# 실시간 환경하에서 혼합 우선순위 할당에 의한 비주기적 태스크 스케줄링 알고리즘

# (An Aperiodic Task Scheduling Algorithm by Hybrid Priority Assignment in Real-Time Environments)

김 형 일 † 이 승 룡 †† 이 종 원 ††† 김 정 순 ††††

(Hyungill Kim) (Sungyoung Lee) (Jongwon Lee) (Jungsoon Kim)

**요 약** 본 논문은 혼합된 고정/가변 우선순위 (Hybrid Static/Dynamic Priority) 실시간 시스템에서 중단형 (preemptive) 연성 마감시간 비주기적 태스크 스케줄링 알고리즘을 제안한다. 제안한 태스크 스케줄링의 주요 목표는 모든 주기적 태스크의 마감시간을 보장하는 범위에서 비주기적 태스크들에 대한 빠른 평균 반응시간을 얻을 뿐만 아니라 구현이 간단하며, 스케줄링 예측성을 가지고자 하는데 있다. 이러한 목표를 달성하기 위하여 본 논문은 새로운 형태의 비주기적 태스크 스케줄링 원리를 적용하였는데, 그것은 고정 우선순위 할당 전략과 오프라인에서 만들어진 가상 역시간 우선순위 테이블의 정보들을 런타임 시 비주기적 태스크의 도착에 따라 가변적으로 혼합하여 스케줄링하는 방법이다. 비주기적 태스크의 평균 반응시간이라는 관점에서 모의 실험 결과, 제안한 알고리즘이 다른 혼합형 비주기적 태스크 스케줄링 알고리즘에 비하여 성능 개선을 이루었으며 특히 과부하 시 잘 작동하였다.

***Abstract*** This paper presents a preemptive scheduling algorithm for servicing soft deadline aperiodic tasks in hybrid static/dynamic priority real-time systems. The major goals of the proposed scheduling algorithm are not only to guarantee all the deadlines of periodic tasks and to obtain the fast response time for aperiodic tasks, but also to gain implementation simplicity and to retain considerable scheduling predictability. To achieve these goals, we have adopted a new aperiodic task scheduling principle in which a fixed-priority assignment strategy and the information on a virtually reversed fixed-priority scheduling table built off-line are properly mixed according to the status of aperiodic tasks' arrivals at runtime. The paper also shows simulation results in terms of the average aperiodic response time verifying that the new algorithm offers significant performance improvement over the other conventional joint scheduling algorithms, especially under a heavy transient overload.

## 1. 서론

실시간 시스템은 작게는 자동차 연료 분사 제어로부터 크게는 내장형 (embedded) 군사 전략 무기체계에 이르기까지 사용 범위가 광범위할 뿐 아니라 그 응용 분야가 점점 확대되어 가고 있다. 또한 가변적이며 예측하기 힘든 환경 (예를 들면, 해저 탐사 또는 무인 로보트 우주 탐사)에서 운용해야 하는 실시간 시스템에서는 효율적이며 예측성이 뛰어나고 부하가 높을 때에도 반응시간이 빠른 비주기적 태스크 스케줄러의 개발이 중요하다[10],[17].

최근 Carnegie Mellon 대학에서 주기적 태스크와 비주기적 태스크들이 혼합된 환경에서 주기적 태스크의 마감시간을 보장하고 비주기적 태스크의 반응시간이 기존의 백그라운드 (background)와 폴링 서버 (polling server) 알고리즘에 비해 빠른 대역보존 (bandwidth

※ 본 논문은 '94 한국과학재단 핵심연구 지원사업의 중간 결과로 이루워졌음

† 준 회 원 : 경희대학교 전자계산공학과
†† 정 회 원 : 경희대학교 전자계산공학과 교수
††† 비 회 원 : 한국통신 소프트웨어 연구소
†††† 비 회 원 : 수원대학교 전자계산학과 교수

논문접수 1994년 11월 18일
심사완료 1995년 2월 28일

preserving)[5],[15],[16] 및 slack stealing[6] 알고리즘을 개발하였다. 대역보존 알고리즘의 기본 개념은 예측할 수 없이 도착하는 비주기적 태스크의 서비스를 위하여 주기적 태스크에 해당하는 하나의 서버 태스크를 생성한 후 그 태스크의 대역 (실행시간) 범위에서 비주기적 태스크에게 서비스를 제공하는 방법으로, 폴링 서버와는 달리 대역을 일정 기간 보존하는 방법이다. 그러나, 대역보존 알고리즘들 (Deferrable Server, Priority Exchange, Sporadic Server)은 대역보존 수준의 최적 값을 결정하기가 어려울 뿐만 아니라 임의의 시간에 도착하는 비주기적 태스크의 서비스를 위하여 가변적으로 그 값을 바꾸기가 힘들다는 제약 점이 있다.

한편 slack stealing 알고리즘은 대역보존 알고리즘들과는 달리 주기적인 서버 태스크를 만들지 않는 대신에 현재 상태에서 비주기적 태스크를 서비스하기 위한 시간을 계산할 수 있는 slack stealer의 개념을 도입하였다. 이 slack stealer는 현재 모든 주기적인 태스크의 완료 시점을 마감시간까지 미루고 비주기적인 태스크 처리를 위한 시간 (슬랙)을 찾아내는 역할을 하게 된다. 이러한 방법은 메모리 시스템에서 cycle stealing[11] 방식과 유사하며, 비주기적 태스크가 마감시간을 가지고 있지 않은 경우는 물론 마감시간을 가지고 있는 경우에도 적용할 수 있다는 장점이 있다.

그러나, slack stealing 알고리즘은 슬랙을 계산할 때 각 스케줄링 지점에서 모든 우선순위 수준에 대하여 검색을 하므로 계산량이 방대하여 그것을 실제 시스템에 적용하기에는 무리가 따르며, 온라인 상에서 최대 슬랙 ($A^*$)을 구하는 과정에서 주기적 태스크 수행조건에 대한 천정 값 (ceiling values)을 고려하기 때문에 모든 가용 슬랙을 충분히 활용할 수가 없다[1]. 참고로, 최근 Tia et al. [19]은 slack stealing 알고리즘을 non-greedy한 방법으로 개선하였는데, 그들은 만일 가용슬랙을 사용하기 전에 낮은 우선순위 태스크들이 서비스 된다면 어떤 특정한 구간에서는 slack stealing 알고리즘에서 구할 수 있는 것 보다 더 많은 슬랙이 있다는 것을 보여 주었다.

