# 혼합 우선순위 시스템에서 경성 비주기적 태스크 스케쥴링 알고리즘

## (A Hard-Aperiodic Task Scheduling Algorithm in Hybrid Priority Systems)

김 형 일 † 이 승 룡 †† 이 종 원 ††† 김 정 순 ††††
(Hyungill Kim) (Sungyoung Lee) (Jongwon Lee) (Jungsoon Kim)

**요 약** 본 논문은 주기적 태스크와 경성 비주기적 태스크가 혼합된 단일 처리기 실시간 시스템 에서 중단형 (preemptive) 경성 비주기적 태스크의 스케쥴링 기법인 자유지역 지시 (Free Region Indicating: FRI) 알고리즘을 제안한다 FRI 알고리즘은 온라인에서 경성 비주기적 태스크에 대하여 할당여부를 판단하며, 주기적 태스크와 비주기적 태스크들에 대하여 고정 우선순위와 가변 우선순위를 혼합한 스케쥴링 기법으로 저자가 개발한 임계 태스크 지시 (Critical Task Indicating: CTI) 알고리즘 [4]을 확장한 것이다 CTI 알고리즘은 연성 비주기적 태스크 스케쥴링 알고리즘으로, 모의 실험 연구에 의하면 slack stealing 알고리즘 [6]보다 성능 개선을 이루었으며 특히 시스템 과부하 시에도 잘 작동하였다 FRI 알고리즘은 모든 주기적 태스크의 마감시간을 보장할 뿐 아니라 오프라인에서 작성된 CTI 테이블과 스케쥴링 변동 사항에 대한 정보를 가지고 있는 FRI 테이블을 사용하므로써 알고리즘 수행시간 복잡도를 감소시켰으며 스케쥴링 예측성도 높였다.

***Abstract*** In this paper, we present a preemptive scheduling of hard-aperiodic task, so called the Free Region Indicating (FRI) algorithm, in jointly scheduling the periodic tasks and hard-aperiodic tasks on a uniprocessor real-time system in which hard deadlines of periodic and aperiodic tasks are scheduled in such a way of mixed scheduling of a fixed and dynamic priority algorithm. The algorithm executes an on-line acceptance test for the hard aperiodic tasks and it has extended the Critical Task Indicator (CTI) algorithm [4] of which simulation study shows a considerable performance improvement over the other soft-aperiodic task scheduling algorithms, such a slack-stealing algorithm [6], especially under a heavy transient overload. The FRI algorithm is not only to guarantee all the deadline of periodic tasks, but also to reduce time complexity and improve scheduling predictability since it maintains both the CTI and FRI tables which have scheduling informations and built off-line.

## 1. 서 론

최근 실시간 시스템의 응용 분야가 가변적인 작업 환경에서도 잘 적응할 수 있는 시스템 개발 쪽으로 확대됨에 따라 비주기적 태스크 스케쥴링에 대한 관심도 증대되어 가고 있다[12],[15] 비주기적 태스크는 마감시간의 제한을 가진 경성 비주기적 태스크 (hard-deadline aperiodic task)와 마감시간의 제한이 없는 연성 비주기적 태스크 (soft-deadline aperiodic task)로 구분할 수 있는데 예측할 수 없는 환경에서 동작하는 실시간 시스템에서는 경성 비주기적 태스크 스케쥴러 개발이 중요하다.

주기적 태스크와 경성 비주기적 태스크가 혼합된 단일 처리기 시스템의 스케쥴링에 대한 연구는 크게 고정 우선순위 시스템과 가변 우선순위 시스템을 기반으로 하

본 논문은 한국과학재단 '94 핵심전문연구(과제번호 : 941-0900-046 -2)의 지원으로 작성되었음
† 준 회 원 : 경희대학교 전자계산공학과
†† 정 회 원 경희대학교 전자계산공학과 교수
††† 비 회 원 : 한국통신 소프트웨어연구원
†††† 비 회 원 수원대학교 전자계산학과 강사
논문접수 : 1994년 12월 16일
심사완료 1995년 8월 21일

여 이루어져 왔다. 최근 Ramos-Thuel과 Lehoczky [10],[11]는 연성 비주기적 태스크 스케쥴링 기법[5],[6],[13],[14]중의 하나인 slack stealing 알고리즘[6]을 확장하여 주기적 태스크의 마감시간을 보장하는 범위 내에 경성 비주기적 태스크들을 온라인상에서 스케쥴링하는 알고리즘을 개발하였다. 이는 어떻게 비주기적 태스크에게 가능한 많은 슬랙을 제공하느냐에 초점이 맞추어져 있으며, 주기적 태스크들에 대하여 마감 단조(deadline monotonic)와 같은 고정 우선순위 알고리즘을 적용하였다.

한편, 최조 마감시간 우선 (earliest deadline first: EDF) 과 같은 가변 우선순위 스케쥴링을 기반으로 한 경성 비주기적 태스크 스케쥴링 알고리즘들이 Chetto [1]와 Schwan [12]에 의해 최근 보고되었다 EDF 스케쥴링은 수행중인 태스크가 마감시간에 가까울수록 우선순위를 높게 주는 스케쥴링 기법으로써 태스크 집합의 CPU 사용율이 1보다 작거나 같으면 스케쥴가능하다 [9]. [1]에서는 가변 우선순위로 모든 주기적 태스크를 사전에 스케쥴링한 상태에서 스포래딕 태스크(sporadic task)를 가변적으로 할당하는 기법으로, 비주기적 태스크의 요청에 대하여 필요한 총 할애 시간을 계산하여 할당 여부를 판단한다. 이 때, 소요되는 시간 복잡도는 O(n)이다. 그러나 어떤 스포래딕 태스크가 시스템에 도착하여 실행되려면 반드시 사전에 할당된 모든 스포래딕 태스크의 수행을 종료시켜야 한다는 제약 조건이 있다 이러한 제약 조건을 제거한 Schwan의 알고리즘은 비주기적 태스크와 주기적 태스크를 동일하게 취급하였는데 (참고로 다른 고정 우선순위 알고리즘들은 주기적 태스크가 비주기적 태스크보다 기본적으로 우선순위가 높다고 가정하고 스케쥴링 하였음), 이는 가변적인 환경에서는 비주기적 태스크도 주기적 태스크에 못지 않은 중요한 임무를 수행할 수 있다는 면을 강조하였다. Schwan의 알고리즘은 시간 복잡도가 O(nlogn)이다 그러나, 이는 태스크가 도착 즉시 스케쥴링 가능성을 시험하여 할당 여부를 결정하는 스케쥴링 방법으로 대부분 주기적 태스크로 구성된 실시간 시스템에서는 할당 여부 판단에 소요되는 비용의 증가로 인하여 시스템 과부하를 야기시킬 수도 있다. 최근 Tia [16]는 가변 우선순위에서 스포래딕 태스크들의 요청에 대한 스케쥴링 알고리즘을 개발하였는데, 이 알고리즘에서 스포래딕 태스크들의 할당 여부를 시험하는데 소요되는 시간의 복잡도는 O(n+m)이다. 여기서, n은 주기적 태스크의 수이고, m은 스포래딕 요청의 갯수이다.

