선택적 우선순위 알고리즘: 가변 우선순위 시스템에서 비주기적 태스크 스케쥴링

(An Alternative Priority Scheduling Algorithm: A Soft-Aperiodic Task Scheduling in Dynamic Priority Systems)

김 형 일 † 이 승 룡 †† 이 종 원 †††

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

요 약 슬랙 스틸링 기반 비주기적 태스크 스케쥴링은 슬랙 계산의 복잡도로 인하여 실용적이지 못하다는 문제점을 가지고있다. 이를 개선하기 위하여 본 논문에서는 가변우선순위 시스템 하에서 단순하게 슬랙을 계산할 수 있으며 임계 태스크 지정 (CTI: Critical Task Indicating) 스케쥴링 틀(framework)에서 최적인 선택적 우선순위 (APS: Alternative Priority Scheduling) 연성 비주기적 태스크 스케쥴링 알고리즘을 제안한다 제안한 알고리즘은 저자가 개발한 EDF-CTI (Earliest Deadline First-Critical Task Indicating) 알고리즘을 확장한 것으로, 오프라인에서 생성한 예측성을 가진 CTI 테이블을 참조하여 온라인에서 우선순위 구동 스케쥴링인 최조 마감 우선순위 (EDF: Earliest Deadline First)와 임계 실행시간 우선 (CEF: Critical Execution time First) 스케쥴링 정책을 선택적 (alternatively)으로 적용하는 스케쥴링 기법이다 APS 알고리즘은 슬랙을 계산할 때 낙관적 (optimistic) 슬랙 계산 방식을 사용하는데, 이는 주기적 태스크의 모든 수준에서 슬랙을 계산해야만 하는 슬랙스틸링 기반 알고리즘의 최소 (minimal) 슬랙 계산 방법에 비하여 단순하다 시뮬레이션 연구결과 제안된 알고리즘은 대부분의 경우 EDF-CTI 알고리즘 및 다른 비주기적 태스크 스케쥴링보다 비슷하거나 더 빠른 반응시간을 나타내었으며 시스템 과부하 상태에서도 잘 동작하였다.

Abstract The major drawback of the slack stealing based schedulings for aperiodic requests is a high computational complexity to calculate slack which in consequence is considered impractical In order to resolve this problem, we introduce a new type of soft aperiodic task scheduling, called an Alternative Priority Scheduling (APS) Algorithm, which has a simple mechanism to calculate slack and is an optimal within a CTI (Critical Task Indicating) scheduling framework in dynamic priority systems The APS algorithm has extended the EDF-CTI (Earliest Deadline First-Critical Task Indicating) algorithm developed by the authors The algorithm references the off line built CTI table which is created by the deadlinewise preassignment policy and choices either an EDF or a CEF (Critical Execution time First) algorithm alternatively at on-line. When calculating the slacks, the proposed algorithm uses an optimistic method which is simpler than the minimal slack calculation scheme used in the slack stealing algorithms The simulation study shows that the proposed algorithm, in most cases, is slightly better than both the EDF-CTI algorithm and the other aperiodic scheduling algorithms in terms of short response time of aperiodic requests, and considerably improves the response time in transient overload.

· 본 논문은 한국과학재단 '95 목적기초 연구과제(과제번호 95-0100-07-01-3)의 지원으로 작성되었음

† 준 회 원 : 경희대학교 전자계산공학과
†† 종신회원 : 경희대학교 전자계산공학과 교수
††† 비 회 원 : 한국통신 소프트웨어연구소

논문접수 : 1996년 1월 19일
심사완료 : 1996년 6월 11일

1. 서 론

실시간 시스템 환경에서 마감시간이 있는 주기적 태스크와 마감시간이 없는 연성 비주기적 태스크가 혼합된 경우의 스케쥴링은 주기적 태스크만으로 이루어진 경우보다 일반적으로 스케쥴링 난이도가 높다. 지난 수년간 이같이 태스크가 혼합된 환경에서 연성 비주기적 요청에 대한 스케쥴링 연구는 크게 고정 우선순위 시스템[7],[8

],[9],[10],[12],[13]과 가변 우선순위 시스템[3],[14],[15]을 기반으로 하여 이루어져 왔다.

Lehoczky와 Ramos-Theul은 고정 우선순위 시스템 하에서 background, polling, bandwidth preservation 알고리즘들[12],[13]보다 비주기적 요청에 대하여 빠른 반응을 보여주는 최적 슬랙 스틸링 (slack stealing) 알고리즘[8],[9]을 제안하였다. 그러나, 슬랙 스틸링 기반 알고리즘들은 비주기적 태스크 반응 시간 향상에 많은 기여를 하였으나 슬랙 계산하는 비용이 높기 때문에 실용성이 낮다는 문제를 가지고 있다. 가변 우선순위 기반 스케쥴링으로는 최근 Tia와 Liu[15]가 최적 연성 비주기적 태스크 스케쥴링 알고리즘을 제안하였다. 이들이 제안한 알고리즘은 오프라인에서 가용한 최대의 슬랙을 구하고, 마감시간 순으로 주기적 태스크의 인스탄스를 순서화 한 다음, 각각의 마감시간 별로 파티션된 주기적 태스크 집합에 대하여 온라인에서 슬랙을 계산하는 방법이다. Tia의 알고리즘은 고정우선순위 슬랙 스틸링의 개념을 가변우선순위 시스템으로 확장하였고, 알고리즘의 최적성(지금까지 개발된 비주기적 태스크 스케쥴링 알고리즘 중에서)을 보여주었으며, 스케쥴링 구간을 파티션하는 정책을 사용함으로써 Lehoczky와 Ramos-Theul의 슬랙 스틸링 알고리즘에 비하여 슬랙 계산의 오버헤드를 줄였다. 그러나, Tia의 알고리즘은 기본적으로 슬랙스틸링 기반 알고리즘의 슬랙계산 틀을 유지하기 때문에 실용성을 가질 만큼의 슬랙계산 비용을 줄이지 못하였다 M. Spuri와 G Buttazzo[14]는 가변 우선순위 시스템 하에서의 최적에 가까운 연성 비주기적 태스크 스케쥴링 알고리즘을 제안하였으나 Tia나 Liu가 제안한 알고리즘보다 런타임 오버헤드가 높다. Homayoun과 Ramanathan [3]은 지연서버 (deferrable server) 스케쥴링 알고리즘을 EDF에서 작동하도록 확장시켰으나, 지연서버 알고리즘은 기본적으로 대역보존 스케쥴링이 가지는 한계 (비주기적 태스크의 임의의 요청 형태로 인하여 최적 대역 수준을 정할 수 없음) 때문에 프로세서의 효율을 항상 최대로 활용할 수 없다.