본 논문에서는 슬랙을 구하는 방법이 단순하고 기존의 알고리즘들에 비하여 슬랙의 양을 상대적으로 많이 찾을 수 있는 우선순위 지시 (Priority Indicating) 알고리즘을 제안한다. 우선순위 지시 알고리즘은 주어진 주기적 태스크 집합에 대한 고정 우선순위[7],[8],[9] 스케줄링을 이용하여 가상 역시간 우선순위 스케줄링 테이블을 작성한 후 비주기적 태스크가 도착하였을 때, 주기적 태스크의 마감시간을 보장하는 범위에서 고정 우선순위 스케줄링과 역시간 테이블을 참조하여 주기적 태스크를 고정 또는 가변 우선순위를 혼합한 방법으로 할당하는 스케줄링 기법이다. 참고로, 가상 역시간 우선순위 테이블은 주기적 태스크가 최소한 어느 시점에 할당되어야 자신의 마감시간을 보장할 수 있는지에 대한 정보를 가지고 있으므로, 비주기적 태스크가 도착하였을 때 그 정보를 참조하여 온라인 상에서 할당해야 할 주기적 또는 비주기적 태스크의 우선순위를 지시 (결정)해야 하므로 본 알고리즘을 "우선순위 지시" 라고 명명하였다. 한편, 역시간 테이블의 크기는 각 주기적 태스크 주기의 최소 공배주기 (hyperperiod)이며 역시간 테이블 내의 할당된 주기적 태스크의 실행시간은 정시간 스케줄링의 각 주기에 대한 대응되는 시간으로, 마치 하이퍼주기의 끝을 원점으로 하여 역시간으로 가상 스케줄링을 하는 것과 같다.

가상 역시간 우선순위 스케줄링 테이블의 사용 목적은 오프라인에서 주어진 주기적 태스크들에 대한 수행시간 할당을 마감시간 직전까지 미루어 놓음으로써 비주기적 태스크들이 주기적 태스크 각 주기의 앞 지역에 왔을 경우 서비스를 즉각 제공할 수 있는 가용 슬랙을 앞 지역에 가상적으로 보유하겠다는 것이다. 이와 같이 주기적 태스크의 할당을 마감시간까지 미룰 수 있는 근거는 주기적 태스크 집합의 value function은 대개 step function 형태로써[3] 태스크의 실행시간을 마감시간까지 미루어도 주기적 태스크의 값 (value)은 변하지 않기 때문이다. 반면 단주기 우선 순위 스케줄링의 목적은 결과적으로 볼 때 비주기적 태스크가 주기적 태스크의 각 주기의 뒷 지역에 몰려서 왔을 경우에 서비스를 제공할 수 있는 가용 슬랙을 뒷부분에 보유하겠다는 것이다.

따라서, 임의의 비주기적 태스크가 왔을 경우에 주기적 태스크의 할당 순서를 고정 우선순위 스케줄링과 가상 역시간 우선순위 테이블을 참조하여 가변적으로 결정함으로써 각 주기적 태스크의 마감시간을 보장하는 범위에서 비주기적 태스크를 위한 최적 슬랙 값을 제공해 주는 방법이 본 알고리즘의 기본 아이디어이다.

본 알고리즘의 장점은 1) 오프라인에서 만들어진 역시간 테이블을 런타임 시 이용하므로 비주기적 태스크를 위한 슬랙을 찾는데 드는 오버헤드가 적으며, 2) 고정

1) $A_u = t - P_i(t)$이다. 여기서 $A_u$는 주기적 태스크 i수준에서 가용슬랙이며, t는 현재시간, $P_i(t)$는 0에서 부터 t시간까지 주기적인 태스크의 실행시간의 합이다 $P_i(t) = C_k\lceil t/T_k \rceil + jC_i$ 이며 여기서 $C_k$는 주기적 태스크 k수준에서의 실행시간이고, $T_k$는 k수준의 주기이고, j는 $T_i$의 j번째 인스턴스이다

우선순위와 가변 우선순위를 혼합하여 사용함으로써 고정 우선순위를 기반으로 한 알고리즘보다 비주기적 태스크에 대한 반응시간이 빠르며, 3) slack stealing 알고리즘이 스케줄링 시 매번 모든 수준의 주기적 태스크를 고려하여 슬랙을 계산하는 것과는 달리, 제안한 알고리즘은 스케줄링 시 해당하는 주기적 태스크만을 고려하므로 슬랙 계산 과정이 단순하여 구현이 쉽고, 4) 미리 만들어진 역시간 테이블 정보를 이용함으로써 스케줄링 예측성이 높다

본 논문의 구성은 다음과 같다. 2장에서는 본 알고리즘에서 사용하는 가정 및 기본 용어를 정의하고 알고리즘을 소개하며, 알고리즘의 실제 운용 예를 보여 주고, 알고리즘에 대한 몇 가지 고려 사항을 언급한다. 3장에서는 주기적 태스크의 부하에 따른 시뮬레이션 결과를 보여 주며, 마지막으로 4장에서는 결론 및 앞으로의 연구 방향에 대해 기술한다.

## 2. 우선순위 지시 알고리즘

### 2.1 알고리즘의 배경

연성 비주기적 태스크 스케줄링의 일반적인 특성은 비주기적 태스크의 반응 시간을 가능한 빨리 하기 위해서 주기적 태스크의 마감시간을 보장하는 범위에서 비주기적 태스크가 도착 시 어떻게 서비스를 즉각 제공할 수 있느냐에 초점을 두고 있다.

본 알고리즘에서 필요한 가정은 다음과 같다.

A1 주기적 태스크 인스턴스의 마감시간은 다음 인스턴스의 도착시간과 동일하다.

A2. 주기적 태스크나 비주기적 태스크에 대한 선점은 언제나 가능하다.

A3 모든 문맥 교환 비용은 해당되는 주기적 태스크나 비주기적 태스크의 실행시간에 포함되어 있다

A4. 주기적 태스크의 도착시간, 실행시간, 주기, 비주기적 태스크의 도착시간 및 실행시간은 모두 스케줄링 단위시간의 정수배이다.

본 알고리즘은 고정 우선순위 (예를 들면 단주기 우선순위 스케줄링: rate monotnoic scheduling) 알고리즘[9]의 스케줄링과 역시간 테이블을 이용하여 비주기적인 태스크를 서비스할 수 있는 슬랙을 찾는 기법이다 고정 우선순위 알고리즘은 간단하면서도 주기적인 태스크에 대한 스케줄링 효율이 비교적 높으며[2],[14], 수학적으로 스케줄하려는 모든 주기적인 태스크가 자신의 마감시간을 넘기지 않는 효율(utilization)의 상한선을 미리 계산할 수 있다. 하지만, 실제 태스크 집합들은 이보다 높은 효율에서도 마감시간을 넘기지 않는 경우가 있으므로 critical zone에서 스케줄 가능성 여부를 시험해야 한다[9]

본 알고리즘에서 필요한 몇 가지 정의 및 정리는 다음과 같다.

Definition 1. 역시간 스케줄링은 정시간 스케줄링에 대한 시간적인 반전을 의미한다.

일반적인 고정 우선순위 (본 논문에서는 단주기 우선순위) 알고리즘으로 스케줄링하는 경우 "정시간 스케줄링"이라고 하며 이에 비하여 역시간 테이블을 이용하여 스케줄링하는 것을 "역시간 스케줄링"이라고 한다. 역시간 스케줄링은 역시간 테이블의 정보를 이용하여 주기적 태스크의 마감시간 내에 실행시간을 보장하기 위한 스케줄링을 말한다. 역시간 테이블은 정시간 스케줄링의 하이퍼주기 내에서 주기적 태스크의 실행시간을 각주기에 대하여 시간적으로 반전 시킨 것이다 이 경우 태스크 집합의 특성은 주기와 마감시간이 같아야 한다

Definition 2. 역시간 스케줄링에서 슬랙의 합은 정시간 스케줄링에서 슬랙의 합과 같다.

