 시 제공할 수 있는 최대의 여유시간을 할당해 준다. 본 논문의 구성은 먼저 2절에서 제안한 알고리즘의 전략과 적용 예를 보여준다. 그리고 3절에서 기존의 알고리즘과의 복잡도 및 시뮬레이션 결과를 비교하여 이 알고리즘의 적합성을 보이며 4절에서 결론을 맺는다.

Ⅱ. 최대여유시간 사전할당 알고리즘

1. 기본 가정

본 논문에 사용되는 주기 태스크에 대한 기본 가정은 Liu와 Layland의 비율단조 알고리즘(Rate Monotonic Algorithm)[3] 적용 시의 가정을 이용한다. 비율단조 알고리즘은 고정 우선순위 스케줄링 알고리즘의 대표적인 알고리즘으로서 다음과 같은 요구사항을 만족시켜야 한다.

- 경성 실시간 태스크들의 시간 제약 조건을 만족해야 한다.
- 경성 실시간 태스크들의 스케줄 가능한 사용률이 필요 이하로 낮아져서는 안 된다.
- 일시적인 과부하에서도 스케줄 가능해야 한다.

위의 두 번째 항목은 비주기 태스크에 대한 서비스로 인하지 않은 주기 태스크의 지연이 있어서는 안 된다는 것을 의미한다. 세 번째 항목은 주기 태스크들이 동시에 발생하는 위기 순간(Critical instant) 또는 이 순간 비주기 태스크가 발생하는 등 태스크 발생이 집중될 때에도 제한시간 내에 태스크가 수행되어야 한다는 것이다.

이러한 요구사항을 만족시키는 스케줄링 알고리즘이 적용되는 태스크의 특성은 다음과 같다.

- 주기 태스크는 고정된 우선순위를 가지고 있으며 선점 가능하다
- 주기 태스크는 주기의 시작에 바로 시작되며 주기의 끝이 종료시한이 된다. 또한 수행도중 자발적

으로 중단되지 않는다.

● 문맥 교환(context switching)에 드는 비용은 태스크의 실행시간에 포함된다.

● 모든 태스크들은 비의존적이다. 즉, 태스크 간의 동기화 등은 존재하지 않으며 따라서 최대 수행시간은 미리 알 수 있다.

제시된 조건들은 실제 시스템 환경에 비해 제한을 가지지만 알고리즘의 성능을 비교하는데 문제되지는 않는다. 제시하는 알고리즘이 적용되는 태스크 집합은 다음의 예와 같다. 공장의 모니터링 시스템은 초기 설정 시에 관리해야 할 모든 기계들의 상태를 체크 하는 주기 태스크를 실행해야 한다. 간혹 발생하는 이벤트는 비주기 태스크로 간주되며, 다른 기계의 상태 체크를 계속 진행하면서 이 이벤트를 처리한다. 즉 위의 가정에 대한 스케줄링 가능성이 성립하는 경성 주기 태스크와 연성 비주기 태스크가 본 스케줄링 알고리즘의 적용 대상이다.

2. 스케줄링 알고리즘

여유시간 계산을 위해 초월주기(H : Hyper-period) 동안의 주기태스크 집합의 스케줄링을 고려한다. 주기태스크의 스케줄링은 H를 주기로 하여 계속 반복되기 때문이다. H는 주기 태스크들의 주기의 최소공배수이다.

