복합 태스크 모델에 대한 효율적인 실시간 스케쥴링

김 인 국†

요 약

이제까지의 실시간 스케쥴링은 대상이 되는 태스크들이 모두 선점가능하거나 모두 선점불가능함을 가정하였다. 본 논문에서는 단일 프로세서 환경에서 선점가능한 서브태스크와 선점불가능한 서브태스크를 모두 포함하는, 보다 일반화된 태스크 모델에 대한 고정 우선순위 전략을 기반으로 하는 실시간 스케쥴링 방법을 제시하였다. 기존에 발표된 바 있는 Harbour등의 방법에 의해 스케쥴링이 가능하다고 판정된 태스크 집합은 본 논문의 방법에 의해서도 스케쥴링이 가능하다고 판정되며 두 방법을 모의실험을 통하여 비교해 본 결과 최대 45% 이상의 효율의 차이가 남이 보여지고 있다.

Effective Real-Time Scheduling in Composite Task Model

In-Guk Kim †

ABSTRACT

Most of the real-time scheduling algorithms assume that all tasks are either preemptive or nonpreemptive. In this paper, we present a real-time scheduling algorithm for the more generalized task model in which each task contains both preemptive and nonpreemptive subtasks in a single processor environment. If the task set is found to be schedulable by the method of Harbour et al., it is also found to be schedulable by the proposed method. A simulation is used to compare two methods and the result shows the maximum of 45% difference between them in their effectiveness.

1. 서 론

실시간 시스템이란 시스템 내에서 처리되어 지는 각 작업에 대해 처리시한의 제한(이하 종료시한 : deadline)이 있고 그것을 지키는 것이 엄격히 요구되어지는 경우의 시스템을 말한다. 실시간 스케쥴링이란 실시간 시스템에서 실행되는 작업들이 종료시한 내에 종료될 수 있도록 스케쥴을 구성하는 것을 말하며 이는 실시간 시스템의 안정성을 위하여 대단히 중요한 연구분야의 하나이다.

실시간 스케쥴링은 주로 Rate Monotonic Scheduling(이하 RMS) 알고리즘과 Earliest Deadline Scheduling(이하 EDS) 알고리즘을 근간으로 하여 발전하여 왔다[8]. 기본적인 RMS 알고리즘은 선점가능한(preemptive) 스케쥴링 알고리즘이고 각 태스크들은 독립적이라 가정되며 각 태스크에게 정적(static)으로 우선순위를 부여하고 이때 주기가 짧은 태스크 일 수록 높은 우선순위가 주어진다. 기본적인 EDS 알고리즘은 선점가능한 알고리즘이고 각 태스크들은 독립적이라 가정되며 각 태스크에게 동적(dynamic)으로 우선순위가 부여되는데 매 스케쥴링 시점에서 볼 때 종

† 정 회 원 : 단국대학교 전자계산학과
논문접수 : 1996년 8월 9일, 심사완료 : 1996년 9월 17일

료시한까지의 거리가 가장 짧은 태스크에게 우선순위가 주어진다. 따라서 각 태스크의 우선순위는 가변적이며 이를 결정하는데 실행 오버헤드가 상당하다.

Jeffay등은 EDS 알고리즘을 변형시켜서 선점불가능한 태스크들에 대해 적용한 바 있다[5]. 태스크들은 주기적이라 가정되었으며, 일단 지명(dispatch)된 태스크는 도중에 선점됨이 없이 실행을 마친다. 스케쥴러는 프로세서가 유휴(idle) 상태에 놓일 때만 작동되며, 이때 종료시한까지의 거리가 가장 짧은 태스크가 지명된다.