본 논문에서는 연성 비주기적 태스크 스케쥴링 기법인 CTI 알고리즘 [4]을 확장한 FRI 알고리즘을 제안한다. 제안한 알고리즘은 오프라인에서 주어진 주기적 태스크를 바탕으로 작성된 FRI 테이블의 정보와 온라인상에서 고정우선순위와 가변우선순위를 혼합하여 주기 및 경성 비주기적 태스크를 스케쥴링하는 기법이다. 이는 단일처리기 하에서 주기적 태스크와 경성 비주기적 태스크가 혼합된 그러나, 임무가 중요한 주기적 태스크가 주류를 이루는 실시간 시스템 환경에서 모든 주기적 태스크의 마감시간을 보장하고 가능한 많은 경성 비주기적 태스크를 할당할 수 있는 알고리즘이다. FRI 알고리즘은 경성 비주기적 태스크가 도착했을 때 마감시간 내에 서비스 가능한지 아닌지를 판단한다.

한편, CTI 알고리즘에 바탕을 둔 FRI 알고리즘은 스케쥴링 시 오프라인에서 주어진 주기적 태스크들에 대하여 고정우선순위로 작성된 CTI 및 FRI 테이블을 참조하므로써 슬랙을 계산하는 과정이 비교적 단순하며 테이블이 태스크들의 성질 (behavior)에 대한 대략적인 사전 정보를 가지고 있기 때문에 실시간 시스템이 지녀야 할 가장 중요한 요구 조건인 예측성을 확보할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 FRI 알고리즘의 기본이 되는 CTI 알고리즘을 간단히 소개하고, 3장에서는 FRI 알고리즘에 필요한 용어 및 정리와 단계적으로 개선된 3가지 알고리즘을 설명한 뒤 알고리즘 구현 방법 및 고려 사항에 대하여 토의하였고, 4장에서는 결론과 앞으로의 연구 방향을 논의한다

## 2. CTI 알고리즘

연성 비주기적 태스크 스케쥴링 기법인 CTI 알고리즘은 어떻게 비주기적 태스크를 위한 슬랙을 비교적 단순한 방법으로 계산할 수 있으며, 또한 어떻게 스케쥴링 예측성을 제공할 수 있을까 라는데에서 출발하였다.

연성 비주기적 태스크 스케쥴링 알고리즘으로 잘 알려진 slack stealing 알고리즘 [6]은 각 스케쥴링 지점에서 모든 우선순위 수준에 대하여 재귀 (recursive) 검색을 하므로 계산이 복잡하여 그것을 실제 시스템에 직접 적용하기에는 무리가 따르며, 온라인상에서 최대 슬랙 ($A^*$)을 구하는 과정에서 주기적 태스크 수행 조건에 대한 천정값 (ceiling values)을 고려하기 때문에 모든 가용 슬랙을 충분히 활용할 수가 없다.

이러한 제한점을 개선한 CTI 알고리즘은 주어진 주기적 태스크에 대하여 오프라인에서 고정 우선순위 [7],[8],[9]를 바탕으로 마감시간지향 사전할당 테이블(deadlinewise preassignment table)을 만들고, 고정우선순위 스케쥴링과 생성된 테이블을 참조하여 주기적

태스크의 마감시간을 보장하는 범위 내에서 비주기적 태스크가 가장 빠른 반응시간을 얻도록 온라인상에서 동적으로 스케쥴링[2]하는 방법이다.

고정 우선순위 마감시간지향 사전할당(fixed-priority deadlinewise preassignment) 이란 각각의 주기적 태스크를 주어진 우선순위에 따라 최대한 마감시간에 가깝게 사전할당하는 주기적 태스크 스케쥴링 방법을 말하며, CTI 사전할당 테이블 (CTI preassignment table) 또는 CTI 테이블이란 주어진 주기적 태스크 집합에 대한 마감시간지향·사전할당의 결과로써 각 태스크의 사전할당 정보를 가진 테이블을 말한다.

**[예제 1]** 다음 (그림 1)은 주기 $T_1=3$, $T_2=5$, 실행요구량 $C_1=1$, $C_2=2$를 각각 갖는 주기적 태스크 $\tau_1$, $\tau_2$로 구성되는 태스크 집합을 단주기 우선순위 마감시간지향 알고리즘을 적용하여 하이퍼주기($LCM(T_1, T_2, .. T_n)$) $T_h=15$까지 사전할당한 결과를 보여 준다. 스케쥴링 시작시점에 2개의 태스크 인스턴스 $\tau_{11}$과 $\tau_{21}$이 도착하여 주어진 우선순위에 따라 각각의 마감시간 $D_{11}=3$과 $D_{21}=5$에 최대한 근접하게 사전할당 되었다. 시간 t=6 시점에 태스크 인스턴스 $\tau_{13}$가 이미 사전할당되어 있던 $\tau_{22}$를 선점하게 되어 $\tau_{22}$의 앞부분이 원래의 할당 구간 [8,9]에서 [7,8]로 재배치 되었다. 이와 유사하게 t=12 시점에서도 선점이 발생한다.

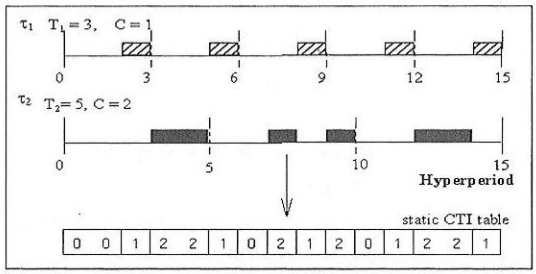


그림 1 마감시간지향 사전할당을 이용하여 만들어진 CTI 테이블

(그림 1)에서 주어진 주기적 태스크 집합에 대한 CTI 테이블은 주기적 태스크의 번호로 구성된 연결 리스트이다. (그림 1)의 CTI 테이블에서 첫 번째 00은 실제 할당되지 않은 지역으로 그 크기는 2 단위 시간이며, 두 번째 1은 $\tau_1$이 1 단위 시간만큼 할당되었다는 의미이다.

또한, CTI 테이블을 이용한 스케쥴링 시 임의의 시간 t에서 즉시 실행되지 않으면 자신의 마감시간을 보장할 수 없는 주기적 태스크를 임계 태스크 (critical task)라 한다. 참고로, CTI 테이블에 사전할당된 개개의 주기적 태스크는 온라인 스케쥴링 시 해당되는 사전할당 시점에 이르면 자동으로 임계 태스크가 된다. 그리고, CTI 테이블에서 태스크 번호가 0으로 나타나는 지역은 태스크 스케쥴링 시 비주기적 태스크의 서비스를 위하여 할당할 수 있는 영역이며 이를 슬랙 (slack)이라 한다. 슬랙에서는 비주기적 태스크가 주기적 태스크에 우선해서 서비스될 수 있다.