본 논문은 가변 우선순위 시스템 하에서 연성비주기적 태스크 스케쥴링인 APS (Alternative Priority Scheduling) 알고리즘을 제안한다. 제안한 알고리즘의 동기는 단순한 방법으로 슬랙을 계산하고, 스케쥴링 예측성 가지는 실용적인 알고리즘을 찾고자 하는 데에서 출발하였다 APS 스케쥴링 알고리즘은 비주기적인 태스크가 도착하였을 때, 오프라인에서 마감시간지향 사전할당 (deadlinewise preassignment)[4] 스케쥴링 정책으로 생성한 CTI 테이블을 참조하여, 온라인에서 주기적인 태스크들에 대하여 스케쥴링 파라메타인 $\Delta(t)$[1]에 따라 EDF와 CEF 알고리즘을 선택적으로 적용하는 스케쥴링 기법이다. 이러한 이유는 온라인 스케쥴링 시 CEF와 EDF를 선택적으로 스케쥴링 하는 것이 EDF만을 사용하는 경우보다 비주기적 태스크에 대한 반응시간을 빨리 할 수 있기 때문이다 (자세한 사항은 본문 3.1에서 기술할 예정이다). 마감시간지향 사전할당이란 주어진 주기적 태스크의 각 인스턴스들의 수행시간을 자신의 마감시간까지, 최대로 뒤로 미루어 스케쥴링 하는 정책을 말한다. CTI 테이블이란 가변 우선순위 시스템 하에서 주어진 주기적 태스크들에 대하여 마감시간지향 사전할당 정책을 사용하여 오프라인에서 생성한 테이블을 말한다. 이렇게 생성된 CTI 테이블은 임계태스크들에 대한 정보를 가지고 있는데, 임계태스크(critical task)란 주어진 주기적 태스크들 중에서 임의의 시점 t에서 할당하지 않으면 자신의 마감시간을 지킬 수 없는 주기적 태스크를 말한다 그리고, CEF 스케쥴링이란 온라인에서 임계태스크 수행시간 (Critical Task's Execution Time)이 큰 태스크에게 우선순위를 주는 정책을 말한다.

슬랙을 계산하는데 있어서 APS 알고리즘은 낙관적 (optimistic) 개념을 사용하는데 이는 주기적 태스크의 마감시간을 보장하기 위한 실행시간을 제외한 나머지 모든 시간을 슬랙이라고 낙관적으로 간주하기 때문이다. 이를 위하여 APS알고리즘은 오프라인에서 생성한 CTI 테이블의 정보를 이용하여 온라인에서 일정한 스케쥴링 구역 $[0,Z(t)]$ [2]별로 슬랙을 갱신한다. 이러한 낙관적 슬랙계산 방법은 슬랙스틸링 알고리즘에서 사용하는 최소(minimal)계산 방법에 비하여 슬랙계산이 단순하다 그 이유는 슬랙스틸링 기반 알고리즘에서는 비주기적 태스크가 도착 시 최하위 수준의 주기적 태스크 마감시간을 보장하기 위해서 모든 주기적 태스크 수준에서 구한 최대 슬랙 값 중 가장 작은 최소 값을 실제 가용한 슬랙 값으로 사용한다. 따라서, 모든 주기적 태스크 수준에서 슬랙을 찾아야하기 때문에 계산이 복잡하다.

APS 알고리즘에서 사용하는 오프라인 CTI테이블의 생성은 Chetto와 Chetto [1]가 제안한 EDL (Earliest Deadline as Late as possible) 스케쥴링 방법 중 t=0일 때 해당되는 특수한 경우 (special case)이다. EDL 스케쥴링은 비주기적 태스크가 도착한 시점에서 주기적 태스크의 마감시간을 보장하는 범위에서 주기적 태스크의 실

1) $\Delta(t)$: 주기적 태스크의 임계 실행시간의 합으로 본문 2.2장에서 정의한다.

2) $Z(t)$: 스케쥴링 시점으로 본문 2.2장에서 정의한다.

행을 가장 뒤로 미룬 스케쥴링으로 최적 비주기적 태스크 스케쥴링 알고리즘이다. 하지만, EDL 알고리즘은 시간복잡도가 높아서 실제로 이를 온라인에서 적용하는 것은 거의 불가능하다[14]. 그러나, 본 논문에서 제안한 알고리즘은 EDL과 유사한 개념을 사용하는 저자의 EDF-CTI 알고리즘[5]을 개선하여 (APS 알고리즘은 EDF-CTI 알고리즘의 단점인 단위 시간 스케쥴링 오버헤드와 임계 태스크 지시 오류로 인하여 비주기적 태스크의 반응시간을 더 단축할 수 없는 문제[6]를 해결할 수 있다) 온라인에서 슬랙의 계산을 단순화하고 알고리즘 구현 시 시간복잡도가 낮은 실용적인 비주기적 태스크 스케쥴링 기법이다.

본 논문의 구성은 다음과 같다. 2장에서는 제안한 알고리즘의 태스크 모델을 제시하고 3장에서는 APS 알고리즘의 슬랙 계산법, 알고리즘 설명, 최적성, 그리고 작동 예를 보여준다. 4장에서는 시뮬레이션 결과를 살펴보고, 5장에서 결론을 맺는다.

2. 태스크 모델

태스크는 n개의 주기적 태스크 (τ_1, τ_2, .. τ_n)로 이루어지며, 태스크 i는 실행시간 C_i, 주기 T_i, 초기 시작 시간 ϕ_i, 상대적 마감시간 D_i를 갖는다. 이때, 주기적 태스크의 집합은 $\phi_1=\phi_2=...=\phi_n=0$으로 동기화되어 있으며, $T_i=D_i$라고 가정한다. 제안한 알고리즘을 수행하는 스케쥴러의 기본 모델은 주기적인 태스크 큐와 비주기적 태스크 큐로 구성되어 있으며, [그림1]은 비주기적 태스크 스케쥴러의 기본 모델을 나내고 있다. 스케쥴러는 주기 및 비주기적 태스크 큐를 검사하여 주기적 태스크의 마감시간을 보장하는 범위에서 현재의 최대 슬랙 양만큼 비주기적 태스크가 실행될 수 있도록 스케쥴링 하게 된다.