주어진 태스크 집합에 대한 스케쥴 S는 집합내의 모든 태스크들이 그들의 종료시한이내에 모두 종료되는 경우에 유효(valid)하다고 하며 유효한 스케쥴이 존재하는 태스크 집합은 스케쥴링 가능하다고 불리운다. 최적(optimal) 스케쥴링 알고리즘은 스케쥴링이 가능한 모든 태스크 집합에 대해 유효한 스케쥴을 만들어 낼 수 있는 알고리즘을 말한다. RMS 알고리즘은 우선순위가 고정적으로 주어진 태스크 집합들에 대한 스케쥴링 알고리즘들중 최적 알고리즘이고 EDS 알고리즘은 단일 프로세서 환경에서 최적 알고리즘이며 이는 Liu와 Layland에 의해 증명되었다[8]. 그러나 기본적인 RMS 알고리즘과 EDS 알고리즘은 태스크들이 모두 선점가능한 경우에만 적용될 수 있다.

선점불가능한 태스크들로 구성된 태스크 집합의 스케쥴링에 대하여는 발표된 연구 결과가 그다지 많지 않다. 또한 발표된 대부분의 연구 결과들은 태스크들의 실행 요구시간이 같은 경우의 태스크 모델을 가정하고 있다[1,2,3,9,10]. 그러나 선점불가능한 태스크들과 그들에 대한 스케쥴링 방법들은 나름대로의 중요한 의미를 가지고 있으며 이는 Jeffay등에 의해 적절히 지적된 바 있다[5].

일반적으로 태스크의 실행은 사용자 모드인 경우에는 선점가능한 상태에서 처리되다가 시스템 서비스 요청(system call)에 대한 처리를 하게 될 때는 선점(preemption)을 보류(disable)시키는 것이 보통이다. 이 경우에 시스템 서비스 요청의 처리가 시작되어서 끝날 때까지의 구간은 선점불가능한 서브태스크로 모델링할 수 있다. 따라서 하나의 태스크는 선점가능한 서브태스크와 선점불가능한 서브태스크를 모두 포함하고 있다고 생각하는 것이 보다 현실적이다. 이때 하나의 태스크를 구성하는 서브태스크들은 그들이 구성된 순서에 의한 우선 실행관계(precedence relation)를 갖는다[4].

본 논문에서는 단일 프로세서 환경에서 각 태스크가 다수개의 선점가능한 서브태스크와 선점불가능한 서브태스크들로 구성된 경우의 태스크 집합에 대한 스케쥴링 가능성을 검토하고자 한다. 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구가 소개되고 3장에서는 시스템의 모델과 사용될 기호 및 용어들에 대하여 기술하며 4장에서는 본 논문에서 제안된 스케쥴링 방법 및 스케쥴링 가능성 검사 기준이 기술된다. 5장에서는 타 모델로의 적용 가능성을 검토하였고 6장에서는 모의 실험을 통하여 제안된 방법의 효율성을 여러 가지 종류의 태스크 집합을 이용하여 분석하였으며 마지막으로 결론을 기술하였다.

2. 관련 연구

이제까지의 스케쥴링 방법들은 대상이 되는 태스크들이 모두 선점가능한(preemptive) 태스크들이거나 모두 선점불가능한(nonpreemptive) 태스크들인 것으로 가정을 하고 문제를 해결하여 왔다. 태스크들이 모두 선점가능한 경우의 태스크 모델에 대한 스케쥴링 방법들은 서론에서 언급된 바 있다. 태스크들이 모두 선점불가능한 경우의 태스크 모델에 대한 이제까지의 연구들은 태스크들이 주로 비주기적임을 가정하였다.

태스크들이 모두 선점불가능하고 또한 주기적인 경우에 대한 실시간 스케쥴링의 연구는 Jeffay등에 의해 발표되었으며 그들은 선점불가능한 태스크들로 구성된 태스크 집합이 EDS 알고리즘을 이용하여 스케쥴링 가능하기 위한 필요충분조건을 제시하였다[5]. Harbour등은 단일 프로세서 환경에서 각 태스크가 다수개의 서브태스크들로 구성되고 그들이 모두 서로 다른 우선순위를 갖는 경우에 주어진 태스크 집합이 고정 우선순위 전략에 의해 스케쥴링 가능하기 위한 충분조건을 제시하였다[4]. 그들은 서브태스크들이 서로 다른 우선순위를 갖는 구체적인 예로서 선점불가능한 부분을 가지고 있는 태스크의 경우를 들고 있다. 즉 선점불가능한 부분을 다른 모든 선점가능한 부분에 비해 우선순위가 현저하게 높은 서브태스크로 간주함으로써 선점가능한 서브태스크와 선점불가

는한 서브태스크를 모두 가지고 있는 태스크의 스케줄링 가능성을 검사할 수 있다고 하였다.