H동안의 주기태스크의 동작 시뮬레이션을 통해 공통여유시간(slack time)을 쉽게 구해낼 수 있으며 그 알고리즘은 그림 1과 같다. 이 후 언급되는 여유시간은 공통여유시간을 말한다. 이 여유시간 값은 오프라인 시에 구해지며, 테이블로 저장된다. 저장된 여유시간을 언제 할당되는가 따라 비주기 태스크의 응답시간을 결정하게 된다. 여유시간은 최대 H/T_n만큼 발생가능하며 여유시간이 사용됨으로 해서 주기태스크가

```
●종료 시간 : Sj
while(current_time < H){
  for(Ti){
    if(current_time%H = 0){
    /*새로운 주기 시작*/
    if(Ti.comp > 0){
      /*이전 주기태스크의 미완료*/
      No Schedulability;
      Exit;
    }
    create Ti
    if(slack_time>0){
      /*여유시간이 존재-테이블에 저장 */
     slack_table.start <- (ending time of previous Tn) or (ending time of previous slack time)
     slack_table.end <- current_time
     slack_table.current_slack <- slack_time
    }
    if(i = n) /*최장 주기 태스크라면*/
      save ending time of current Tn
 }

 if((Ti.prio > T_suspend .prio) and (Ti.comp>0))
   /*더 높은 우선순위 태스크가 존재하지 않고 현재 태스크의 실행이 종료되지 않았다면*/
   compute for a time-unit
 else if(Ti.prio < T_suspend.prio )/*
   T_suspend <- Ti
 if(모든 태스크가 유휴상태였다면)
   slack_time <- slack_time +1
 }
}
```

그림 1. 오프라인 시의 여유시간 테이블 계산 알고리즘

Fig. 1. The Off-line Slack-Time Computing Algorithm.

종료시한을 만족하지 못하는 상황이 일어나서는 안 된다. 이를 위해 여유시간의 할당시간을 제한할 필요가 있다. 할당 시작시각은 최하 우선순위 태스크의 이전 실행에 영향을 받는다. 각 태스크는 이전의 태스크의 종료시점부터 종료 시한 전까지만 수행이 완료되면 된다. 또한 이전의 여유시간이 있다면 그 여유시간의 사용범위에서는 현재의 여유시간은 사용되지 못 한다. 현재 주기 태스크 T_n이 i번째 수행을 마치고 j번째 여유시간이 할당된다면 이 여유시간의 할당시간은 다음과 같이 정해진다.

- 시작 시간 : $\{(t/P_n(*P_n+T_{n,i}+\alpha\}$or $\{S_{j-1}\}$(Pn : 태스크 Tn의 주기, Tn, i : 태스크 Tn의 i번째 수행시간, α : 지연시간, Sj : j번째 여유시간의 사용 종료시간)

그림 2는 비주기 태스크를 고려하지 않은 순수한 주기 태스크 집합의 비율단조 알고리즘에 따른 스케줄링 상태이다. 이러한 스케줄링 시에 존재하는 여유시간은 최장 주기를 가진 주기 태스크 T_n의 종료 시점 이후에 나타나게 된다. 그림 2에서 보여지는 것처럼 여러 개의 주기 태스크가 모두 끝마치는 시점, 즉 우선순위가 가장 낮은 최장주기를 가진 태스크 T_2가 끝난 뒤의 시점이 된다. 이 때에 생성된 여유시간이 비주기 태스크 실행을 위해 제공되며, 여유시간의 제공기간은 다음과 같다. 비주기 태스크가 도달하면 이 때 할당할 수 있는 최대의 여유시간을 알아야 한다. 이는 전체 시간에서 주기 태스크에 제공되고 남은 시간과 같다. 비주기 태스크가 수행되기 위해서는 현재 수행중이거나 수행되려고 하는 주기 태스크가 대신 지연되어야 한다. 즉, 주기 태스크가 지연된 만큼 그 시간에 여유시간이 비주기 태스크에게 할당되는 것이다. 따라서, 이 때의 여유시간은 그 시각에 걸쳐져 있는 주기에서 실행되고 있는 주기 태스크들에 속한 것이기 때문에 최대 기간은, 최종적으로 영향을 받게 되는 최하위 우선순위를 가진 주기태스크 T_n의 실행에 의존하게 된다. 즉, T_n이 이전 실행을 종료한 시점, 또는 이전의 여유시간 종료 기간 중 더 늦은 시각이 현재 여유시간의 할당 시작 시각이 된다. 이렇게 할당 시작 시각을 정함으로써 비주기 태스크들의 스케줄링에 있어 종료시한을 만족시켜 줄 수가 있다.