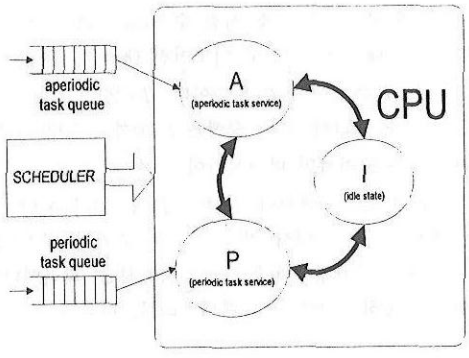


그림 1 비주기적 태스크 스케쥴러 모델

또한, 표 1은 본 논문에서 사용하는 용어 및 기호를 나타낸 것이다.

표 1 본 논문에서 사용되는 표기법 및 설명

표기	설 명
n	주기적 태스크의 수
H	하이퍼주기
N	H에 속하는 모든 주기적 태스크의 인스탄스 개수
$Z(t)$	t 시간에서 가장 가까운 주기적 태스크의 절대 마감시간
$A(t)$	t 시간까지 비주기적 태스크에 할당된 시간
$P(t)$	t 시간까지 주기적 태스크에 할당된 시간
$I(t)$	t 시간까지 유휴시간
$S(t)$	t 시간에서의 슬랙 크기
τ_i	주기적 태스크 $_i$
ϕ_i	τ_i의 초기 시작 시간
T_i	τ_i의 주기
D_i	τ_i의 상대 마감시간
C_i	τ_i의 최악 실행시간
$d_{i,j}$	τ_i가 반드시 완료되어야 할 절대 마감시간
$Cp_i(t)$	τ_i가 t시간까지 완료된 실행시간의 총합
$Cr_i(t)$	τ_i가 수행되어야할 t시간까지 수행되어야할 실행시간
$\delta_i(t)$	τ_i의 임계 실행시간: $Cr_i(t)-Cp_i(t)$
$\Delta(t)$	t시간까지 모든 주기적 태스크 임계 실행시간의 총합

한편, 본 논문에서 필요한 가정은 다음과 같다.

가정 1. 주기적 태스크의 초기 시작시간은 0이다.

가정 2. 주기적 태스크의 주기와 마감시간은 같다.

가정 3. 스케쥴링 문맥 교환 비용은 주기적 태스크의 최악 실행시간에 포함되어 있다.

3. APS 알고리즘

3.1 슬랙 계산

본 논문의 핵심은 슬랙의 계산을 단순화시키는데 있다. 다시 말하면, 현재 시간에서 어떻게 최대 슬랙을 간단하게 계산하여 요청한 비주기적 태스크에 할당할 것인가이다.

슬랙 계산 방법에는 일반적으로 정적 (static)인 방법과 동적 (dynamic)인 방법이 존재한다. 동적 계산은 온라인 상에서 슬랙 값을 계산하는 방법으로, 메모리의 양

은 적게 차지하나 런타임 시 오버헤드가 높은 단점을 가지고 있다. 정적 계산은 오프라인에서 기본적인 슬랙을 구하고 온라인에서 슬랙을 갱신하는 방법으로 변경되는 슬랙 값을 지속적 보상해주어야 하나 런타임 시 오버헤드가 상대적으로 적다는 장점을 가진다. APS 알고리즘에서 슬랙 계산은 정적 계산 방법에 해당되며 이를 위하여 제안한 알고리즘은 오프라인에서 슬랙에 대한 정보뿐만 아니라, 마감시간을 보장하는 범위에서 각 주기적 태스크의 인스탄스들을 최대로 뒤로 미룬 정보를 포함하는 CTI 테이블을 생성한다. 그런 다음 오프라인에서 생성된 CTI 테이블의 슬랙 값을 온라인에서 갱신하여 실제 비주기적 태스크를 위한 가용한 슬랙을 얻는다.

슬랙 스틸링 알고리즘 [8],[9]에서 0에서 t까지 정해진 시간동안의 스케쥴링은 주기적 태스크 수행시간 $P_{ij}(t)$, 비주기적 태스크 수행구간 $A(t)$, 태스크가 수행되지 않은 운휴구간$I(t)$ 으로 나눌 수 있으며 다음과 같이 표현된다.

$$t = \sum_{i,j} P_{ij}(t) + A(t) + I(t) \quad \cdots \quad (1)$$

이때, 주기적 태스크의 마감시간을 보장하는 범위에서 비주기적 태스크에게 할당할 수 있는 시간의 양을 슬랙이라고 한다. 따라서, 0에서 t까지의 슬랙 $S_0(t)$는 전체 스케쥴링 시간 중에서 주기적 태스크가 할당된 시간을 제외한 시간이나 실제 비주기적 태스크에 할당된 시간은 전체 슬랙과 같거나 그보다 작은 시간이 된다. 비주기적 태스크에 할당되지 않은 시간은 운휴시간이며 (2)의 식으로 나타낼 수 있다. 여기서, 아래첨자 0은 시작 시점에서 t까지 스케쥴링된 결과를 의미한다.

$$S_0(t) = A_0(t) + I_0(t) \quad \cdots \quad (2)$$

그러므로, 슬랙 스틸링 기반 스케쥴러는 현재 주기적인 태스크의 마감시간을 보장하기 위해서 주기적인 태스크를 수행하든지 아니면, 비주기적 태스크 할당을 위해서 슬랙을 계산하는 역할을 한다. 지금까지 제안된 슬랙 스틸링 기반 비주기적 태스크 스케쥴링 알고리즘들의 슬랙 계산은 각 주기적 태스크 수준에서 산출된 최대슬랙 중에서 가장 작은 값을 실제 사용 가능한 슬랙으로 간주하는 최소 슬랙 계산 (minimal slack calculation) 방법이며 전형적으로 (3)식과 같은 형태를 갖는다. 그러나, 이 경우 비주기적 태스크가 도착 시 최하위 주기적 태스크의 마감시간을 보장하기 위해서 모든 주기적 태스크 수준에서 슬랙을 찾아야하기 때문에 계산이 복잡하다.