역시간 스케줄링이 진행된다고 하더라도 실제 주기적인 태스크가 실행되는 시간적인 크기에는 변함이 없다. 따라서, 슬랙의 크기도 변함이 없다.

Definition 3 역시간 스케줄링 시 각 태스크의 주기는 정시간 스케줄링의 대응되는 각 태스크의 주기와 같다.

하이퍼주기는 각 주기적 태스크 주기의 최소공배수로써 스케줄링의 단위이며 모든 태스크의 시작이 동일한 곳에서 시작하여 모든 태스크의 마감시간이 동일한 곳까지를 말한다. 따라서, 다음과 같은 속성을 갖는다.

$$H_i = \frac{T_h}{T_i} \quad (H_i = 1,2,..,)$$

여기에서,

$T_h$ : 하이퍼주기,

$T_i$ : 태스크 i의 주기

$H_i$ : 태스크 i의 주기가 $T_h$안에 반복되는 횟수이다.

하이퍼주기는 시작과 끝이 대칭이고, 대응되는 각 태스크의 $H_i$가 같으므로 각각의 주기는 동일하다.

Theorem 1. 정시간 스케줄링에서 스케줄이 가능한 태스크 집합은 역시간 스케줄링에서도 스케줄이 가능하다.

증명 어떤 주기적인 태스크 i의 j번째 주기에 해당하는 인스턴스의 처리 과정을 살펴보면 그림1과 같다.

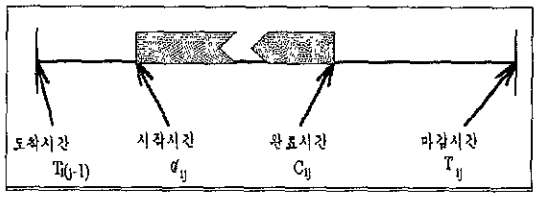



그림 1 임의의 주기의 인스턴스

태스크 i의 j번째 인스턴스가 도착하는 시간은 $T_{i(j-1)}$이며, 이 인스턴스의 수행이 시작되는 시간 (시작시간)은 $\phi_{ij}$가 되며 완료시간은 $C_{ij}$이다. 완료시간은 시작시간과 마감시간 사이에 존재하며 마감시간은 가정에 의하여 다음 인스턴스의 도착 시간이므로 $T_{ij}$이다. 이를 다음과 같이 정리할 수 있다

단주기 우선순위 알고리즘에서 스케줄 가능한 태스크 집합 $\{\tau_i | i=1,2,\ldots,n\}$이 있다고 가정하자. 이 태스크 집합은 다음의 성질을 만족한다.

$$T_{i(j-1)} \le \phi_{ij} \tag{1}$$

$$\phi_{ij} < C_{ij} \tag{2}$$

$$C_{ij} \le T_{ij} (= D_{ij}) \tag{3}$$

$$E_i \le C_{ij} - \phi_{ij} \tag{4}$$

여기에서

$\tau_i$ = 주기적 태스크 $i$
$E_i$ = $\tau_i$의 실행시간
$T_i$ = $\tau_i$의 주기
$D_i$ = $\tau_i$의 마감시간
$T_{ij}$ = $jT_i$
$\phi_{ij}$ = $\tau_i$의 $j$번째 인스턴스의 시작시간
$C_{ij}$ = $\tau_i$의 $j$번째 인스턴스의 완료시간
$i$ = $1,2,\ldots,n$이고,
$j$ = $1,2,\ldots,H_i (H_i = T_h/T_i)$이다.

역시간 스케줄링은 하이퍼주기의 끝을 그 시작점으로 하고 하이퍼주기의 시작까지 역시간으로 스케줄하므로 어떤 태스크의 임의의 인스턴스에 대하여 다음의 식이 성립해야 적합성을 갖을 수 있다

$$T'_{i(j-1)} \le \phi'_{ij} \tag{1}$$

$$\phi'_{ij} < C'_{ij} \tag{2}$$

$$C'_{ij} \le T'_{ij} (= D'_{ij}) \tag{3}$$

$$E'_i \le C'_{ij} - \phi'_{ij} \tag{4}$$

여기에서

$$T'_{ij} = T_h - T_{i(H_i-j)}$$
$$\phi'_{ij} = T_h - C_{i(H_i-j+1)}$$
$$C'_{ij} = T_h - \phi_{i(H_i-j+1)}$$
$$E'_i = E_i$$

부등식 (5),(6),(7),(8)을 단주기 우선순위 알고리즘의 속성을 표현하는 변수들로 풀어 쓰면 다음과 같다.

$$T'_{i(j-1)} \le \phi'_{ij}$$
$$\Rightarrow T_h - T_{i(H_i-j+1)} \le T_h - C_{(iH_i-j+1)} \tag{9}$$
$$\Rightarrow T_{iH_i-j+1}' \le 'C_{i(H_i-j+1)}$$

$$\phi'_{ij} < C'_{ij}$$
$$\Rightarrow T_h - C_{i(H_i-j+1)} < T_h - \phi_{(iH_i-j+1)} \tag{10}$$
$$\Rightarrow \phi_{(iH_i-j+1)} < C_{(iH_i-j+1)}$$

$$C'_{ij} \le T'_{ij}$$
$$\Rightarrow T_h - \phi_{(iH_i-j+1)} \le T_h - T_{(iH_i-j+1)} \tag{11}$$
$$\Rightarrow T_{(iH_i-j+1)} \le \phi_{(iH_i-j+1)}$$

$$E'_i (= E_i) \le C'_{ij} - \phi'_{ij}$$
$$\Rightarrow E_i \le T_h - \phi_{(iH_i-j+1)} - T_h + C_{(iH_i-j+1)} \tag{12}$$
$$\Rightarrow E_i \le C_{(iH_i-j+1)} - \phi_{(iH_i-j+1)}$$

이상과 같이 정리된 부등식 (9),(10),(11),(12)는 부등식 (3),(2),(1),(4)에 의해 각각 성립함을 알 수 있다 그러므로, 부등식 (5),(6),(7),(8)은 모두 성립한다.

### 2.2 우선순위 지시 알고리즘 설명

우선순위 지시 알고리즘은 역시간 테이블과 정시간 스케줄링 기법을 이용한다.

역시간 테이블은 임의의 주기적 태스크 집합에 대하여 단주기 우선순위 알고리즘을 적용하여 만들 수 있으며 (태스크 집합의 하이퍼주기 범위 내에서 정시간 스케줄링 기법을 적용하여 정시간 스케줄링 테이블을 만들고) 이것을 기반으로 하여 시간적으로 반전된 스케줄링 정보를 갖는 역시간 테이블을 만든다.