그림 2에서 빗금으로 묶여진 여유 시간들의 예를 보면, 첫 번째 여유시간 1만큼은 T_2의 지연에 의해 0에서부터 5이전까지의 비주기 태스크에 미리 할당될 수 있다. 두 번째 여유시간 1만큼은 주기 태스크 (2의 종료가 최악의 경우 5에서 발생 할 수 있다. 따라서 5부터 9 이전까지 할당될 수 있다. 즉, 여유시간의 할당 시작시간은 이전 주기의 태스크의 종료시각 또는 이전 여유시간의 할당 종료시간이 된다. 따라서 마지막 여유시간 2만큼은 9에서 15사이에 할당된다.

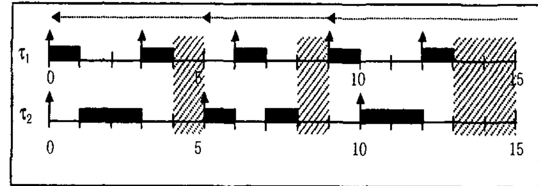


그림 2. 주기 태스크 스케줄링에 의한 여유시간 및 각 여유시간의 할당 기간

Fig. 2. Slack-time and Slack-Time consumption duration for Periodic Real-Time Task Scheduling.

그림 3에서와 같이 주기 태스크의 부하가 큰 경우를 보면, 이전 경우와는 달리 14에 존재하는 여유시간의 시작시간은 이전 여유시간이 존재하지 않기 때문에 다른 기준이 필요하다. 마찬가지로 이 여유시간은 그 시간대에 걸쳐진 최하위 우선순위 태스크 T_2의 3번째 실행에 영향을 받는다. T_2가 1 단위시간만큼 지연됨으로써 T_1의 3번째 실행 또한 시각 11까지 종료를 지연시킬 수 있다. 따라서 시각 8이 이 여유시간의 사용시작시간이 될 수 있다. 그림 2와 그림 3의 예에서와 마찬가지로 0부터 H까지의 여유시간의 시작시간, 종료시간 및 그 크기는 오프라인 시에 테이블로 저장이 되고, H의 정수배에 해당하는 시간마다 이 테이블은 초기값으로 재설정 되며, 비주기 태스크의 발생 시 이 표를 참조하여 해당하는 시간에 존재하는 여유시간을 사용하여 수행하게 된다.

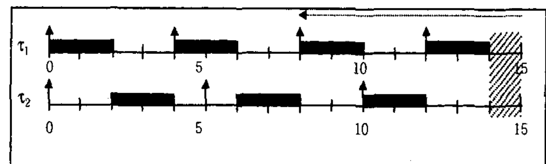


그림 3. 고 부하 주기 태스크 집합의 스케줄링에 의한 여유시간 할당 기간

Fig. 3. Slack-Time consumption duration for Periodic Real-Time Task Scheduling in High Overload.

그림 4는 수행시간 2를 가지는 비주기 태스크의 스케줄링을 보여준다. 시각 7에서 발생한 태스크는 이때의 여유시간 1만큼을 수행하고, 더 이상 할당할 여유시간이 없어 잠시 지연된다. 그리고 나머지는 다음 여유시간이 발생하는 시각 9에 수행을 시작한다. 이상의 동작은 그림 5의 알고리즘을 따른다. 이리하여 비주기 태스크 발생 시 할당할 수 있는 최대 여유시간을 제공함으로써 비주기 태스크의 낮은 응답시간을 보장하며, 온라인 시의 연산의 횟수를 줄일 수가 있다.