$$S^*(t) = \min_{\{1 \le k \le n\}} S_k(t) \quad \cdots \quad (3)$$

이와는 달리 APS 알고리즘은 주기적 태스크의 마감시간을 보장하기 위한 최소한의 주기적 태스크 할당 시간을 제외한 나머지 시간을 슬랙으로 간주하는 낙관적 슬랙 계산 (optimistic slack calculation) 방법을 사용하며 (4)식과 같이 나타낼 수 있다. 낙관적 슬랙 계산 방법은 최소 슬랙 계산 방법과는 달리 모든 주기적 태스크의 우선순위 수준에 따른 슬랙의 최소 값을 구할 필요가 없으며, (4) 식과 같이 사용되지 않은 주기적 태스크의 실행시간을 모두 슬랙으로 포함한다는 점이다

$$S(t_1, t_2) = t_2 - t_1 - \sum_{i=1}^{n} \delta_i \quad (4)$$

(4)식은 t_1과 t_2사이의 실제 가용한 슬랙 값을 각 태스크의 마감시간을 보장하기 위한 최소한의 주기적 태스크 할당시간 (δ)을 제외함으로써 얻을 수 있음을 보여준다. 여기서, t_2는 슬랙 계산의 범위를 제한하는 값이며 t_2가 생략된 경우에는 그것은 현재 시간으로부터 주기적 태스크 인스탄스의 가장 가까운 마감시간을 의미한다.

한편, 슬랙을 효과적으로 계산하기 위하여 APS 알고리즘에서는 비주기적 태스크가 시점 t에 도착하였을 때 어느 시점까지 슬랙을 계산해야 할지를 고려한다. 이러한 시점을 t에서 가장 가까운 주기적 태스크 인스탄스의 마감시간인 $Z(t)$로 정의하며 다음과 같은 식으로 나타낼 수 있다.

$$Z(t) = \min\left\{ d_{i,j} \mid 1 \le i \le n, j = \left\lceil \frac{t}{T_i} \right\rceil \right\} \quad \cdots \quad (5)$$

임의의 시점 t에서 스케쥴러의 가장 큰 관심은 $Z(t)$에 해당하는 주기적 태스크의 마감시간을 보장하면서 슬랙을 구하는데 있다. 이는 하이퍼주기 전체를 고려하지 않고도 CTI 테이블의 임계태스크 정보를 참조하여 [0, $Z(t)$]에서 비주기적 태스크를 위한 슬랙을 계산함으로써 슬랙 계산의 복잡성을 감소 시킬 수 있다. 예를 들어, 두 개의 주기적 태스크 τ_1, τ_2로 이루어진 태스크 집합이 있다고 하자. T_1은 5이고 C_1은 2이다. T_2=20이고 C_2는 1이다. τ_2만을 고려한 경우 슬랙은 t=20까지 19가 된다. 하지만, τ_1을 고려하면 마감시간이 가장 빠른 τ_1의 마감시간인 5이므로 t=0에서의 슬랙은 T_1-C_1=3이 된다. 따라서, $Z(t)$는 t 시간에서 가장 가까운 주기적 태스크의 마감시간까지 슬랙을 계산하는데 사용하는 파라메타이다. 이를 이용하여 계산된 슬랙의 크기는 다음과 같다.

$$S(t) = S(t, Z(t)) = Z(t) - t - \sum_{i=1}^{n} \delta_i(t) \quad \cdots \quad (6)$$

(6)식의 $\delta_i(t)$는 본 알고리즘 핵심적인 개념으로 이를 τ_i의 임계 태스크 실행시간이라 한다. 임계 태스크 실행시간이란 슬랙을 계산할 때 마감시간을 보장하기 위하여 사전 할당된 주기적 태스크가 반드시 수행되어야 할 것인가, 아닌가를 판단하는 값으로 CTI 테이블로 부터 구할 수 있다. t=0에서 t까지 반드시 실행 완료되어야 할 τ_i 부분을(cumulating all the computation requirement: $Cr_i(t)$)라하고 t까지 실제 실행된 τ_i 부분을 (cumulating all the computation requirement: $Cp_i(t)$)라 한다면[6], $Cp_i(t)$와 $Cr_i(Z(t))$의 차가 임계 실행시간 $\delta_i(t)$이며 다음과 같은 식으로 나타낼 수 있다.

$$\delta_i(t) = \begin{cases} 0 & \text{if } (Cr_i(Z(t)) - Cp_i(t)) < 0 \\ Cr_i(Z(t)) - Cp_i(t) & \text{otherwise} \end{cases} \quad \cdots \quad (7)$$

한편, $\delta_i(t)$의 값이 큰 태스크에게 우선 순위를 높게 할당하는 정책을 임계 실행시간 우선 (CEF: Critical Execution time First) 스케쥴링이라고 정의한다. 마감시간지향 사전할당에 의한 CTI 테이블의 적합성(feasibility)은 이미 [1],[5]에서 증명되었다. 그러므로, CEF 스케쥴링은 CTI 테이블에 사전 할당된 태스크들의 실행시간보다 늦지 않게 태스크들을 스케쥴링 하는 것이므로 적합성을 가지고 있다.

온라인 스케쥴링을 EDF만 사용 하지않고 CEF와 선택적으로 수행하는 이유는 비주기적 태스크의 반응 시간을 빠르게 하기 위함이다. 예를 들어 스케쥴러가 온라인 스케쥴링을 EDF 정책만을 가가고 수행할 때 임계 태스크보다 마감시간이 빠른 경우의 주기적인 태스크가 있다면 그것을 먼저 할당할 것이다. 이때, 식 (6)은 음수가 되며 스케쥴러는 더 이상의 가용슬랙을 찾을 수 없다 (참고로, 임계 태스크라고 해서 주기적 태스크보다 항상 마감시간이 더 빠른 것을 의미하지는 않는다). 따라서, 이런 경우 가용 슬랙이 있는데도 비주기적 태스크를 스케쥴할 수 없는 경우가 발생할 수 있다. 그러나 CEF를 사용하는 경우 스케쥴러는 CTI 테이블의 정보에 의해서 $Z(t)$까지는 최소한의 주기적 태스크 수행시간만을 할당하므로 (6)식은 음수가 되지 않는다. 그러므로 CEF를 사용하는 경우 더 많은 슬랙을 찾을 수 있는 가능성을 가지고있다. 한편, 수식을 간단히 하기 위하여 모든 주기적 태스크의 임계 실행시간의 합을 $\Delta(t)$라고 정의하며 다음과 같은 식으로 나타낸다.