CTI 알고리즘은 실행 시 CTI 테이블과 고정 우선순위 알고리즘을 이용한다. CTI 테이블에서 임계 태스크를 발견하면 그 태스크를 즉시 수행시키고, 임계 태스크가 없는 상태에서 비주기적 태스크가 도착 또는 이미 존재하는 경우 비주기적 태스크를 수행시킨다. 즉, 임의의 시점에서 비주기적 태스크가 도착하면 알고리즘은 다음과 같은 2가지 조건을 검사한다.

(C1) 현재 CTI 테이블에 슬랙이 있는가?
(C2) 현재 CTI 테이블이 지시하는 임계 태스크가 이미 처리되었는가?

위의 조건들에 의해 다음의 2가지 시나리오를 고려할 수 있다.

(S1) (C1)과 (C2)중 어느 하나를 만족하면 즉시 비주기적 태스크를 서비스한다.
(S2) (C1)과 (C2)를 모두 만족하지 않으면 즉, CTI 테이블이 지시하는 임계 태스크가 처리되지 않았을 경우, 해당 임계 태스크를 서비스한다.

비주기적 태스크가 도착하지 않았을 때에는 (C1)과 관계없이 다음의 2가지 시나리오를 고려할 수 있다.

(S3) (C2)를 만족하지 않으면 임계 태스크를 서비스한다.
(S4) (C2)를 만족하면 주기적 태스크를 고정 우선순위 알고리즘에 의해 서비스한다. 만약 주기적 태스크가 존재하지 않으면 이를 유휴 시간 (idle time)으로 간주한다.

여기서, (S1),(S2),(S3)는 모두 CTI 테이블에 근거한 스케쥴링이며, (S4)는 일반 고정 우선순위 스케쥴링이다. 즉, CTI 알고리즘은 온라인 상에서 마감시간지향 사전할당 기법과 일반 고정 우선순위 알고리즘을 가

변적으로 혼합한 스케쥴링 기법이다

참고로, 이러한 알고리즘의 동적인 속성으로 인하여 모든 주기적 태스크의 마감시간을 엄격히 지키면서도 가능한 최대의 비주기적 태스크 처리를 위한 슬랙이 전체 스케쥴링 구간을 통하여 유지된다. CTI 알고리즘의 의사 코드와 이에 대한 설명, 적용 예, 적용가능성 분석, 그리고 slack stealing을 포함한 다른 연성 비주기적 태스크 스케쥴링 알고리즘들과 성능 비교를 위한 모의실험 결과 등이 [4]에 수록되어 있다.

## 3. 경성 비주기적 태스크 스케쥴링 알고리즘

본 장에서는 CTI 알고리즘을 경성 비주기적 태스크 스케쥴링 알고리즘으로 확장 시키는데 필요한 기본적인 용어와, 정리 및 가정에 대하여 기술하고, 이어서 점진적으로 개선된 3가지 알고리즘들을 각각의 실행 예를 통하여 제시한다.

### 3.1 알고리즘의 정리 및 가정

본 알고리즘의 전개를 위하여 슬랙의 용어를 다음과 같이 확장한다.

1) (C1)에 해당되는 슬랙 즉, CTI 테이블에서 태스크 번호가 0으로 표시된 영역을 정적 슬랙 (static slack)이라 하며, $t_1$에서 $t_2$까지의 정적 슬랙은 $S_{sta}(t_1, t_2)$로 표기한다 여기서, 현재 시간을 t라 할 때 $t \le t_1 < t_2$이다.

2) (C2)에 해당되는 슬랙 즉, CTI 테이블의 임계 태스크가 이미 처리되어 비주기적 태스크를 할당할 수 있는 지역을 동적 슬랙 (dynamic slack)이라 하며, $t_1$에서 $t_2$까지의 동적 슬랙은 $S_{dyn}(t_1, t_2)$로 표기한다

위의 정적 슬랙과 동적 슬랙의 개념을 통합하여 자유지역이라는 새로운 개념을 정의하자.

**[정의 1]** 주기적 태스크의 마감시간을 보장하는 범위내에서 비주기적 태스크에 가장 높은 우선순위를 부여할 수 있는 지역을 자유지역 (free region)이라 하며, $t_1$에서 $t_2$까지의 자유지역 (비주기적 태스크에 대한 서비스 가능 시간)을 $F(t_1,t_2)$로 표시한다.

CTI 테이블에서 태스크 번호가 0으로 표시된 부분은 정적 슬랙으로 좁은 의미의 자유지역이며, (C1)은 만족하지 않고 (C2)를 만족하는 경우 (CTI 테이블에서 태스크 번호가 0으로 표시되지는 않았지만 이미 처리된 임계 태스크인 경우)도 비주기적 태스크를 할당할 수 있는 동적 슬랙으로 넓은 의미의 자유지역이라 할 수 있다 따라서, 자유지역 $F(t_1,t_2)$는 정적 슬랙 $S_{sta}(t_1, t_2)$과 동적 슬랙 $S_{dyn}(t_1, t_2)$의 합과 같다.

$$F(t_1, t_2) = S_{sta}(t_1, t_2) + S_{dyn}(t_1, t_2) \quad (1)$$

이와 같이 슬랙의 개념에서 자유지역의 개념으로 범위를 확장하는 이유는 (C1)과 (C2)를 하나의 조건으로 통일하여 비주기적 태스크 할당에 대한 명확한 조건을 제시하기 위함이다

또한, 자유지역은 할당 가능성에 따라서 다음과 같이 구분된다.

3) 어떤 시점 이전에 도착한 비주기적 태스크를 위하여 할당된 자유지역을 예약된 자유지역 (reserved free region)이라 하며, $t_1$이전에 도착한 경성 비주기적 태스크를 위하여 $t_1$에서 $t_2$까지 할당된 자유지역을 $F_{rev}(t_1,t_2)$로 표기하며, $t_2$를 명시하지 않았을 때는 $F_{rev}(t_1)$로 표기한다.

4) 예약된 자유지역을 제외한 자유지역으로 현재 시점에 도착한 비주기적 태스크를 위하여 할당할 수 있는 지역을 할당 가능한 자유지역 (available free region)이라 하며, 에서 까지 할당 가능한 자유지역을 로 표기한다

한편, 본 알고리즘의 전개에 부가적으로 필요한 표기법 (notation)은 다음과 같다.

- $P(0,t)$ : 시간 0에서 현재 시간 까지 주기적 태스크 총 실행 시간
- $A(0,t)$ : 시간 0에서 현재 시간 까지 비주기적 태스크 총 실행 시간
- $I(0,t)$ : 시간 0에서 현재 시간 까지의 총 유휴시간
- $P(t_1,t_2)$ : 시간 에서 시간 까지의 주기적 태스크 실행 시간
- $U_p$ : 주기적 태스크 집합의 사용률 (utilization)

편의상, 다음의 정리 1과 정리 2에서 사용할 목적으로 임의의 시간에 CTI 알고리즘의 작동을 4가지 경우로 분류한다

(경우 1) 현재의 자유지역이 비주기적 태스크를 서

비스하는데 사용되었다.

(경우 2) 현재 CTI 테이블의 임계 태스크를 서비스하였다.