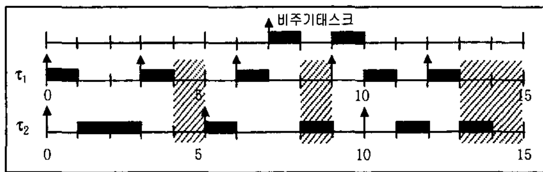


그림 4. 비주기 태스크의 스케줄링
Fig. 4. Aperiodic Real-Time Task Scheduling.

```
while((T_apt의 수행시간) != 0){
   if(slack_table.current_slack != 0){
      //여유시간이 있다면
      if(T_active 존재)
         T_suspend = T_active
      T_active = T_apt
      slack_table.current_slack 갱신
      // 1감소
   }
   else{          //여유시간이 없다면
      T_active = T_suspend
      T_suspend = T_apt
      while(slack_table.current_slack =0)
         delay
   }
}
if(T_suspend가 존재)
 T_active = T_suspend;
```

그림 5. 온라인 시의 스케줄링 알고리즘
Fig. 5. The On-lin Scheduling Algorithm.

Ⅲ. 시뮬레이션

여유시간 도용 알고리즘과의 연산 횟수 비교는 표 1과 같다. 온라인 시의 여유시간을 알아내기 위한 연산횟수는 테이블로부터 해당시간의 여유시간을 검색하는 시간이 되며, 사용되는 검색 알고리즘에 따라 연산횟수가 결정된다. Slack Stealing 방식은 여유시간을 계산하기 위해 n개의 주기 태스크 전체에 대해 현재 할당할 수 있는 각 태스크별 여유시간을 획득하여 이들이 공통적으로 할당할 수 있는 시간을 연산하므로 온라인 시의 연산횟수가 O(n)인데 비해 이 알고리즘은 O(log n)으로 줄어든다.

온라인 시의 자료의 갱신은 여유시간이 소모될 때마다 발생되며 초월주기가 다시 시작할 때 테이블의 값이 초기화되면서 발생한다. 즉 테이블에는 사용되지 않은 여유시간이 저장되어 있으며, 최대 갱신횟수는 이 여유시간이 전부 소모 되었을 때이다. 이에 비해 Slack Stealing 방식은 각각의 주기 태스크 및 비주기 태스크가 시작하고 종료할 때마다 각 태스크별 여유시간을 재갱신 한다.

저장되는 자료의 크기는 표의 결과처럼 이 알고리즘은 주기태스크의 주기에 따라 오프라인 시의 테이블의 크기가 크게 의존하는 성질을 가지고 있다. 최대 용량은 최장 주기 태스크 Tn의 종료 시마다 여유시간이 존재하는 경우로 H/Tn만큼이 된다. Slack Stealing은 각 주기 태스크마다 여유시간을 다로 저장하게 된다. 초월주기 동안 각 태스크의 수행 횟수만큼 여유시간이 생기므로 이에 해당하는 H/Ti 개의 자료를 저장하게 된다.

표 1. 연산 횟수 비교
Table 1. Comparison of the counts Computation.

	본 알고리즘	Slack Stealing 알고리즘
On-line시 여유시간 연산횟수	$O(\log_2 n)$	O(n)
On-line시 자료갱신횟수	$Slack_{total}+H/T_n$ (H내의 총 여유시간)	$\sum_{i=1}^{n}(H/T_i*(i-1)) + n*N_{idle}$
Off-line시 테이블의 크기	H/T_n	$\sum_{i=1}^{n}(H/T_i)$

시뮬레이션은 주기 태스크의 부하를 각각 73%, 88%, 95% 일 때 비주기 태스크의 부하를 바꾸어가며 테스트 한다. Background Server(BS) 방법의 응답시간에 비해 Deferrable Server(BS) 방법과 본 논문에서 제시하는 방법, 그리고 Slack Stealing 방법은 주기태스크의 부하가 비교적 적은 환경에서는 유사한 정도의 응답시간을 낸다. 그림 5부터 그림 7까지의 결

과를 보면 주기 태스크의 부하가 높아질수록 그 응답 시간은 알고리즘에 따라 점차 많은 차이를 나타냄을 알 수 있다. 시뮬레이션 결과는 주기태스크의 부하가 가장 크고 비주기 태스크의 부하 또한 큰 상황에서도 본 알고리즘과 Slack Stealing 알고리즘이 최대 10%~20% 정도의 응답시간의 차이를 보인다.