그들은 임의의 태스크 τ_i의 응답 시간을 구할 때 τ_i의 해당 서브태스크에 대해 나머지 태스크들을 (H) 형태, $((HL)^*)$ 형태, $((HL)^+H)$ 형태, $((LH)^*L^0)$ 형태, 그리고 (L) 형태중의 하나로 분류하고 있다. 그러나 그들의 방법은 태스크들을 분류함에 있어 다소의 애매함이 엿보이고 최악의 경우의 응답 시간을 구해서 해당 태스크의 스케줄링 가능성을 검사함에 있어서도 응답 시간이 불필요하게 길게 계산되어지는 경향이 있으며 이를 다음의 예에서 볼 수 있다.

(예제 1)

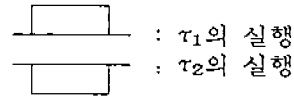
단일 프로세서 환경에서 태스크 집합이 두개의 태스크 τ_1과 τ_2로 구성되어 있고 그들 각각은 두개의 서브태스크들로 구성되며 첫 번째 서브태스크는 선점가능하고 두 번째 서브태스크는 선점불가능하다. τ_1의 우선순위가 τ_2보다 높다고 하고 한 태스크의 서브태스크들은 우선순위가 같다고 하자. 그들의 실행 요구시간 및 주기와 같은 시점에서 시작된 경우의 태스크들의 실행 상황은 아래의 그림과 같다. 주어진 태스크 집합은 스케줄링이 가능하며 τ_2의 최악의 경우의 응답시간은 세 번째 작업에서 10으로 나타나고 있다.

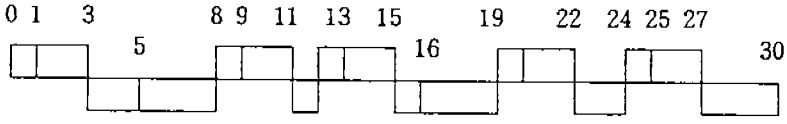
τ_2의 스케줄링 가능성을 보기 위해서 Harbour등의 방법에서는 일단 τ_2를 정규 형태로 바꾼다. 이때 $\tau_{2,2}$가 매우 높은 우선순위의 서브태스크로 간주되므로 $\tau_{2,1}$은 그 자신의 우선순위를 유지한다. 그리고 $\tau_{2,2}$의 종료 시간을 계산하기 위하여 τ_1을 4가지 형태중의 하나로 분류한다. Harbour등은 τ_1을 (H) 형태로 분류하고 있다. 이는 모순으로써 실제로는 τ_1은 그들 네 가지 형태중의 어느 하나에도 속하지 않는다. τ_1이 (H) 형태이라면 $\tau_{2,2}$의 실행중 언제라도 선점할 수 있어야 하는데 $\tau_{2,2}$가 선점불가능한 서브태스크이므로 이는 불가능하다. 한편 $\tau_{1,1}$이 일단 시작된 경우라면 도중에 $\tau_{2,2}$의 방해없이 $\tau_{1,2}$가 연속해서 실행가능하므로 $((HL)^+)$ 형태도 $((HL)^+H)$ 형태에도 해당되지 않는다. 비슷한 논리로 τ_1은 $\tau_{2,2}$에 대해 $((LH)^+L^0)$ 형태도 아니다. Harbour등의 방법에서와 같이 τ_1을 (H) 형태로 간주하고 τ_2의 최악의 경우의 응답시간을 구했을 때 이 값이 τ_2의 종료시한보다 작거나 같다면 τ_2의 스케줄링 가능성을 확인할 수는 있다. 그러나 이 경우에는 τ_1의 실행시간이 τ_2의 최악의 경우의 응답시간을 구하는데 불필요하게 더해질 수 있으며 따라서 실제는 τ_2가 스케줄링이 가능한 많은 경우에 Harbour등의 방법으로는 스케줄링이 불가능한 것으로 판정될 수 있다.