$$\Delta(t) = \sum_{i=1}^{n} \delta_i(t) \quad \cdots \quad (8)$$

3.2 알고리즘 설명

이 절에서는 위에서 정의한 식들을 사용하여 APS 알고리즘을 설명하는데 그것의 의사코드는 그림 2와 같다.

```
1  Build off-line CTI table
2  Initialize counters
3  while TRUE do
4    Calculate S(t) and Cr_i(Z(t));
5    while (t < Z(t)) do
6    if (there is slack and there is an aperiodic task in the queue) then
7      while (S(t) > 0 and there is an aperiodic task in the queue) do
8        Service the aperiodic task;
9        Update S(t) /* reduce the amount of service time from S(t) */
10   else if (there is periodic task in the queue) then
11     if (Δ(t)>0) then
12               Service the periodic task with the CEF;
13       Update Cp_i(t) /* add the amount of processing time */
14     else
15               Service the periodic task with the EDF;
16       Update Cp_i(t)  /* add the amount of processing time */
17       Update S(t)  /* reduce the amount of processing time */
18   else
19     Process idle state;
20     Update S(t) /* reduce the amount of idle time */
21   endwhile
22 endwhile
```

그림 2 APS 알고리즘 의사코드

알고리즘 의사코드를 살펴보면 다음과 같다. 1~2라인은 오프라인에서 CTI 테이블을 생성하고 스케쥴링 파라메타를 초기화한다. 4라인에서 스케쥴링 구간의 시작에서 슬랙 값을 계산한다. 6~9라인은 슬랙 값이 있을 경우 주기적 태스크보다 비주기적 태스크를 우선해서 서비스하며 10~17라인에서 실행할 주기적 태스크가 있는 경우 Δ(t)에 따라 CEF 스케쥴링 또는 EDF 스케쥴링을

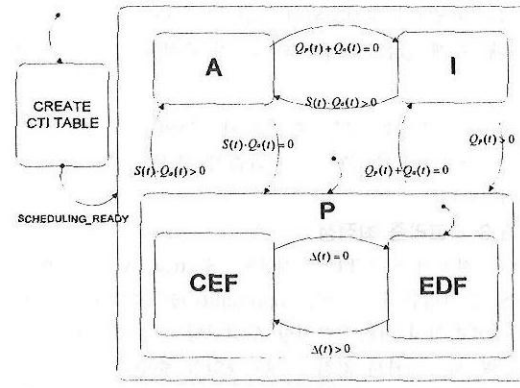


그림 3 Statechart로 표시된 APS 스케쥴링의 상태전이도

선택적으로 적용한다. 19라인은 서비스할 태스크가 존재하지 않을 경우 운휴시간으로 처리한다.

한편, APS 알고리즘의 상태전이를 Statechart[2]로 나타내면 그림 3과 같다.

그림 3에서 스케줄링 상태는 크게 A, I, P 상태로 구성되어 있으며 P 상태는 다시 CEF와 EDF 상태로 나누어진다. $Q_p(t)$, $Q_a(t)$, $S(t)$는 각각 t시간에 도착되어 있는 주기적 태스크의 실행시간 합, 도착한 비주기적 태스크의 실행시간 합, 사용 가능한 슬랙의 크기를 나타낸다. 각 스케줄링 상태 및 전이에 대한 설명은 다음과 같다.

- A 상태: 비주기적 태스크 서비스 상태이며, 슬랙이 0보다 크고, 비주기적 태스크가 있는 경우 ($S(t) \cdot Q_a(t) > 0$)에 P 상태 또는 I 상태로부터 전이된다. 이때, 슬랙 값은 비주기적 태스크를 서비스한 만큼 감소한다.
- I 상태: 프로세서 운휴상태로 주기적 태스크와 비주기적 태스크 모두 존재하지 않는 경우 ($Q_p(t)+Q_a(t)=0$)에 A 상태 또는 P 상태로부터 전이된다. 이때, 운휴시간만큼 슬랙 값은 감소한다.
- P 상태: 주기적 태스크 서비스 상태이며, 비주기적 태스크가 없거나 슬랙이 0일 경우 ($S(t) \cdot Q_a(t)=0$)에 A 상태로부터 전이되고, 주기적 태스크가 도착하였을 경우($Q_p(t)>0$)에 I 상태로부터 전이된다.
- CEF 상태: $\Delta(t)>0$을 만족할 경우 EDF 상태로부터 전이된다. 이때, 슬랙 값에는 변화가 없다.
- EDF 상태: 스케줄링이 처음 시작할 때 기본적으로 EDF 상태가 되며, $\Delta(t)=0$을 만족할 경우 CEF 상태로부터 전이된다. EDF 스케줄링은 마감시간이 짧은 태스크의 우선순위가 높은 스케줄링 정책이다. 이때, 슬랙 값은 주기적 태스크가 수행된 만큼 감소한다.

여기서, $\Delta(t)$는 모든 주기적 태스크의 임계 실행시간의 합으로 CEF와 EDF 알고리즘의 선택을 결정하는 파라메타이다.

3.3 알고리즘 최적성

이 절에서는 CTI 스케줄링 틀(framework) 에서 APS 알고리즘의 최적성 (optimality)을 증명하기 위해서 오프라인의 마감시간지향 사전할당 기법이 온라인 스케줄링 시 $t=0$에서 최적 스케줄링임을 증명하고 이를 이용한 낙관적 슬랙 계산 방법이 온라인 스케줄링 시 최적임을 증명한다. 이를 위하여 본 알고리즘은 greedy한 방법을 사용하여 슬랙을 계산하며 다음의 정리를 도입한다.

정리1) 가변우선순위 시스템에서 오프라인 스케줄링인 EDF 마감시간지향 사전할당 기법은 하이퍼주기의 시작점에서 비주기적 태스크에 대한 반응시간이 최적인 스케줄링이다.