(경우 3) 고정 우선순위로 주기적 태스크를 서비스하였고, 현재의 자유지역은 동적 슬랙으로 이동하였다.

(경우 4) 서비스할 주기적, 비주기적 태스크가 없으므로 현재의 자유지역은 유휴시간으로 처리되었다.

**[정리 1]** 현재의 자유지역은 (경우 3)에 의해서 동적 슬랙으로 이동될 수 있으며, 이로 인하여 주기적 태스크의 스케쥴링 가능성은 영향을 받지 않는다. 이때, 슬랙이 이동되는 시점은 현재 서비스된 $\tau_U$의 마감시간 $D_U$안에 CTI 테이블이 보장하고 있는 한 시점이며, 이동 범위는 $t < t_x < D_U$이다.

$$F(t) \rightarrow S_{dyn}(t_x) \tag{2}$$

**증명**: 전체 하이퍼주기 안에서의 모든 스케쥴링 상황은 다음과 같이 표현할 수 있다.

$$\begin{aligned} T_h &= \underline{P}(0,t)+\underline{A}(0,t)+\underline{I}(0,t) \\ &\quad +P(t,T_h)+F(t,T_h) \\ &= \underline{P}(0,t)+P(t,T_h)+\underline{A}(0,t) \qquad (3) \\ &\quad +\underline{I}(0,t)+F(t,T_h) \\ &= U_p\times T_h+(1-U_p)\times T_h \qquad (4) \\ &= (U_p+1-U_p)\times T_h \\ &= T_h \end{aligned}$$

스케쥴링 가능성이 확인된 주기적 태스크 집합이 존재할 때, 현재 시간을 기준하여 이전 시간과 이후 시간으로 구분한 식 (3)에서 주기적 태스크의 실행시간과 그 나머지 시간으로 구분한 식 (4)를 얻을 수 있으며, 식 (4)에서 하이퍼주기 안에서 주기적 태스크의 총 서비스 시간은 고정되어 있으며, 총 자유지역은 비주기적 태스크의 서비스 시간 또는 유휴시간으로 처리된다 따라서, 슬랙 시점에서 주기적 태스크가 마감시간 이전에 서비스되고, 자유지역이 이동된다 하더라도 다른 주기적 태스크의 마감시간을 보장하지 못하는 경우는 발생하지 않으므로 전체 스케쥴링 가능성에는 영향을 미치지 않는다. ■

**[정리 2]** 예약된 자유지역 $F_{rev}(t_1,t_2)$을 포함하고 있는 스케쥴링 영역내의 사전할당된 모든 태스크는 CTI 테이블에 의해서만 스케쥴링된다.

**증명**: 예약된 자유지역에서는 비주기적 태스크가 대기하고 있으므로 고정 우선순위로 스케쥴링되는 (경우 3)을 만족하지 못하고 항상 (경우 1) 또는 (경우 2)의 조건에 해당되므로 모든 태스크는 오직 CTI 테이블에 의해서만 스케쥴링된다. ■

예약된 자유지역이 CTI 테이블에 의해서만 스케쥴링된다는 것은 허가된 비주기적 태스크와 사전할당된 주기적 태스크의 마감시간이 모두 보장되어 있으며 영역내의 총 사용률이 100%라는 의미이다.

지금까지 전개한 기본 개념을 바탕으로 할당 가능한 자유지역을 다음과 같은 식으로 정리할 수 있다.

$$\begin{aligned} F_{ava}(t_1,t_2) &= F(t_1,t_2)-F_{rev}(t_1,t_2) \\ &= S_{sta}(t_1,t_2)+S_{dyn}(t_1,t_2)-F_{rev}(t_1,t_2) \end{aligned} \tag{5}$$

위의 식은 $[t_1,t_2]$내에서 할당 가능한 자유지역 $F_{ava}(t_1,t_2)$을 나타내는 것으로 $t_1$ 이전에 도착한 경성 비주기적 태스크에 이미 할당되어 있는 영역을 제외한 자유지역을 말한다. $t_1$에 도착한 경성 비주기적 태스크가 마감 시간이 $t_2$이고, 실행시간이 $C_a$인 경우, 마감시간까지 할당 가능한 자유지역을 의미하며 $F_{rev}(t_1,t_2) \geq C_a$인 경우 할당 가능한 비주기적 태스크가 된다. 그렇지 않으면 마감시간을 보장하지 못하므로 비주기적 태스크는 거부된다 또한, 경성 비주기적 태스크가 도착하여 할당이 허가될 때에는 이미 허가된 비주기적 태스크는 반드시 마감시간을 보장할 수 있어야 한다. 현재 도착한 비주기적 태스크를 마감시간 안에 서비스 가능하다고 하더라도 이 때문에 이미 허가된 다른 비주기적 태스크의 마감시간을 보장할 수 없다면 현재의 비주기적 태스크는 당연히 거부된다. 따라서, 할당 가능 자유지역을 정확하게 산출하는 방법이 가장 중요한 문제이다

한편, 본 논문에서 사용하는 가정은 다음과 같다.

(가정 1) 임의의 주기적 태스크 인스턴스를 위한 마감시간은 다음 인스턴스의 도착 시간과 같다.

(가정 2) 주기적 또는 비주기적 태스크에 대한 선점은 언제나 가능하다.

(가정 3) 모든 주기적, 비주기적 태스크의 문맥 교환 비용은 각각 대응되는 주기적, 비주기적 태스크의 실행 시간에 포함된다.

(가정 4) 모든 주기적 태스크의 실행 시작시간은 0이다.

(가정 5) 태스크의 도착시간, 실행시간, 주기, 비주기적 태스크의 도착시간 및 실행시간은 모두 스케쥴링 단위시간의 정수배이다.

(가정 6) 모든 비주기적 태스크는 경성 비주기적 태스크로 도착 시 마감 시간과 실행 시간을 알 수 있다.

이와 더불어 본 논문에서는 다음과 같은 두 가지 스케쥴링 정책을 채택하였다. 첫째, 온라인에서 비주기적 태스크의 할당 여부 판단은 도착 즉시 이루어지는 것을 원칙으로 한다. 이는 도착한 비주기적 태스크가 실제 할당되지 못함에도 불구하고 불필요하게 시스템에서 대기하는 것을 방지하기 위함이다. 둘째, 할당이 허가된 비주기적 태스크는 FIFO 순서로 서비스 된다.

### 3.2 알고리즘 설명

본 절에서는 점진적으로 개선된 세가지 형태의 경성 비주기적 태스크 스케쥴링 알고리즘을 제시하고 각각에 대하여 기술한다.