그림2에서 슬랙 시간은 비주기적인 태스크를 처리할 수 있는 시간으로 주기적인 태스크를 처리하고 남은 시간 또는 마감시간을 넘지 않는 범위 내에서 주기적인 태스크를 뒤로 연장하여 얻을 수 있는 시간이다 그림 2(a)는 단주기 우선순위 알고리즘을 적용했을 경우에 얻어지는 정시간 스케줄링 테이블이고 그림2(b)는 이를

시간적으로 반전 시켰을 때의 상태, 즉, 역시간 테이블이다

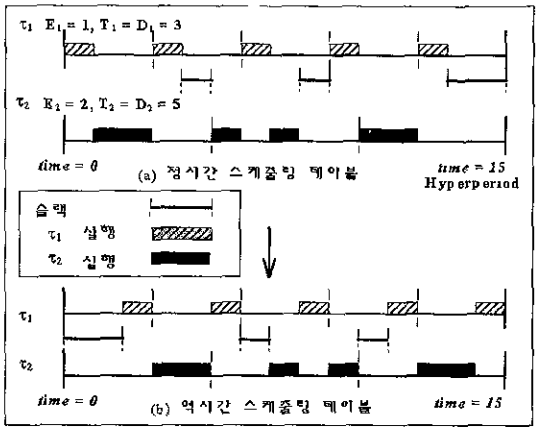



그림 2 역시간 스케줄링 테이블

그림2(a)와 그림2(b)를 비교해 보면, 하이퍼주기 내 각각의 슬랙의 합은 동일하며 시간상으로 하이퍼주기에 대칭인 위치에 존재한다. 따라서, 스케줄러가 역시간 테이블을 참조할 경우 비주기적인 태스크를 실행할 수 있는 시간을 가능한 각 주기의 앞 지역에서 확보할 수 있다.

어느 한 스케줄링 시점에서 시스템이 지시 알고리즘을 이용하여 비주기적 태스크를 서비스하려면 다음 조건들 중 어느 하나를 만족해야 된다.

C1) 현재 역시간 테이블에 비주기적 태스크를 위한 슬랙이 있는가?

C2) 현재 역시간 테이블이 가리키는 주기적 태스크 단위 (태스크 인스턴스의 실행시간의 일부 또는 전부)가 이미 처리되었는가?

만일 서비스할 비주기적인 태스크가 없거나 위의 조건에 해당되지 않으면 정시간 스케줄링에 의해 주기적인 태스크가 서비스 된다.

본 알고리즘은 온라인 상에서 역시간 스케줄링과 정시간 스케줄링을 가변적으로 혼합한 스케줄링 알고리즘이다. 이러한 알고리즘의 동적인 속성으로 인하여 모든 주기적 태스크의 마감시간을 보장하면서도, 비주기적 태스크 처리를 위한 최대 가용 슬랙이 하이퍼주기 내의 스케줄링 구간을 통하여 유지된다 (비주기적 태스크가 하이퍼주기의 어느 특정 지역에 많이 도착하면 그 지역에서는 역시간 스케줄링에 의해 주기적 태스크가 처리될 확률이 크기지만, 반대로 비주기적 태스크가 거의 도착하지 않는 지역은 정시간 스케줄링에 의해 주기적 태스크가 처리될 확률이 크다)

아래의 그림 3은 우선순위 지시 알고리즘의 의사 코드이다.

```
1 initialize data structures
  /* include getting a periodic task set,
  creating a reversed scheduling table, setting timer to zero, etc. */
2 loop begin
3       if (a periodic task unit not yet been serviced has occurred) then service it
4       else if (aperiodic task(s) is ready or arrived) then service it
5       else if (periodic task(s) is ready) then service it
6       else process a CPU idle state
7       advance timer
8       if ((timer_value MOD hyperperiod) is equal to zero)  then
          reinitialize the global parameters
9 end loop
```

그림 3 우선순위 지시 알고리즘 의사 코드 (pseudo code)

의사 코드의 1번은 필요한 자료구조를 초기화하는 것이며 구체적으로 주기적 태스크, 비주기적 태스크, 태스크 생성기, 스케줄러의 자료구조를 초기화하고 역시간 테이블을 만든다 스케줄링이 시작되면 2번부터 9번은 무한히 반복된다. 위에서 기술한 두 가지 조건 (C1), (C2) 중 어느 하나라도 만족하는지를 3번에서 결정한다. 4번에서는 만약, 두 가지 조건이 모두 만족하지 않으면 주기적 태스크를 서비스하고, 두 가지 조건 중 하나라도 만족하게 되면 비주기적 태스크를 즉시 서비스할 수 있다 5번에서 만약 서비스할 비주기적 태스크가 존재하지 않으면 일반적인 고정 우선순위 알고리즘으로 주기적 태스크를 서비스하게 된다. 6번에서 현재 서비스할 주기적 태스크와 비주기적 태스크가 존재하지 않을 경우에 해당되므로 유휴 상태로 처리한다. 7번에서 다음 단위시간의 스케줄링을 위하여 타이머를 1증가시킨다. 이것은 시뮬레이션을 위한 것이므로 실제 상황에서는 고려하지 않는다. 8번에서는 역시간 테이블이 하이퍼주기 만큼의 정보를 가지고 있으므로 하이퍼주기를 넘었는지를 조사한다. 하이퍼주기가 넘었을 경우 9번에서

테이블을 다시 초기화한다. 마지막으로 3번으로 되돌아간다.

### 2.3 우선순위 지시 알고리즘 적용 예

우선순위 지시 알고리즘을 이용하여 주어진 3개의 주기적인 태스크로 이루어진 시스템에서 비주기적인 태스크를 스케줄하는 예를 살펴보자.

그림4는 주기적인 태스크 스케줄링 방법으로 단주기 우선 순위 알고리즘을 이용하였을 경우 하이퍼주기 내에서 예상할 수 있는 태스크들의 동작 상태이다. 역시간 테이블은 이를 단순히 시간적인 역 방향으로 바꾼 것이며 그 내용은 각 스케줄 시간마다 처리할 태스크의 번호를 포함하고 있다.

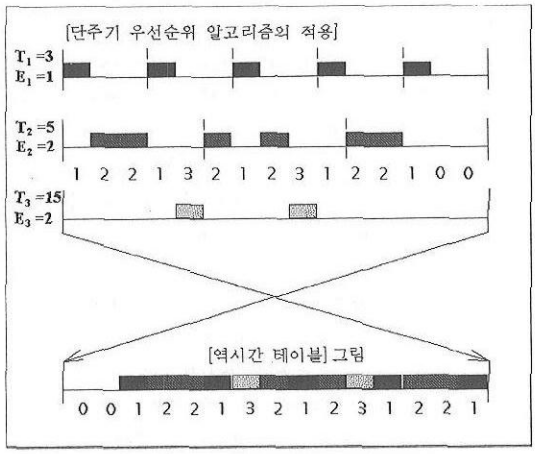



그림 4 역시간 테이블의 생성

두개의 비주기적인 태스크가 t = 5와 t = 8에서 실행 시간 E = 1로 도착했을 때의 동작 상태를 살펴보자.

그림5에서 처럼 t = 0부터 t = 5이전 (비주기적인 태스크가 도착하기 전)까지는 단주기 우선 순위 알고리즘을 적용해서 스케줄링이 진행된다. 3개의 주기적인 태스크의 우선 순위는 주기가 짧은 순서이므로 $t_1$, $t_2$, $t_3$ 순이다. 즉, 우선 순위와 태스크의 번호는 동일하다.