(증명) 마감시간 지향 사전할당은 결과적으로 EDF의 최적 비주기적 태스크 스케줄링 알고리즘인 EDL 알고리즘 수행 시 하이퍼주기의 시작점에서의 특별한 한 경우에 해당된다. EDL은 주기적 태스크 인스탄스의 실행시간을 가장 마감시간에 근접하게 할당한 것으로 Chetto 와 Chetto[1]에 의하여 알고리즘의 적합성과 최적의 슬랙을 찾는다는 것이 증명되었다. 따라서, 마감시간 지향 사전할당은 하이퍼주기의 시작점에 도착한 비주기적인 태스크에 대한 반응시간이 최적인 오프라인 스케줄링이다. ■

EDL과 마감시간 지향 사전할당의 차이점은 EDL은 온라인 스케줄링 알고리즘이며, Chetto와 Chetto는 이를 경성 비주기적 태스크 스케줄링에 적용하였으나, 마감시간 지향 사전할당은 오프라인 스케줄링이며 연성 비주기적 태스크 스케줄링에 적용하였다는 점이다. 오프라인 스케줄링의 결과만으로는 온라인에서도 최적의 스케줄링이라고 할 수 없다. 왜냐하면, 어떤 시점에서 가능한 슬랙이 모두 사용되는 것이 아니기 때문이다.

정리2) CTI 스케줄링 틀(framework)에서 APS 알고리즘은 온라인 스케줄링 시 임의의 시점 t에서 최대 슬랙을 찾을 수 있다.

(증명) 스케줄링 구간 $[0, Z(t)]$에서 APS 알고리즘보다 더 많은 슬랙을 찾을 수 있는 알고리즘을 X 라고 가정하자. 그리고 알고리즘 X에 의해서 구한 모든 슬랙을 비주기적 태스크를 위해 사용하였다고 가정하자. 그러면 $Z(t)$ 이후의 어떤 시간 t'에서 최악의 경우 알고리즘 X는 주기적 태스크의 마감시간을 보장할 수 없을 지도 모른다. 왜냐하면 알고리즘 X는 $Z(t)$이후에 스케줄되어야 할 주기적 태스크의 실행시간과 마감시간을 고려할 수 없기 때문이다. 다시 말하면, APS 알고리즘에서 임계태스크 수행시간은 $Z(t)$까지 주기적 태스크의 마감시간을 보장할 수 있는 최소값만을 유지하므로, 만일 APS 보다 더 많은 슬랙을 찾은 알고리즘 X가 있다면 그것은 $Z(t)$ 이후의 주기적 태스크의 마감시간은 보장할 수 없게 된다. ■

참고로, 가변우선순위 시스템에서 어떤 greedy 스케

쥴링 알고리즘이 최대의 슬랙을 구한다면 그것은 비주기적 태스크의 반응시간을 빠르게 한다는 점에서 최적이다 [19]. 따라서, APS 스케쥴링 알고리즘은 CTI 스케쥴링 틀에서 비주기적 태스크에 대한 최적 온라인 스케쥴링이 된다. 한편, APS 알고리즘은 오프라인에서 슬랙을 계산하는 공간복잡도가 O(N)으로 주로 CTI 테이블을 생성하는데 의존하며 Tia의 $O(N^2)$보다 간단하다. 그리고, 온라인 상에서의 슬랙 계산시 APS 알고리즘은 Tia의 O(n)과 같지만 실제 알고리즘 구현 시 Tia의 경우는 주어진 모든 주기적 태스크의 실행시간을 고려해야 되지만 APS의 경우는 스케쥴링 구간 내에 CTI 테이블에 할당되어 있는 주기적 태스크 만을 고려하기 때문에 Tia의 경우보다 단순하다. 여기서 N은 하이퍼주기 내의 주기적 태스크 집합의 인스탄스의 개수이며, n은 주기적 태스크의 개수이다.

3.4 알고리즘 예제

이 절에서는 제안한 알고리즘의 작동을 예를 통하여 설명하기로 한다.

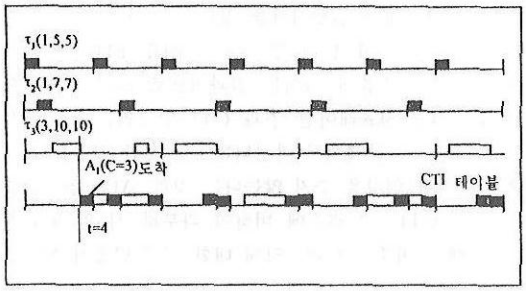


그림 4 APS 알고리즘의 예

(예제1) 그림 4는 t=4에 첫 번째 비주기적 태스크 A_1(C=3)이 도착하였을 때 스케쥴링 과정과 A_1의 반응시간 R이 얼마인가를 보여준다.

(a) t=4까지 스케쥴링 과정은 다음과 같다.

스케쥴링 시작 (t=0) 시점에서 $Z(t)$=5라는 것을 알 수 있다. 따라서, 슬랙 값 $S(0)$는 다음과 같이 구할 수 있으며 t=5에서 다시 슬랙 값은 갱신됨을 알 수 있다.

$$S(0) = S(0, Z(0)) = S(0,5) = 5 - 0 - \sum_{i=1}^{3} \delta_i(t)$$
$$= 5 - (1+0+0)$$
$$= 4$$

$S(0)$=4이므로 0과 5사이 구간에서는 1만큼의 임계 실행시간을 주기적 태스크에 할당한 나머지 4는 슬랙 값이라고 할 수 있다. t=0에서는 비주기적 태스크가 존재하지 않으므로 임계 태스크 $\tau_{1,1}$을 CEF 알고리즘에 의해서 수행 완료하였다. t=1에서 t=4까지는 EDF 알고리즘에 의해서 $\tau_{2,1}$, $\tau_{3,1}$이 차례대로 수행되었으며 슬랙 값은 3만큼 감소하여 $S(4)$=1이 되었다.

(b) t=4에서 비주기적 태스크 A_1이 도착하였다. t=5까지 1만큼의 슬랙이 남았으므로 1만큼 비주기적 태스크를 서비스하였다. t=5에서 $S(5)$를 구하면 다음과 같다.

$$S(5) = S(5, Z(5)) = S(5,10) = 10 - 5 - \sum_{i=1}^{3} \delta_i(t)$$
$$= 5 - (1+0+1)$$
$$= 3$$

이때, $Z(5)$=7이 아니라 $Z(5)$=10인 이유는 $\tau_{2,1}$은 이미 수행 완료되었으므로 $\tau_{2,2}$의 마감시간 14가 τ_2의 가장 가까운 절대 마감시간이기 때문이다. $S(5)$=3 임으로 A의 남은 실행시간 2를 바로 서비스할 수 있다. 따라서, 이 경우의 반응시간은 4가 된다.