#### 3.2.1 알고리즘 1

할당 가능한 자유지역을 산출하는 첫번째 방법은 온라인에서 경성 비주기적 태스크가 도착하면 예약된 자유지역 이후부터 마감시간까지 CTI 테이블을 이용하여 자유지역을 검색하는 것이다. 이 때 CTI 테이블상에서 0으로 나타난 정적 슬랙뿐만 아니라 동적 슬랙도 할당 가능한 자유지역에 포함시킨다. 할당가능한 자유지역 산출에 대한 알고리즘 의사 코드는 (그림 2)와 같다.

```
1  initialize S_sta and S_dyn
2  repeat
3        if (CTI table indicates 0)
                then increment S_sta
4        else if (CTI table indicates already serviced task)
                then increment S_dyn
5        advance search pointer
6  until (deadline of the given aperiodic task)
7  return (S_sta+ S_dyn)
```

그림 2 알고리즘 1: 할당 가능한 자유지역 산출의 의사 코드

(그림 2)의 3행과 4행에서 정적 슬랙과 동적 슬랙을 구분하여 자유지역을 원하는 시간까지 구하게 된다. 이와 같은 방법으로 현재 도착한 비주기적 태스크에게 할당 가능한 자유지역을 산출할 수 있다.

**[예제 2]** 두개의 주기적 태스크 $\tau_1$( $T_1=3, C_1=1$), $\tau_2$ ( $T_2=5, C_2=2$)로 이루어진 태스크 집합이 있다. 이 태스크 집합에 대한 정적 CTI 테이블은 (그림 3)과 같다. 마감시간이 7이고 실행시간이 3인 첫번째 비주기적 태스크가 t=3에 도착했고, 마감시간이 8이고 실행시간이 3인 두번째 비주기적 태스크가 t=6에 도착했을 때, 이들 경성 비주기적 태스크들이 어떻게 처리되는지 살펴보자.

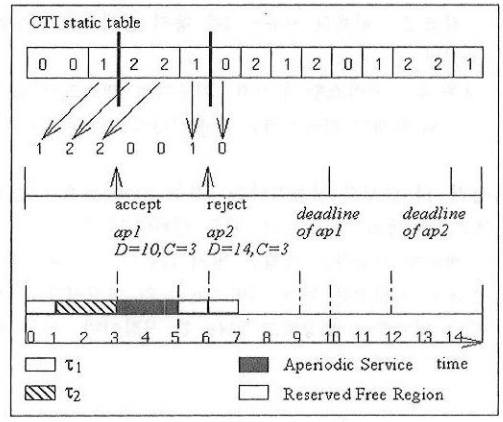


그림 3 알고리즘 1을 이용한 자유지역의 계산

스케쥴링이 시작되면, 마감시간지향 CTI 테이블에 사전할당된 정적 슬랙이 나타나므로 비주기적 태스크를 즉시 서비스할 수 있지만, 이미 도착한 비주기적 태스크가 없으므로 주기적 태스크를 서비스하게 된다.

| t | CTI 테이블 | 도착 태스크 | 처 리 방 법 |
|---|---|---|---|
| 0 | 정적 슬랙 | $\tau_{11},\tau_{12}$ | 주기적 태스크 완결 (Fixed-Prority Scheduling) |
| 1 | 정적 슬랙 | | 주기적 태스크 부분 실행 (Fixed-Prority Scheduling) |
| 2 | 동적 슬랙 | | 주기적 태스크 완결 (Fixed-Prority Scheduling) |
| 3 | 동적 슬랙 | $ap_1,\tau_{12}$ | 자유지역 산출, 허가 및 부분 실행 (CTI) |
| 4 | 동적 슬랙 | | 부분 실행 (CTI) |
| 5 | 임계태스크 발생 | | 임계 태스크 실행, 완결 (CTI) |
| 6 | 정적 슬랙 | $ap_2$ | 자유지역 산출, 거부, 완결 (CTI) |

그림 4 알고리즘 1에 의하여 t=0에서 t=6까지 스케쥴링되는 과정

t=3에서 첫번째 비주기적 태스크가 도착하였으며 알고리즘 1을 이용하여 마감시간까지 CTI 테이블을 검색하면서 (그림 4)처럼 정적 슬랙과 동적 슬랙을 구했다. 그 결과 첫번째 비주기적 태스크를 마감시간을 보장할 수 있으므로 완료시간인 t=6까지는 CTI 테이블에 의하여 스케쥴링 되었다. t=6에는 두번째 비주기적 태스크

가 도착하였고 자유지역을 산출한 결과 마감시간을 보장할 수 없으므로 두번째 비주기적 태스크는 거부되었다. 참고로 (그림 4)의 결과는 비주기적 태스크의 도착 여부에 따라 온라인에서 결정되는데 그 이유는 스케줄링 시 슬랙의 이동이 동적으로 발생하기 때문이다.

이와 같이 알고리즘 1에서는 비주기적 태스크가 도착되면 CTI 테이블상에서 정적 슬랙과 동적 슬랙을 구별하고 이를 마감시간까지 검색하여 비주기적 태스크의 허가 또는 거부를 판단해야 한다. 그러나, CTI 테이블의 선형 검색만으로는 동적 슬랙을 산출하지 못하므로 이를 더욱 단순화시켜 선형 검색으로 스케줄링 가능성을 판단할 수 있는 방법을 찾기로 한다.

3.2.2 알고리즘 2

알고리즘 2는 알고리즘 1의 정적 슬랙과 동적 슬랙에 대한 이원적인 검색을 제거하기 위하여 새로운 동적 CTI 테이블을 도입한다. 동적 CTI 테이블이란 정적 CTI 테이블을 복사한 것으로 자유지역의 이동이 발생할 때 마다 갱신하며 선형 검색만으로도 자유지역을 산출할 수 있게 변형된 테이블이다.

알고리즘 2의 의사 코드는 (그림 5)와 같다.

```
1  initialize data structures  (static CTI table,  dynamic CTI table)
2  loop begin
3     if (hard aperiodic task(s) is arrived) then
          if (F_ava(t, t + D_a) >= C_a) then accept it
          else reject it
4     if (dynamic CTI table do not indicate 0)
          service periodic task by dynamic CTI table
      else
5         if (hard aperiodic task(s) is ready) then service it
          else if (periodic task(s) is ready or arrived ) then service it
                    and move the slack in dynamic CTI table
               else process CPU idle state
6  advance timer (t)
7  if ((t MOD hyperperiod) is equal to zero)
       then reinitialize the global parameters and dynamic CTI table
8 end loop
```

그림 5 알고리즘 2: 동적 CTI 테이블을 이용한 알고리즘의 의사 코드

(그림 5)의 3행은 경성 비주기적 태스크가 도착하였을 때 마감시간까지 할당 가능한 자유지역의 크기를 산출하여 허가 또는 거부를 판단하는 것이고, 4행은 동적 CTI 테이블이 자유지역을 나타내지 않으면 동적 CTI 테이블에 의하여 주기적인 태스크를 서비스하고, 5행에서 비주기적 태스크가 존재하면 이를 서비스하지만 그렇지 않으면 고정 우선 순위로 주기적인 태스크를 서비스함과 동시에 슬랙은 다른 곳으로 이동되게 된다.