이 때, t = 5에서 비주기적인 태스크 $A_1$이 도착하였다. 현재, 역시간 테이블이 알려 주고 있는 태스크는 $\tau_{12}$이지만, 실제 $\tau_{12}$은 자신의 주기 안에서 이미 처리가 된 상태이므로 (조건 C2를 만족하는 경우) 바로 $A_1$을 서비스하였다.

t = 6 부터는 다시 정상적으로 주기적인 태스크가 실행된다. t = 8에 다시 $A_2$가 도착하였다. 이 때, 역시간 테이블이 $\tau_{13}$를 알려 주고 있지만, $\tau_{13}$ 역시 이미 실행이 끝난 상태이다 (조건 C2를 만족하는 경우). 따라서, 바로 서비스가 가능하다.

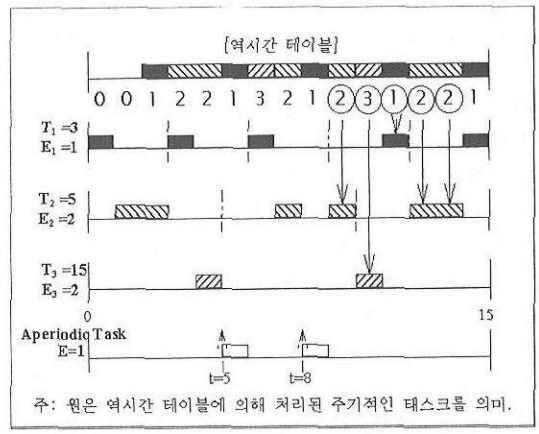



그림 5 비주기적 태스크 서비스 예

t = 9에서 주기적인 태스크 $\tau_{14}$이 큐에 도착했지만, 역시간 테이블은 $\tau_{22}$를 알려 주고 있으며 실행된 시간은 1이고 테이블에서 알려 준 시간은 2이므로 실행된 시간이 테이블에서 알려 준 시간보다 작으므로 $\tau_{22}$를 실행한다. 이 경우에, 실제로 $\tau_{22}$를 실행하지 않으면 $\tau_{22}$를 마감시간 내에 처리하지 못하는 결과를 초래한다. 이후 t = 10에서는 역시간 테이블이 $\tau_{31}$을 알려 주었고 역시간 테이블에 의해 보장되어 있는 자신의 실행 시간이 2이고 실제 실행된 시간은 1이므로 $\tau_{14}$, $\tau_{23}$보다 $\tau_{31}$이 먼저 실행되었다. t = 11에서는 역시간 테이블에서 $\tau_{14}$을 지시하고 있으므로 $\tau_{14}$을 수행하게 된다.

t = 12, 13 에서 $\tau_{23}$는 한번도 실행되지 않았으므로 역시간 테이블에 의해 수행되었고, 마지막으로 t = 14 에서 $\tau_{15}$가 수행을 마치게 된다.

스케줄러는 간단한 조건에 의하여 처리해야 할 태스크를 선택하는데 비하여 주기적인 태스크의 마감시간을 정확하게 보장하고 비주기적인 태스크의 반응시간 (response time)을 매우 빠르게 한다. 간단한 조건만 필요한 것은 이미 역시간 테이블이 제한 시간을 넘지 않는 범위에서 가급적 뒤로 미루어 처리할 수 있는 정보를 제공하고 이 정보를 이용하여 주기적인 태스크를 마감시간 안에 처리하기 때문이다.

### 2.4 알고리즘 구현 시 고려 사항 및 토의

우선 순위 지시 알고리즘을 실제 시스템에 적용하기 위하여 필요한 몇 가지 중요한 사항과 알고리즘의 시간 복잡도에 대하여 알아보자. 첫 번째 주목해야 할 사항

은 역시간 테이블의 자료 구조이다. 역시간 테이블은 주기적 태스크 집합이 적은 경우에는 문자열로 나타낼 수 있으며, 많은 수의 주기적 태스크가 존재할 경우에는 주기적 태스크의 번호가 나열된 정수형 배열로 나타내든지 또는 주기적 태스크의 번호와 실행 시간으로 결합된 배열로 나타낼 수 있다. 2.3 절의 예제에 사용된 역시간 테이블은 하이퍼주기 만큼 문자열 001221321231221로 나타낼 수 있다. 여기서 0, 1, 2, 3은 각각 슬랙, $\tau_1$, $\tau_2$, $\tau_3$를 의미한다. 이렇게 만들어진 역시간 테이블을 이용하여 조건 (C1)을 판단하는 것은 매우 단순한 작업이다. 스케줄링 시 역시간 테이블이 0을 가르키고 있다면 이는 비주기적 태스크를 즉시 서비스 할 수 있음을 나타낸다.

그러나, 조건 (C2)를 판단하기 위해서는 주기적 태스크의 개수 만큼 두개의 카운터로 쓰일 또 다른 자료 구조가 필요하다. 그 중 하나는 $CP_i$ (Computation Processing done for the periodic task i)로 현재까지 수행된 실행시간의 크기이고 다른 하나는 $CP_i$ (Computation Requirements preassinged for the periodic task i)로 역시간 테이블에 보장되어 있는 실행시간의 크기이다. 온라인 상에서 $(CP_i - CR_i)$를 계산하여 조건 (C2)를 만족하였는지를 판단할 수 있다. 예를 들어, [그림 5]의 t=5일 때 비주기적 태스크 A1이 도착하였다. 첫째로, 알고리즘은 (C1)의 판단을 위하여 역시간 테이블을 조사한다. 현재 역시간 테이블은 슬랙이 아닌 1을 지시하므로 (C1)은 성립하지 않는다. 다시, (C2)를 판단하기 위하여 $(CP_1 - CR_1)$를 계산한다. $CP_1=2$ 이고 $CR_1=1$이므로 $(CP_1 - CR_1)$은 양수인 1이므로 즉시 $A_1$을 서비스하였다

역시간 테이블을 구성하는 단위는 단위 시간이기 때문에 주기적 태스크의 평균 실행시간보다 단위 시간이 큰 경우에는 CPU의 낭비를 초래하고, 평균 실행시간보다 단위 시간이 작은 경우에는 스케줄링의 오버헤드가 증가하기 때문에 경험적 방법으로 주기적 태스크 집합의 평균 실행시간과 유사하게 단위시간을 조정하는 작업이 필요하다.

우선순위 지시 알고리즘의 시간 복잡도를 알아보기 위하여 n개의 주기적 태스크로 구성된 태스크 집합이 있다고 가정하자. 우선순위 지시 알고리즘에서 역시간 테이블을 만들 때의 시간 복잡도는 가상 다항시간 (pseudo polynomial time)이며, 최적의 경우는 O(nlog2n)[8]이다. 그러나, slack stealing 알고리즘은 사전에 슬랙을 계산하는 과정의 시간 복잡도는 O(n2)[18]이다. 한편, 스케줄링 시 비주기적 태스크가 도착하게 되면 현재 슬랙이 있는지를 판단해야 한다. 이 판단에 필요한 시간 복잡도는 모든 대역보존 알고리즘과 우선순위 지시 알고리즘은 O(1)이다. 반면에 slack stealing 알고리즘은 모든 우선순위 수준에 대하여 이를 검사해야 하기 때문에 시간 복잡도는 O(n)이다. 따라서 우선순위 지시 알고리즘이 slack stealing 알고리즘보다 많은 수의 주기적 태스크로 이루어는 태스크 집합의 경우 스케줄링 시 오버헤드가 훨씬 작다는 장점을 가진다. 따라서, 우선순위 지시 알고리즘이 오버헤드가 적은 이유는 스케줄링 시 미리 작성된 역시간 테이블을 이용하여 비주기적 태스크를 서비스할 수 있는 슬랙을 찾으므로 많은 양의 계산이 필요하지 않기 때문이다.