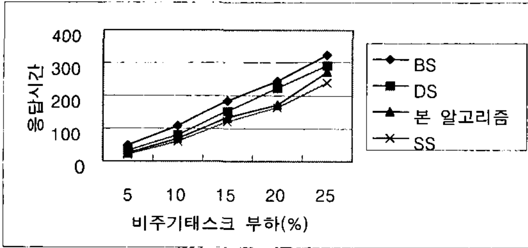


그림 6. 73% 주기 태스크 부하 상의 비주기 태스크 응답시간
Fig. 6. The Response Time of Aperiodic Task with 73% Loads of Periodic Task.

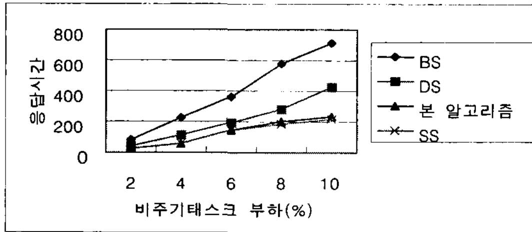


그림 7. 88% 주기 태스크 부하 상의 비주기 태스크 응답시간그림 7. 88% 주기 태스크 부하 상의 비주기 태스크 응답시간
Fig. 7. The Response Time of Aperiodic Task with 88% Loads of Periodic Task.

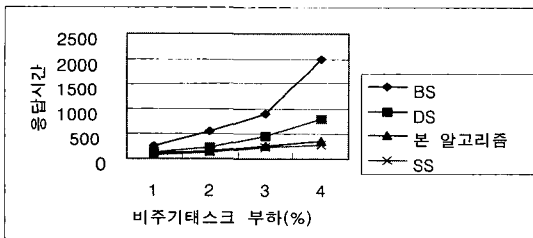


그림 8. 95% 주기 태스크 부하 상의 비주기 태스크 응답시간
Fig. 8. The Response Time of Aperiodic Task with 88% Loads of Periodic Task.

표 2에 나타난 결과는 위와 같은 주기 태스크 부하에서 연산횟수, 즉 오버헤드가 본 알고리즘과 Slack Stealing알고리즘에서 약 7배 정도의 차이를 보여준다. 하지만 이 비율은 장기간의 시스템 사용시간을 고려한다면 Slack Stealing 알고리즘은 한번의 초월주기에서 연산하는 횟수가 많기 때문에 높은 비율로 증가되지만, 본 알고리즘의 경우 낮은 비율로 증가한다. 또 주기 태스크의 개수가 많으면 많을수록 Slack Stealing 알고리즘의 연산 횟수는 본 알고리즘보다 더욱 증가하게 된다.

표 2. 온라인 시의 연산횟수 비교
Table 2. Comparison of the count of Computation in On-line time.

주기 태스크 부하	본 알고리즘	Slack Stealing 알고리즘
73%	2195번	15483번
88%	2328번	15095번
95%	2426번	16156번

Ⅳ. 결론

주기 태스크는 제한 시간을 만족시켜 주는 한 지연에 대해 관용적이라는 점을 이용해 비주기 태스크의 응답시간을 줄이는 방안을 제시한다. 경성 주기 태스크의 스케줄링을 위해 비율단조 알고리즘을 사용하며, 이로써 주기 태스크의 종료시한을 만족하고 주기 태스크의 스케줄링 성능을 보장한다. Slack Stealing 방식이 온라인 시의 부하가 많아 계산복잡도가 높은 반면, 제시된 알고리즘은 오프라인 시의 계산을 통해 온라인 시의 부하를 약 7배 줄일 수 있다. 또한 여유시간의 제공 기간의 설정을 통해 주기 태스크의 스케줄링에 영향을 주지 않으면서도 비주기 태스크에게 할당할 수 있는 최대의 여유시간을 제공으로써 비주기 태스크의 응답시간을 Slack Stealing 알고리즘 성능에 비해 그다지 차이가 나지 않는 정도로 줄일 수 있었다.