**[예제 3]** 예제 2와 동일한 상황에서 동적 CTI 테이블을 도입하여 스케줄링 하였을 때 온라인에서 동적 CTI 테이블이 변경되는 과정을 살펴보기로 하자.

t=0 시점에서 CTI 테이블은 비주기적 태스크가 도착하지 않았기 때문에 주기적 태스크가 서비스되고 해당하는 정적 슬랙이 이동하여 실행된 주기적 태스크가 보장되어 있던 t=2 시점이 0으로 표시되었고, 슬랙의 이동이 발생하였다. 이와 마찬가지로 t=1 시점의 슬랙이 t=3 시점으로 이동하여 동적 CTI 테이블상의 t=3 시점이 0으로 변경되었다. 마지막으로 t=2 시점의 동적 슬랙이 다시 t=4 시점으로 이동이 발생되어 동적 CTI 테이블의 t=4 시점이 0으로 변경되었다.

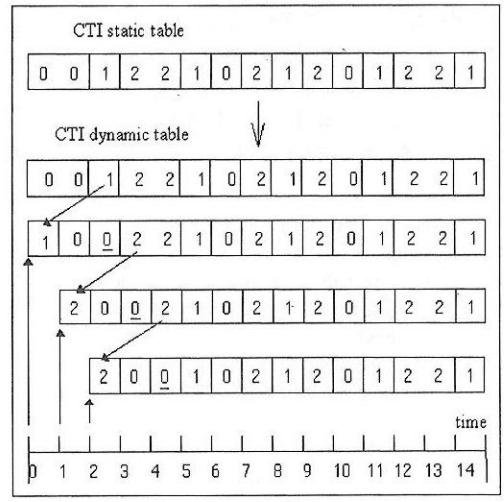


그림 6 알고리즘 2: 동적 CTI 테이블의 갱신

이와 같이 동적 CTI 테이블을 도입한 알고리즘 2는 정적 슬랙과 동적 슬랙의 구분 없이 0으로 표시되므로 선형 검색으로 비주기적 태스크의 허가 또는 거부를 판단할 수 있다.

3.2.3 알고리즘 3: FRI 알고리즘

알고리즘 2의 단점은 (그림 6)에서 볼 수 있듯이 사전에 만들어진 정적 CTI 테이블과 온라인 상에서 만들어지는 동적 CTI 테이블과의 정보의 중복으로 인하여 기억 장소의 낭비를 초래한다는 점이다. 따라서, 이같은 중복성을 제거한 새로운 형태의 동적 테이블을 다음과 같이 도입한다.

**[정의 2]** CTI 테이블의 정보를 이용하여 동적으로 생성되는 자유지역과 금지지역 (Forbidden Region)으로 구분되는 비트맵 테이블을 동적 FRI 비트맵 테이블

(dynamic FRI bitmap table) 또는 이를 간단히 FRI 테이블 (FRI table)이라고 한다. 참고로, 동적 슬랙은 CTI 테이블상에서 0으로 표시되지 않는 지역이므로 온라인에서 동적으로 자유지역이 갱신되는 FRI 테이블과는 의미가 다르다. 여기서 말하는 금지 지역이란 비주기적 태스크를 할당 할 수 없는 (비주기적 태스크에게 할당이 금지된) 지역을 말한다.

FRI 테이블은 주기적 태스크와 비주기적 태스크를 구별하기 위하여 자유지역과 금지 지역으로 나누어지는 비트맵 테이블로 현재 비주기적 태스크의 서비스 가능한 자유지역을 나타내는 테이블이며 온라인에서 자유지역 이동이 발생했을 경우 해당되는 위치로 자유지역을 이동시킨다. 동적 FRI 테이블에서 0은 비주기적 태스크를 위해 할당할 수 있는 지역이고, 1은 주기적 태스크를 할당해야 하는 지역 (비주기적 태스크에 대한 금지 지역)이며 구체적인 주기적 태스크 정보는 대응되는 정적 CTI 테이블을 이용하여 얻을 수 있다. FRI를 이용한 알고리즘 3은 (그림 7)과 같다.

```
1  initialize data structures  (CTI, FRI, ...)
2  loop begin
3    if (hard aperiodic task(s) is arrived) then
       if (F_ava(t, t + D_a) >= C_a) then accept it
       else reject it
4    if (FRI table indicate 1)
       service periodic task by CTI table
     else
5      if (hard aperiodic task(s) is ready) then service it
       else if (periodic task(s) is ready or arrived ) then
            service it and move the slack in FRI table
          else
            process CPU idle state
6  advance timer (t)
7  if ((t MOD hyperperiod) is equal to zero)
     then reinitialize the global parameters and FRI table
8 end loop
```

그림 7 알고리즘 3 (FRI 알고리즘)의 의사 코드

알고리즘 3은 알고리즘 2와 비슷하나 FRI 테이블에 의하여 자유지역이 아니라고 판별될 경우 CTI 테이블에 의하여 주기적인 태스크를 서비스한다. 또한, 서비스할 비주기적인 태스크가 없을 경우에는 고정 우선순위로 서비스하고 이동된 슬랙에 의하여 FRI 테이블은 변경된다. 이와 같이 FRI 테이블의 특징을 이용한 경성 비주기적 태스크 스케줄링 알고리즘을 FRI 알고리즘이라 하며, 다음과 같은 두 가지 특징을 가진다.

첫째, FRI 테이블이 자유지역을 지시하나 일반 고정 우선순위 스케줄링에 의하여 주기적 태스크가 할당되면 자유지역의 이동 (free region transition)이 일어난다. 자유지역의 이동이 발생하면 FRI 테이블은 갱신되는데 이는 자유지역이 보존됨을 의미하며 이때, FRI 테이블은 자유지역을 0으로 지시한다. 둘째, FRI 테이블은 온라인에서 항상 자유지역과 금지 지역으로 구분된다. CTI 테이블에서는 0인 정적 슬랙과 0은 아니지만 (C2) 조건에 해당되는 동적 슬랙으로 자유지역을 구분하였으나 FRI 테이블에서는 정적 슬랙과 동적 슬랙 모두 0으로 나타나며 주기적 태스크가 사전할당되어 있는 금지 지역은 1로 나타낸다.

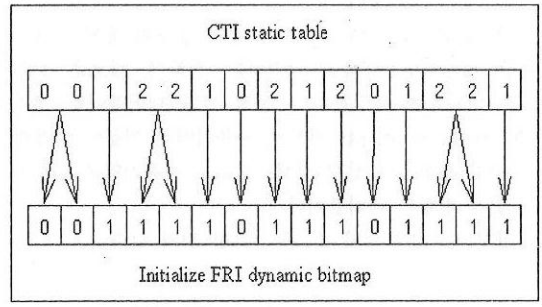


그림 8 CTI 테이블로 부터 FRI 테이블의 생성

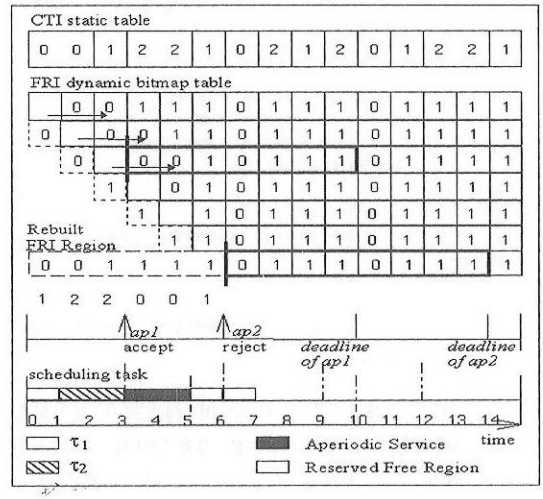


그림 9 FRI 알고리즘의 적용 예

FRI 알고리즘에서 사용하는 동적 FRI 비트맵 테이블을 생성하는 방법은 (그림 8)과 같다. 이렇게 만들어진

FRI 테이블은 온라인 상에서 자유지역의 이동이 발생할 때 마다 갱신되며, FRI 테이블을 이용하여 현재의 시점이 금지 지역에 해당되는지 자유지역에 해당되는지 판단하고 각각의 태스크를 서비스하게 된다.