본 알고리즘 구현 시 주기적 태스크가 도착하면 고정 우선순위에 의해 스케줄링 되지만 비주기적 태스크가 도착하면 주기적 태스크는 역시간 테이블의 정보에 따라 우선순위가 바뀌어 할당될 수 있다. 이러한 우선순위 역전 현상이 고정 우선순위 스케줄링의 장점[4],[7],[8],[9]을 손상시키는 것 처럼 보일 수 있으나 비주기적 태스크 스케줄링의 궁극적인 목표가 모든 주기적인 태스크의 마감시간을 보장하면서 비주기적 태스크에 대한 반응시간을 빠르게 함과 동시에 알고리즘의 구현이 용이하고 슬랙을 계산하는 방법이 단순해야 한다는 관점으로 본다면, 고정 우선순위 정책을 지키기 위한 대가로 슬랙을 계산하는 과정이 너무 복잡한 경우 (예를 들면, slack stealing 알고리즘), 스케줄링의 원래 목표가 그로 인하여 흐려지게 될 수 있다. 따라서, 고정 우선순위를 유지함으로써 얻을 수 있는 장점보다 고정 우선순위는 지키지 못하지만 다른 방법 (예를 들면, 가변 우선순위 또는 혼합 우선순위)으로 슬랙 계산과 구현의 단순성을 확보할 수 있다면 그것이 비주기적 태스크 스케줄링의 목표에 더욱 부합되는 접근 방법이라고 할 수 있다. 따라서, 본 알고리즘은 온라인에서 혼합 우선순위로 태스크들을 할당하지만 고정 우선순위를 기반으로 한 slack stealing 알고리즘보다 전체적으로 구현이 간단하며 성능 개선 및 예측성을 확보하였다는 면에서 의의를 찾을 수 있다.

또한 본 알고리즘에서 사용한 역시간 우선순위 테이블은 Chetto[1]의 EDL (Earliest Deadline as Late as possible)의 방법과 유사하나 그것은 다음과 같은 점에서 근본적인 차이점이 있다. 첫째 Chetto는 마감시간과 주기가 같아야 EDL 스케줄링을 할 수 있으나 본 알고리즘에서 사용하는 역시간 우선순위 테이블은 EDL 우선순위를 가지고 마감시간지향 사전할당 (deadline-

wise preassignment)으로 생성할 수 있기 때문에 그 같은 제약조건을 완화 (relaxation)할 수 있는 개선된 스케줄링 기법이다. 둘째로는 Chetto는 경성 비주기적 태스크를 스케줄링하기 위한 방법으로 EDL을 도입하였으나 본 알고리즘은 slack stealing의 문제점인 계산의 복잡도를 줄이기 위한 접근 방법으로 역시간 우선순위 테이블을 도입하였다.

## 3. 알고리즘 시뮬레이션

이 장에서는 시뮬레이션을 통해 백그라운드, Sporadic, slack stealing 알고리즘과 지시 알고리즘의 성능을 비교하겠다 Deferrable Server와 Priority Exchange에 비해 Sporadic Server는 성능이 우수하므로 대역보존 알고리즘의 대표로 선택하였다

주기적인 태스크는 임의로 설정한 주기가 서로 다른 10개로 스케줄링 되며, 비주기적인 태스크는 무작위 수 추출을 위하여 pseudo random number generator를 이용하여 만들었으며, 도착 시간은 Poisson 분포를 따르며, 실행 시간은 exponential 분포를 이용하였으나 단위시간의 스케줄링을 위하여 실행시간과 도착시간의 소수점 이하 부분은 생략하여 정수형으로 이용하였다. 제안한 알고리즘과 대역보존 및 slack stealing 알고리즘을 객관적으로 유사한 시뮬레이션 환경을 제공하기 위하여 본 논문은 세 가지 다른 종류의 주기적 태스크 집합에 대한 부하량 (40%, 70%, 90%)을 제공하였으며 그것은 [그림6]에 나타나 있다. 이 같은 구분은 부하에 따른 비주기적 태스크 반응시간의 변화를 살펴보고 각 알고리즘이 동일한 조건하에서 어떤 특성을 보이는지 알아보기 위함이다. 또한, 비주기적인 태스크의 평균 실행 시간은 부하에 상관없이 3 단위시간으로 하였다. 알고리즘간의 비교를 정확하게 하기 위한 시뮬레이션 규칙은 다음과 같다.

1) 주기적 태스크, 비주기적 태스크에 대한 자료 구조를 통일 시켰다
2) 무작위 수 추출 모듈을 공유하여 같은 시점, 같은 실행 시간을 갖는 비주기적인 태스크가 만들어지도록 하였다.
3) 시뮬레이션 결과를 출력하는 모듈을 공유 하였다.

따라서, 각 알고리즘의 핵심 부분을 제외한 부분의 변화는 모든 시뮬레이션 프로그램에 동일하게 영향을 받도록 되어 있다. slack stealing 알고리즘에서 보여 준 시뮬레이션과 마찬가지로 계산된 M/M/1의 평균 반응시간은 실제 알고리즘의 특성과 비교되는 기준으로 삼았으며 실제 시뮬레이션으로 구현한 M/M/1의 평균 반응시간과 거의 유사하였다.

| Task ID | Width 40% Periodic Workload | | Width 70% Periodic Workload | | Width 90% Periodic Workload | |
|---|---|---|---|---|---|---|
| | Period | Computation | Period | Computation | Period | Computation |
| 1 | 33 | 2 | 100 | 2 | 100 | 2 |
| 2 | 105 | 7 | 280 | 8 | 280 | 14 |
| 3 | 21 | 3 | 2100 | 30 | 22100 | 108 |
| 4 | 60 | 4 | 440 | 29 | 440 | 29 |
| 5 | 55 | 4 | 350 | 14 | 350 | 14 |
| 6 | 70 | 1 | 210 | 8 | 210 | 30 |
| 7 | 22 | 3 | 35 | 3 | 35 | 8 |
| 8 | 315 | 10 | 70 | 4 | 70 | 11 |
| 9 | 180 | 12 | 2200 | 46 | 2200 | 231 |
| 10 | 540 | 23 | 300 | 9 | 300 | 12 |

그림 6 시뮬레이션에 사용된 주기적 태스크 집합

### 3.1 주기적인 태스크의 부하가 40%일 때

주기적인 태스크의 부하가 낮은 경우의 기준으로 40%를 삼았다. 실제 상황에서 이와 같이 낮은 부하를 가지는 시스템은 거의 없지만, 낮은 부하에서는 대부분의 알고리즘이 빠른 반응 시간을 갖고 있음을 알 수 있다. 백그라운드 서비스는 비주기, 태스크 가용률이 낮은 관계로 낮은 부하에서도 빠른 반응 시간을 보이지 못했다. 비주기적인 태스크의 부하가 올라감에 따라 평균 반응 시간이 매우 급격하게 증가함을 알 수 있다.