향후 연구방향은 온라인 시에 발생하는 주기 태스크들을 처리하는 방법, 종료시한이 주기와 다른 주기 태스크들의 스케줄링 및 비주기 태스크들 간의 FIFO방식 처리가 아닌 종료시한이나 실행 시간의 차에 따른 이성적인 스케줄링 기법 적용 등이 있다.

참 고 문 헌

[1] E. Douglas Jensen, "Scheduling in Real-Time Systems," http://www.realtime-os.

com/sched o3.html. November 20, 1996.

[2] C. L. Liu and J. W. Layland. "Scheduling algorithms for multiprogramming in a hard real-time environment." *Journal of the ACM.* Vol.20, pp. 46-61, Jan. 1973

[3] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic task scheduling for hard real-time systems," *The Journal of Real-Time Systems,* vol. 1, pp. 27-69, Dec. 1989.

[4] J. P. Lehoczky, L.Sha, and J.K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environment," in *Proceedings of the Real-Time Systems Symposium,* pp. 261-270, Dec. 1987.

[5] C.M. Krishna, Kang G. Shin, "Real-Time Systems", *McGRAW-HILL,* pp.47-80, 1977.

[6] J.P. Lehoczky and S.Ramos-Thuel. "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems." in *Proceedings of the IEEE Real-Time Systems Symposium,* pp. 110-123, Dec. 1992.

[7] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling slack time in fixed-priority preemptive systems," in *Proceedings of the Real-Time Systems Symposium,* pp. 160-171, Dec. 1993.

[8] J.Y.T. Leung and J.Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation,* Vol.2, No.4, pp. 237-250, 1982.

[9] J.P.Lehoczky, L.Sha, J.K.Strosnider, and H.Tokuda, "Fixed priority-scheduling theory for hard real-time systems," A.M. van Tilborg and G. M. Koob, eds., *Foundations of Real-Time Computing: Scheduling and Resource Management,* Kluwer Academic Publishers, Boston, pp. 1-30, 1991.

[10] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers,* Vol.39, No.9, pp.1175-1185, 1990.

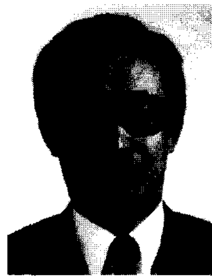
[11] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," *Proceedings of the Real-Time Systems Symposium,* pp. 201-209, Dec. 1990.

[12] Giorgio C. Buttazzo, Scuola Superiore S. Anna, "Hard Real-Time Computing Systems-Predictable Scheduling Algorithms and Applications", *Kluwer Academic Publisher,* 1997.

저 자 소 개

林 德 周(正會員)

1998. 2 : 한양대학교 전자계산학과 졸업(공학사). 2000. 2 : 한양대학교 대학원 전자계산학과 졸업(공학석사). 1997. 3～현재 ㈜ 케이디이컴 소프트웨어 개발실 연구원. 주관심분야 : 실시간 운영체제, 작업 스케줄링 알고리즘



朴 成 漢(終身會員)

1970. 2 : 한양대학교 전자공학과 졸업(B.S). 1973. 8 : 서울대학교 대학원 전자공학과 졸업(M. S.). 1984. 5 : 텍사스 주립대학 전기 및 컴퓨터공학과 졸업(Ph. D.). 1974.3～1978.8 : 경북대학교 전자공학과 전임강사. 1984.5～1984.8 : 미국 텍사스 주립대학 Instructor. 1984.8～1986.2 : 금성사 중앙 연구소 수석 연구원. 1989.8～1990.7 : 미국 텍사스 주립대학 Visiting Researcher. 1995.3-1997.2 : 한양대학교 공학대학 학장. 1986.3～현재 : 한양대학교 전자컴퓨터공학부 교수. 주관심분야 : IMT-2000, 멀티미디어 통신, 컴퓨터 비젼