**[예제 3]** 예제 1에 FRI 알고리즘을 적용한 스케쥴링의 동작을 살펴보자. FRI 테이블의 생성은 이미 (그림 8)에 나타나 있으며 이를 적용한 예제 1의 스케쥴링 동작은 (그림 9)에서 보여 주고 있다.

초기화 시 CTI 테이블과 FRI 테이블이 만들어지고 실제 스케쥴링이 진행되면 t=0, 1, 2까지 자유지역의 이동이 발생하였고 그에 따라 FRI 테이블이 갱신되었다 t=3, t=6에서 즉시 FRI 테이블을 선형 검색하였으며 도착한 비주기적 태스크에 대하여 허가 또는 거부되는 것을 알 수 있다.

### 3.3 FRI의 구현 방법

CTI 알고리즘은 (C1)과 (C2)를 현재 시간에서만 계산하여 결정하기 때문에 이동된 자유지역을 정확하게 산출하기 어렵다. 그러나, FRI 알고리즘은 스케쥴링 시 이동되는 자유지역의 위치를 정확하게 FRI 테이블에 유지시키고 실제 비주기적 태스크를 서비스할 때는 단지 FRI 테이블의 자유지역 크기에 따라 경성 비주기적 태스크를 서비스할 수 있는지 여부를 간단하게 결정할 수 있는 알고리즘이다.

3.3.1 FRI 테이블의 초기화

FRI 테이블은 CTI 테이블과 함께 오프라인에서 만들어지며 CTI 테이블의 정보를 근거로 하여 자유지역과 금지 지역으로 구분되는 비트맵 테이블로 초기화된다. 그리고, 자유지역의 이동이 발생할 경우 대응되는 시점을 알 수 있도록 주기적 태스크의 위치를 지시하고 있는 포인터를 CTI 테이블의 각 태스크 시작 위치에 둔다. 이 때, 자유지역 이동의 발생에 대응되는 시점을 지시하고 있는, CTI 테이블상에 존재하는 각각의 주기적 태스크 포인터를 periodic task pointer (PTP)라 한다. 참고로, PTP는 현재 자유지역 이동에 대응되는 CTI 테이블의 정보를 이용해 얻을 수 있는 이동 시점을 나타내며 주기적 태스크가 할당될 때 마다 해당 PTP가 다음 사전할당 지역으로 이동된다. 예를 들면, (그림 9)에서 τ1의 PTP는 t=2 시점이며, τ2의 PTP는 t=3 시점이다.

3.3.2 FRI 테이블의 재사용

FRI 테이블은 동적 테이블이기 때문에 온라인에서 변경된다. 스케쥴링의 특성상 하이퍼주기를 기준으로 테이블의 정보는 반복되므로 스케쥴링이 끝난 FRI 테이블은 다음 하이퍼주기의 스케쥴링을 위하여 원래의 FRI 테이블로 복구되어야 한다. 이때, 동적 FRI 테이블의 스케쥴링이 끝난 영역을 CTI 테이블의 정보로 다시 복구하게 되며 이 영역을 재전 FRI 영역 (rebuilt FRI region)이라고 한다 ( (그림 9) 참조).

### 3.4 스케쥴링 가능성 분석 및 토의

주기적 태스크와 비주기적 태스크가 혼합된 환경에서 지금까지 개발된 경성 비주기적 태스크 스케쥴링 알고리즘은 크게 고정 우선순위[11]와 가변 우선순위[12]로 구분할 수 있다. 고정 우선순위 알고리즘은 모든 주기적 태스크의 마감시간을 보장하는 범위에서 경성 비주기적 태스크가 도착하였을 때 이를 선별적으로 처리하는데 목적이 있으며 가변 우선순위 알고리즘은 온라인 상에서 마감시간을 보장할 수 있는 태스크를 선별적으로 스케쥴링 하는데 그 목적이 있다[2]. 따라서, 이 두 가지 스케쥴링은 기본 철학이 다르기 때문에 성능을 단순 비교하기는 어렵다.

FRI 알고리즘은 엄밀히 말하면 혼합 우선순위 알고리즘이나 크게 본다면 고정 우선순위 영역에 가까운 알고리즘이다. 고정 우선순위 경성 비주기적 태스크 스케쥴링 알고리즘을 서로 비교할 수 있는 기준은 다음과 같다.

1) 도착한 경성 비주기적 태스크의 마감시간을 보장하기 위한 자유지역 (비주기적 태스크를 위한 실행시간)의 확보 여부
2) 도착한 경성 비주기적 태스크의 허가 또는 거부를 결정하기 위한 시간 복잡도

1)은 자유지역을 더 많이 확보할 수 있는 스케쥴링 알고리즘이 더 많은 경성 비주기적 태스크를 서비스할 수 있다는 의미이고, 2)는 비주기적 태스크의 허가 또는 거부를 결정하는 알고리즘의 시간 복잡도가 크면 그 만큼 구현 과정의 오버헤드가 증가한다는 의미이다.

FRI 알고리즘에서 구할 수 있는 자유지역의 크기는 CTI 알고리즘에서 구할 수 있는 자유지역과 동일하며, [11]에서 구하는 경성 비주기적 태스크를 위한 슬랙은 slack stealing [6] 알고리즘에서 구할 수 있는 슬랙의 양과 동일하다. 그런데, 앞서 언급했듯이 연성 비주기적 태스크 스케쥴링의 모의 실험을 통한 상호 비교 [4]에 의하면 CTI 알고리즘이 slack stealing 알고리즘보다 통계적으로 마감시간 내에서 자유지역을 더 많이 확보할 수 있으므로 결론적으로 FRI 알고리즘이 [11]보다 더 많은 경성 비주기적 태스크를 허가할 수 있다.

한편, [11]에서는 경성 비주기적 태스크를 허가 또는

거부를 결정하기 위한 시간 복잡도가 n개의 주기적 태스크에 대하여 최선의 경우 이고 최악의 경우 인 것과는 달리 FRI 알고리즘은 주기적 태스크의 갯수와 상관없이 FRI 테이블을 선형 검색하는 시간 복잡도만이 관여한다. FRI 테이블을 검색하는데 필요한 시간 복잡도는 다음과 같다.

$$C_a \le T_s \le D_a$$

즉, 일반적으로 선형 검색 영역 $T_s$는 실행시간 만큼만 선형 검색하고 허가되는 최선의 경우와, 마감시간까지 선형 검색하고도 거부되는 최악의 경우 사이에 존재하므로 경성 비주기적 태스크의 실행시간과 마감시간에 따라 시간 복잡도가 달라진다.