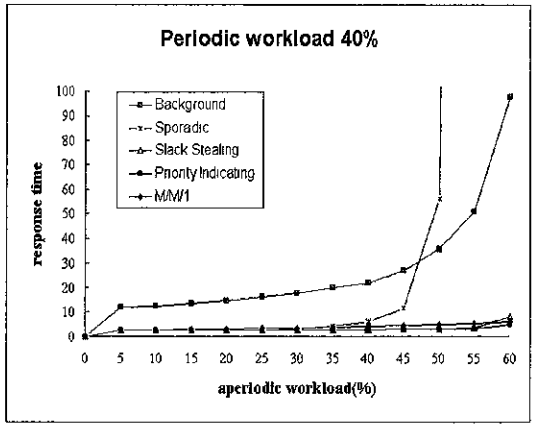



그림 7 시뮬레이션 결과 (40%)

특히, Sporadic Sever는 server의 용량이 주기적인 태스크에 대한 적합성에 영향을 미치므로 가장 최적의 용량을 정하는데 불편함이 따른다. 따라서, 비주기적인

태스크의 부하가 50%가 넘으면서 급격하게 평균 반응 시간이 증가하는 이유는 server 용량의 한계를 넘어섰기 때문이다. 하지만, 낮은 부하에서는 매우 빠른 평균 반응 시간을 가지고 있음을 확인할 수 있다. slack stealing 알고리즘과 본 알고리즘은 M/M/1과 거의 비슷한 평균 반응 시간을 가지고 있음을 알 수 있다. 비주기적인 태스크의 부하 50%까지는 M/M/1의 평균 반응 시간보다 약간은 빠른 평균 반응 시간을 가지고 있으며, 전체 부하가 99%가 되면서 slack stealing 알고리즘의 평균 반응시간이 약간 높아짐을 알 수 있다. 본 알고리즘은 가장 빠른 평균 반응시간을 가지고 있으면서, 높은 부하에서도 상승 곡선이 완만함을 확인할 수 있다.

### 3.2 주기적인 태스크 부하가 70%일 경우

주기적인 태스크가 70%를 차지하므로 전반적으로 알고리즘의 특성이 확연히 구분된다.

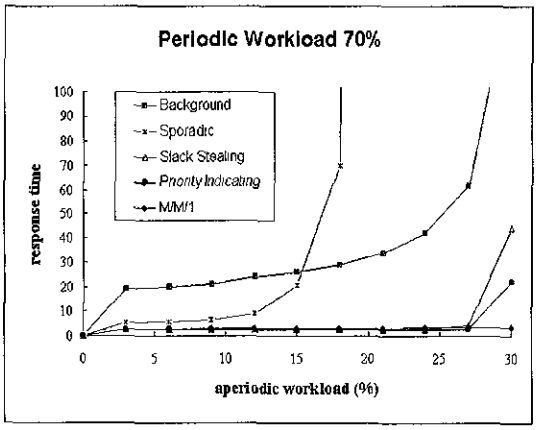



그림 8 시뮬레이션 결과 (70%)

*Sporadic Server*는 이러한 부하가 높아질수록 server는 자신의 용량을 넘는 비주기적인 태스크 부하가 발생하게 되면 백그라운드 서비스 이상으로 급격하게 평균 반응 시간이 높아짐을 알 수 있다.

비주기적인 태스크의 부하가 27%를 넘어서면서 백그라운드, slack stealing, 우선순위 지시 알고리즘 모두 평균 반응시간이 커지기 시작하지만, 40%일 경우와 마찬가지로 그 증가 비율이 다소 차이가 남을 알 수 있다. slack stealing 알고리즘에 비하여 본 알고리즘이 45% 정도 빠른 평균 반응시간을 보여준다

### 3.3 주기적인 태스크 부하가 90%일 경우

매우 높은 부하이므로 Sporadic Server는 그 용량을 정하지 못하므로 시뮬레이션 대상에서 제외하였다. 따라서, 90%의 부하에서는 slack stealing 알고리즘과 우선순위 지시 알고리즘만을 비교하였다.

### 3.4 시뮬레이션 결과

우선순위 지시 알고리즘은 주기적인 태스크의 부하에 상관없이 그리고 전체적으로 높은 시스템의 부하에서도 매우 빠른 평균 반응속도를 가질 수 있음을 보였다. 시뮬레이션에서는 지금까지 발표된 대역보존 비주기적인 태스크 스케줄링 알고리즘에 비하여 더 빠른 평균 반응속도를 나타내었다.

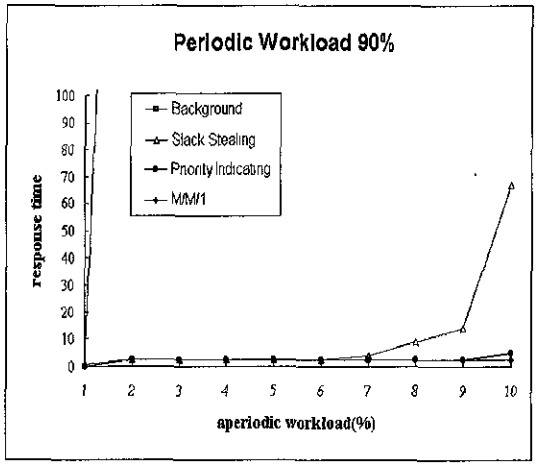



그림 9 시뮬레이션 결과 (90%)

특히 slack stealing 알고리즘에 비하여 구현 과정에서의 오버헤드가 적으면서 높은 부하에서 안정된 평균 반응시간을 가짐을 알 수 있다. 예를 들면, 90% periodic 부하 시 전체 시스템의 부하가 98%인 경우 약 73% 비주기적 태스크 반응 속도 개선, 70% periodic 부하 시 전체 시스템 부하가 98%인 경우 약 55% 반응 속도 개선을 이루었다. 한편 본 알고리즘에서 사용한 가상 역시간 테이블은 주기적인 태스크들의 하이퍼주기에 해당하는 시간 동안의 태스크 정보를 담고 있으며 연결 리스트를 이용하여 테이블을 운영하는데 드는 오버헤드를 줄였다

## 4. 결론 및 향후 연구 계획

대역보존 알고리즘들은 기존의 비주기적 태스크 스케줄링 알고리즘 보다 반응속도를 높이는 결과를 내었지만 많은 한계점을 가지고 있다. *Priority Exchange*와

Deferrable Server는 부하가 급격하게 늘어나면 주기적인 태스크의 마감시간을 보장하지 못하는 결과를 초래하며 이를 개선한 Sporadic Server역시 비주기적인 태스크 부하가 자신의 용량을 초과하면 평균 반응시간이 급격하게 느려지는 단점이 있다. 이와는 달리 slack stealing 알고리즘은 server가 존재하지 않기 때문에 대역 결정 방법으로 인한 문제가 발생하지 않는다. 하지만, 슬랙을 충분히 활용할 수 없다는 점과 슬랙을 계산하는 과정에서 커다란 오버헤드를 가지므로 실제 시스템에 이를 곧바로 적용하기가 어렵다.