4. 시뮬레이션

이 장에서는 EDF-CTI 알고리즘과 APS 알고리즘의 시뮬레이션 결과를 비교하였다. 주기적 태스크의 부하가 낮은 경우에는 두 알고리즘 모두 거의 같은 반응시간을 나타내므로 주기적 태스크의 부하가 90%를 비교하여 차이를 얻을 수 있게 하였다. 하지만 부하가 높은 경우에도 주기 및 비주기적 태스크 집합의 총 부하가 100%에 근접할 때 비로소 평균 반응시간의 차이를 확인할 수 있었다. 시뮬레이션에서 비주기적 태스크의 도착시간과 실행시간은 지수 분포를 따르고, 사전에 주어진 평균 실행시간과 비주기적 태스크의 부하에 따라 발생하는 비주기적 태스크의 실행시간을 결정하였다. 시뮬레이션에 사용된 태스크 집합의 구성은 [표2]와 같다. 이는 고정 우선순위 알고리즘인 비율 단조 알고리즘 (RMA: Rate-Monotonic Algorithm)[11]에서도 스케쥴링 가능한 10개의 태스크 들이다. 이때, 주기와 실행시간은 단위 시간으로 한다.

평균 실행시간에 대한 비주기적 태스크의 평균 반응시간의 차이를 알아보기 위하여 짧은 평균 실행시간은 3.5를 긴 평균 실행시간은 21.0을 기준으로 하여 각 알고리즘에 대한 평균 반응시간을 살펴보았다. 이 때, CTI 알고리즘은 오프라인 알고리즘과 온라인 알고리즘에 따

라 RMA-RMA, RMA-EDF, EDF-RMA, EDF-EDF 등 4가지 경우를 가질 수 있는데, EDF-CTI 알고리즘은 EDF-EDF 알고리즘을 의미한다. 따라서, 시뮬레이션 결과는 APS 알고리즘을 포함하여 모두 5개의 평균 반응시간을 나타낸다.

표 2 시뮬레이션에 사용된 태스크 집합의 주기와 실행시간

태스크 번호	주기	실행시간
1	100	2
2	280	14
3	2100	108
4	440	29
5	350	14
6	210	30
7	35	8
8	70	11
9	2200	231
10	300	12

주기적 태스크의 부하가 90%이고, 비주기적 태스크의 평균 실행시간이 3.5 일 때, 비주기적 태스크에 대한 평균 반응시간에 대한 결과는 그림 5와 같다. 그림 5에서 높은 부하가 될 경우에 고정 우선순위를 기반으로 한 알고리즘보다 가변 우선순위를 기반으로 한 알고리즘이 더 빠른 평균 반응시간을 얻을 수 있음을 알 수 있으며, EDF-EDF로 표기된 EDF-CTI와 APS 알고리즘은 매우 근소한 평균 반응시간의 차이를 나타내었다.

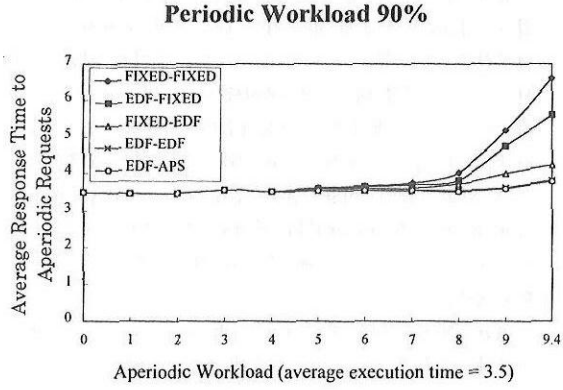


그림 5 비주기적 태스크의 평균 실행시간이 3.5일 경우

주기적 태스크의 부하가 90%로 동일하나 비주기적 태스크의 평균 실행시간이 21일 때, 평균 반응시간에 대한 결과는 그림 6과 같다.

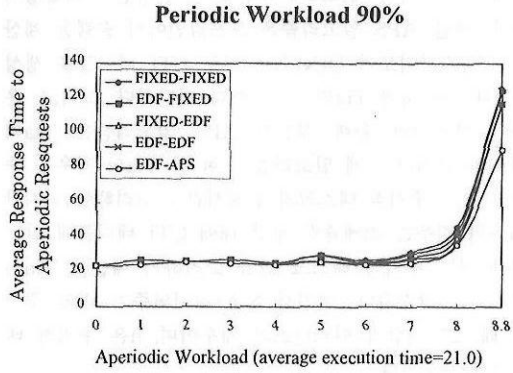


그림 6 비주기적 태스크의 평균 실행시간이 21일 경우

그림 6과 같이 긴 실행시간을 갖는 비주기적 태스크의 경우에도 짧은 실행시간을 갖는 경우와 마찬가지 결과를 가진다. EDF-CTI 알고리즘 (EDF-EDF로 표시)과 APS 알고리즘의 차이는 결과적으로 크지 않았음을 알 수 있다. 시뮬레이션 결과 CTI 알고리즘이 슬랙을 잘못 지정하는 문제에서 파생되는 반응시간 지연은 시스템에 심각한 영향을 주지 않는다는 것과, APS 알고리즘은 EDF-CTI 알고리즘에 비하여 과부하 시 약 5% 정도 더 빠른 비주기적 태스크에 대한 평균 반응시간을 나타낸다는 것을 알 수 있다.

5. 결 론

슬랙스틸링 기반 비주기적 태스크 스케쥴링은 슬랙 계산의 복잡도로 인하여 실용적이지 못하다는 문제점을 가지고있다. 이를 개선하기 위하여 본 논문에서는 가변 우선순위 시스템 하에서 단순하게 슬랙을 계산할 수 있는 선택적 우선순위 연성 비주기적 태스크 스케쥴링 (APS) 알고리즘을 제안하였다. 본 논문에서 제안한 APS 알고리즘은 스케쥴링 시 오프라인에서 예측성을 가지는 CTI 테이블을 참조하여 온라인에서 EDF 정책과 CEF 정책을 선택적으로 사용하는 스케쥴링 기법이다.