FRI 알고리즘을 이용하여 스케쥴링 할때 가능한 모든 상태표를 (그림 10)과 같이 얻을 수 있다. FRI 비트맵 테이블의 상태는 항상 0 또는 1이며, CTI 테이블의 상태는 0 혹은 i (i는 주기적 태스크 중 하나)이다.

| FRI | CTI | 상태 | 비주기적 태스크가 있는 경우 | 비주기적 태스크가 없는 경우 |
|---|---|---|---|---|
| 0 | 0 | $S_{sta}$ | 비주기적 태스크 실행 | Fixed-Priority Scheduling |
| 0 | i | $S_{dyn}$ | 비주기적 태스크 실행 | Fixed-Priority Scheduling |
| 1 | i | $P_i$ | 임계 태스크 실행 | 임계 태스크 실행 |
| 1 | 0 | N/A | | |

그림 10 FRI 알고리즘의 스케쥴링 상태표

FRI 알고리즘에서 상태는 자유지역 혹은 임계 태스크 지역으로 구분된다. 예약 자유지역에 비주기적 태스크가 있으면 비주기적 태스크를 서비스하고 비주기적 태스크가 없으면 고정 우선순위 알고리즘을 이용하여 주기적 태스크를 서바스하는데 이때 자유지역의 이동이 발생한다.

경성 비주기적 태스크 스케쥴링의 특징 중의 하나는 스케쥴링 시 하이퍼 주기의 반복 특성을 사용할 수 없다는 점이다. FRI 알고리즘에서는 이를 극복하기 위하여 CTI 테이블을 환형 연결 리스트로 사용한다. 만약 도착한 경성 비주기적 태스크의 마감시간이 하이퍼주기보다 큰 경우에는 다음의 식을 만족하면 서비스가 가능하다.

$$\lfloor \frac{D_a}{T_h} \rfloor \times (1-U_p) \times T_h + F(t, t+(D_a \text{ MOD } T_h)) - F_{res}(t) \ge C_a$$

위의 식은 하이퍼주기의 정수배에 해당하는 영역과 비주기적 태스크 할당을 위한 이용율의 곱을 나머지 할당 가능한 자유지역에 더한 뒤, 그 값이 실행시간보다 크거나 같으면 도착한 경성 비주기적 태스크는 허가됨을 의미한다.

한편, 현재 도착한 비주기적 태스크가 이미 도착한 다른 비주기적 태스크보다 마감시간이 항상 긴 것은 아니다. 따라서, 현재 도착한 비주기적 태스크가 이미 도착한 다른 비주기적 태스크보다 마감시간이 짧은 경우를 중첩(overlapping)이라 하며, 이 경우에는 다음과 같은 조건이 만족해야만 허가할 수 있다.

이미 n개의 중첩된 비주기적 태스크가 허가되었다면 다음의 식을 만족해야 한다.

$$F(Co_{ai}, D_{ai}) \ge C_a, \forall\{i | D_a \le D_{ai}\}$$

(여기서, $Co_{ai}$는 허가된 비주기적 태스크의 예정되어 있는 완료시간이다.)

즉, 중첩이 이미 일어난 경우 도착한 비주기적 태스크보다 마감시간이 긴 현재의 비주기적 태스크는 상대적으로 이미 확보해 놓은 실행시간을 도착한 비주기적 태스크에게 할당해 주어야 하므로 그 실행 시간만큼 뒤로 밀리게 되며, 이로 인하여 마감시간을 보장하지 못하는 태스크가 존재할 경우는 거부된다.

## 4. 결론 및 앞으로의 연구 방향

CTI 알고리즘을 확장한 FRI 알고리즘은 경성 비주기적 태스크가 도착했을 때 비트맵 FRI 테이블을 선형 검색하여 마감시간 내에 스케쥴링 가능성 여부를 판단한다. FRI 테이블을 이용하면 임계 태스크 서비스 여부를 판단할 수 있고 스케쥴링 상황을 예측할 수 있다. CTI 테이블이 정적인 테이블로 주기적 태스크의 서비스를 위해 만들어진 반면, FRI 테이블은 이를 기반으로 하여 만들어진 동적 테이블로 자유지역의 계산과 온라인 상에서 변경되는 스케쥴링 정보를 제공해 준다.

이는 오프라인에서 주기적 태스크들에 대한 사전 정보를 갖고 있는 FRI 테이블을 이용하여 온라인 상에서 고정 우선순위와 가변 우선순위를 혼합하여 주기적, 경성 비주기적 태스크를 스케쥴링하는 방법으로 실시간 시스템의 가장 중요한 요구 사항인 예측성을 확보할 수 있다. 또한, CTI 알고리즘에 기반을 둠으로써 다른 고정 우선순위 경성 비주기적 태스크 스케쥴링 알고리즘 [11]에 비해 비주기적 태스크들에 대한 서비스 율도 상대적으로 높다.

한편, 저자는 CTI 알고리즘에 바탕을 두고 경성 비주기적 태스크의 허락 여부의 시험 방법을 달리한 경성 비

주기적 태스크 스케줄링 알고리즘[3]을 최근 개발하였으며, 향후 마감시간 지향 사전할당 기법을 가변 우선순위 비주기적 태스크 스케줄링 알고리즘에 확대 적용하는 방법과 이를 실제 시스템에 응용하는 방법에 대한 연구를 진행할 예정이다.

## 참 고 문 헌


[1] H. Chetto and M Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm", IEEE Transactions on Software Engineering, Vol. 15 (10), pp. 466-473, 1989.

[2] K.S. Hong and J Y-T Leung, "On-line Scheduling of Real-Time Tasks", Proceedings of the IEEE Real-Time System Symposium, pp. 244-250, December 1988.

[3] J.W.Lee, S.Y.Lee, and H I Kim, "Scheduling Hard-Aperiodic Tasks in Hybrid Static/Dynamic Priority Systems", ACM SIGPLAN on Languages, Compilers, and Tools for Real-Time Systems, La Jolla, CA, pp. 7-19, June 1995.

[4] 김형일, 이승룡, 이종원, 김정순, "실시간 환경하에서 혼합 우선순위 할당에 의한 비주기적 태스크 스케줄링 알고리즘", 한국정보과학회 논문지 제 22권 제 5 호, pp. 748-758, 1995.

[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] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] J.A. Stankovic and K. Ramamrithm, "What is Predictability for Real-Time System?", The International Journal of Time-Critical Computing Systems Vol. 2, No 4, pp 247-254, November 1990.



김 형 일
제 22권 5호 참조

이 승 룡
제 22권 5호 참조

이 종 원
제 22권 5호 참조

김 정 순
제 22권 5호 참조