이에 비하여 우선순위 지시 알고리즘은 슬랙을 찾아내는 방법이 매우 단순하여 구현에 필요한 오버헤드가 적고 주기적 태스크의 스케줄링 시 혼합 우선순위 할당을 사용하므로 높은 부하에서 더 빠른 평균 반응 속도를 나타내었다. 따라서, 본 알고리즘은 비주기적인 태스크가 돌발적으로 발생하는 가변적인 환경하의 시스템과 주기적인 태스크의 부하가 매우 높은 시스템에 적용이 가능하다

우선순위 지시 알고리즘이 가지고 있는 제한 점이라면 주기적인 태스크의 하이퍼주기만큼 테이블을 만들기 때문에 하이퍼주기가 커지면 그 만큼 역시간 테이블의 크기가 커진다는 점이다. 테이블의 크기에 대한 문제는 slack stealing 알고리즘에서도 동일하게 갖고 있는 문제이며 이를 해결할 수 있는 하나의 방법으로써 슬랙을 찾는데 필요한 정보를 줄일 수 있는 자료구조에 대한 연구가 필요하다.

또한, 앞으로는 본 알고리즘의 개념을 확장하여 가변 우선순위, 예를 들면 EDF (earliest deadline first), 스케줄링에 기초한 연성 비주기적 태스크 스케줄링 알고리즘을 개발할 예정이다. 한편 본 알고리즘에 기초한 경성 비주기적 태스크 (hard deadline aperiodic task) 스케줄링 알고리즘은 [1],[12],[13],[18]과는 다른 방법으로 연구 개발을 마쳤다.

## 참 고 문 헌


[1] H Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm", Proceedings of the IEEE Real-Time System Symposium, pp 110-123, December 1992

[2] M. Gonzalez Harbour, M Klein, and J.P. Lehoczkey, "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority", Proceedings of the IEEE Real-Time System Symposium, pp. 116-128, December 1991

[3] E.D. Jensen, C.D Locke, and H Tokuda, A Time-Driven Scheduling Model for Real-Time Operating Systems, Proceeding of the IEEE Real-Time System Symposium, pp 112-122, December 1985.

[4] J.P Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", Proceedings of the IEEE Real-Time System Symposium, pp 201-209, December 1990

[5] J.P. Lehoczky, L. Sha, and J.K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", Proceedings of the IEEE Real-Time Systems Symposium, pp 261-270, San Jose, CA, December 1987.

[6] J.P. Lehoczky and S Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems", Proceedings of the IEEE Real-Time Systems Symposium, pp. 110-123, December 1992.

[7] J.P. Lehoczky, L. Sha, and Y. Ding. "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", Technical Report, Department of Statistics, Carnegie Mellon University, Pittsburgh, PA, 1987

[8] J.Y.-T. Leung and J. Whitehaed, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks", Performance Evaluation, 2·253-250, 1982.

[9] C.L Liu and J.W. Layland, "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environments", Journal of the Association for Computing Machinery 20(1):46-61, January 1973

[10] A.K. Mok, Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment, Ph.D Thesis, M.I.T., 1983.

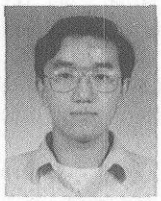
[11] K. Rajkuman, L. Sha, and L. Lehoczky, "On Computing the Effects of Cycle-Stealing in A Hard Real-Time Environment", Proceedings of the IEEE Real-Time System Symposium, pp. 2-11, December 1987

[12] S.Ramos-Thuel and J.P Lehoczky, "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed Priority Systems", Proceedings of the IEEE Real-Time System Symposium, pp 160-171, December 1993.

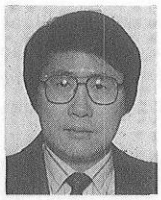
[13] K. Schwan and H. Zhou, "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads", IEEE Transactions on Software Engineering, vol 18, No. 8, August 1992

[14] L Sha, J P Lehoczky, and R. Rajkumar, "Solutions for Some Practical Problems in

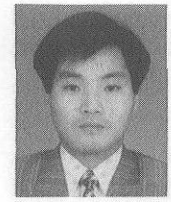
Prioritized Preemptive Scheduling", Proceedings of the IEEE Real-Time Systems Symposium, pp. 181-191, December 1986.
[15] B. Sprunt, J.P. Lehoczky, and L. Sha. "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System", Technical Report CMU/SEI-890TR-11, April 1989.
[16] B. Sprunt, J.P. Lehoczky, and L. Sha. "Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm", Proceeding of the IEEE Real-Time System Symposium, pp. 251-258, December 1988.
[17] J.A. Stankovic and K. Ramamrithm, "What is Predictability for Real-Time System?", in the International Journal of Time-Critical Computing Systems Vol. 2, No. 4, pp. 247-254, November 1990.
[18] S. R. Thuel and J.P. Lehoczky, Algorithms for Scheduling Hard-Aperiodic Tasks in Fixed-Priority Systems using Slack Stealing, Proceedings of the IEEE Real-Time System Symposium, pp. 22-33, December 1994.
[19] T.S. Tia, J. W.S. Liu, and M. Shankar, Algorithms and Optimality of Scheduling Aperiodic Requests in Fixed-Priority Preemptive Systems, Technical Report, University of Illinois, 1994.



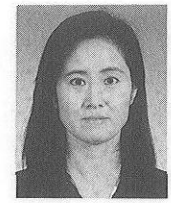
김 형 일
1994년 경희대학교 물리학과 졸업. 1994년~현재 경희대학교 전자계산공학과 대학원 석사과정. 관심분야는 실시간 시스템, 실시간 운영체제, 멀티미디어 시스템임.



이 승 룡
1978년 고려대학교 공과대학 재료공학과 졸업. 1987년 미국 Illinois Institute of Technology 전산학과 석사. 1991년 미국 Illinois Institute of Technology 전산학과박사. 1990년~1992년 미국 Governors State University 전임강사. 1992년~1993년 미국 Governors State University 조교수. 1993년~현재 경희대학교 전자계산공학과 조교수. 관심분야는 운영체제, 실시간 컴퓨팅, 실시간 스케줄링, 멀티미디어 시스템임.



이 종 원
1985년 전북대학교 수학과 졸업. 1987년 미국 Illinois Institute of Technology 전산학과 석사. 1991년~현재 한국통신 소프트웨어 연구소 연구원. 1994년~현재 경희대학교 산업정보대학원 강사. 관심분야는 실시간 시스템, 운영체제, 데이타베이스임.



김 정 순
1982년 동국대학교 영어영문학과 졸업. 1991년 미국 Illinois Institute of Techno logy 전산학과 석사. 1993년~현재 수원대학교 전자계산학과 강사. 관심분야는 실시간 시스템, 운영체제임.