APS 알고리즘에서 선택적 스케쥴링을 하는 이유는 온라인 스케쥴링 시 CEF와 EDF를 선택적으로 적용 하는 것이 EDF만을 사용하는 경우보다 비주기적 태스크에 대한 반응시간을 빨리할 수 있기 때문이었다. 그리고,

제안한 알고리즘은 스케줄링 시 오프라인에서 생성한 CTI 테이블을 사용하여 스케줄링 구간의 주기적 태스크의 마감시간을 보장하기 위한 최소한의 실행시간을 제외한 나머지 시간을 슬랙으로 할당하는 낙관적인 슬랙 계산법을 적용하였다. 이러한 개념은 주기적 태스크의 모든 수준에서 슬랙을 계산 해야하는 슬랙 스틸링 기반 알고리즘의 최소 슬랙 계산 방법에 비하여 훨씬 단순하다. 이를 효과적으로 구현하기 위하여 APS 알고리즘에서는 온라인 상에서 비주기적 태스크가 도착한 시점 t부터 가장 가까운 주기적 태스크 인스탄스의 마감시간 $Z(t)$까지 모든 주기적 태스크의 마감시간을 보장하면서 슬랙을 구하는 것이다. 이 경우, 스케줄러는 하이퍼주기 전체를 고려하지 않고도 CTI 테이블의 임계태스크 정보를 참조하여 스케줄링 구간 $[0, Z(t)]$에서 비주기적 태스크를 위한 슬랙을 계산함으로써 슬랙계산의 복잡성을 감소시킬 수 있다. 결국, APS 알고리즘은 하이퍼 주기를 파티션한 최초의 스케줄링 구간에서 최대 슬랙을 찾고, $t = 0$ 시점으로 부터 임의의 $Z(t)$ 까지 항상 최대의 슬랙을 찾으므로써 결과적으로 하이퍼주기 전체 스케줄링 구간이 최적의 스케줄링이 되도록 하는 접근 방법이다.

또한, APS 알고리즘은 EDF-CTI 알고리즘에 비하여 단위시간마다 슬랙을 판별해야 하는 오버헤드를 줄임으로써 더 빠른 비주기적 태스크의 반응시간을 얻을 수 있었고, 임계 태스크를 잘못 지시하는 문제도 해결하였다. 시뮬레이션 연구결과 제안된 알고리즘은 대부분의 경우 EDF-CTI 알고리즘 및 다른 비주기적 태스크 스케줄링 보다 비슷하거나 더 빠른 반응시간을 나타내었으며 시스템 과부하 상태에서도 잘 동작하였다. 앞으로 연구과제는 APS 알고리즘과 다른 최적 비주기적 태스크 스케줄링 알고리즘과 성능 비교를 하는 것이며, 보다 견고한(robust)한 온라인 CEF 스케줄링 기법을 개발할 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, No. 10, pp. 466-473, 1989

[2] D Harel, "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, Vol.8, pp. 231 274, 1987

[3] N. Homayoun and P. Ramanathan, "Dynamic Priority Scheduling of Periodic and Aperiodic Tasks in Hard Real-Time Systems", Real-Time Systems: The International Journal of Time Critical Computing Systems, Vol. 6, No. 2, pp. 207-232, 1994

[4] H.I. Kim, S Y Lee, J.W Lee, and J.S. Kim, "An Aperiodic Task Scheduling Algorithm by Hybrid Priority Assignment in Real-Time Environments", Journal of the Korea Information Science Society, Vol. 22, No. 5, pp. 748-758, May, 1995

[5] S.Y. Lee, H I Kim and J.W. Lee, "A Soft Aperiodic Task Scheduling Algorithm in Dynamic-Priority Systems", Proceedings 2nd International Workshop on Real-Time Computing Systems and Applications, pp. 68-72, Tokyo, October, 1995.

[6] 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, pp 7-19, La Jolla, CA, June, 1995.

[7] 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.

[8] 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

[9] J P Lehoczky and S. R. Thuel, Scheduling Periodic and Aperiodic Tasks using the Slack Stealing Algorithm (Chapter 8), Advances in Real-Time Systems, (ed. S. Son) Prentice Hall, Englewood Cliffs, NJ, 1995.

[10] R. Davis and A. Wellings, "Dual Priority Scheduling", Proceedings of the IEEE Real-Time Systems Symposium, pp. 100-109, Pisa, Italy, December 1995.

[11] 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, Vol. 20, No.1, pp. 46-61, January 1973.

[12] B Sprunt, J.P. Lehoczky, and L. Sha, "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System", Technical Report CMU/SEI-89DTR-11, April 1989.

[13] B. Sprunt, J.P. Lehoczky, and L. Sha, "Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm", Proceedings of the IEEE Real-Time System Symposium, pp 251-258, December 1988.

[14] M Spuri and G.C Buttazzo, "Efficient Aperiodic Service under Earliest Deadline Scheduling",

Proceedings of the IEEE Real-Time System Symposium, pp. 2-11, 1994.

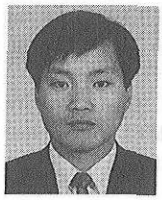
[15] T.S. Tia, Utilizing Slack Time for Aperiodic and Sporadic Requests Scheduling in Real-Time Systems, Technical Report No. UIUCDCS-R-95-1906, University of Illinois, April, 1995.

김 형 일

1994년 경희대학교 물리학과 졸업. 1996년 경희대학교 전자계산공학과 대학원 석사. 1996년~현재 경희대학교 전자계산공학과 대학원 박사과정. 관심분야는 실시간 운영체제, 멀티미디어 시스템, fault-tolerant 시스템임.

이 승 룡

1978년 고려대학교 공과대학 재료공학과 졸업. 1987년 미국 Illinois Institute of Technology 전산학과 석사. 1991년 미국 Illinois Institute of Technology 전산학과 박사. 1990년~1992년 미국 Governors State University 전임강사. 1992년~1993년 미국 Governors State University 조교수. 1993년~현재 경희대학교 전자계산공학과 조교수. 관심분야는 운영체제, fault-tolerant 시스템, 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. 관심분야는 실시간 시스템, 운영체제, 데이타베이스임.