3. 시스템 모델

본 논문에서 다루고자 하는 태스크 시스템 $\tau=\{\tau_1, \tau_2, ..., \tau_n\}$의 모델은 다음과 같다. 태스크 시스템을 구성하는 각 태스크는 다수 개의 서브태스크들로 구성되며 그들은 각각 선점가능한 서브태스크이거나 선점불가능한 서브태스크들이다. 각 태스크는 주기적이라 가정되며 태스크 τ_i의 j번째 수행을 τ_i의 j번째 작

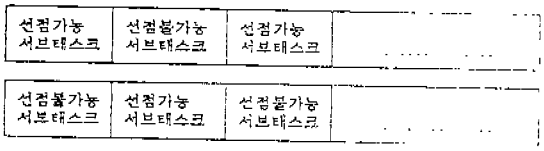
	$C_{i,1}$	$C_{i,2}$	T_i
τ_1	1	2	6
τ_2	2	3	10





(그림 1) 태스크들이 임계 순간에서 시작된 경우
(Fig. 1) Execution status of Critical Instant

업이라 부르기로 한다. 하나의 태스크의 구성은 (그림 2)와 같다. 즉, 선점불가능한 서브태스크로 시작된 태스크는 그 뒤를 선점가능한 서브태스크와 선점불가능한 서브태스크가 순서대로 교차하여 나타나고 선점가능한 서브태스크로 시작된 태스크는 그 뒤를 선점불가능한 서브태스크와 선점가능한 서브태스크가 순서대로 교차하여 나타난다.



(그림 2) 태스크의 구성
(Fig. 2) Organization of a task

각 태스크 τ_i는 주기적인 태스크로 가정되며 τ_i의 j번째 작업은 $\tau_i(j)$로 나타낸다. 각 태스크 τ_i의 종료시한 D_i는 주기 T_i보다 작거나 같다고 가정된다. 태스크들은 우선순위대로 나열되어 τ_1의 우선순위가 가장 높고 τ_n의 우선순위가 가장 낮다고 가정한다. 태스크의 우선순위는 고정적으로 부여되며 서브태스크들의 실행 요구시간은 임의의 값들로 고정된다. τ_i의 서브태스크의 수는 N(i)로, 선점가능한 서브태스크의 수는 NP(i)로, 선점불가능한 서브태스크의 수는 NN(i)로 각각 표시하기로 한다. 따라서 N(i)=NP(i)+NN(i) 이다. τ_i의 선점가능한 서브태스크들 중 j번째 것은 $P_{i,j}$로 또한 그것의 실행 요구시간은 $CP_{i,j}$로 나타내고 τ_i의 선점불가능한 서브태스크들 중 k번째 것은 $N_{i,k}$로 또한 그것의 실행 요구시간은 $CN_{i,k}$로 표시하기로 한다. 이때 CP(i)는 τ_i의 선점가능한 서브태스크들의 실행 요구시간의 합을 CN(i)는 τ_i의 선점불가능한 서브태스크들의 실행 요구시간의 합을 나타내기로 한다.

4. 스케줄링 방법

4.1 각 태스크의 마지막 서브태스크가 선점가능한 서브태스크인 경우

본 절에서 각 태스크는 서브태스크들로 구성되고 그들의 우선순위는 모두 같으며 마지막 서브태스크는 모두 선점가능한 서브태스크들이라고 가정한다. 따라서 각 태스크의 구성을 보면 선점가능한 서브태스크와 선점불가능한 서브태스크들이 교차하여 나타나며 마지막은 선점가능한 서브태스크들로 끝난다. τ_i의 각 서브태스크는 같은 우선순위를 갖는다. 그러나 편의상 선점불가능한 서브태스크들의 우선순위는 무한히 높은 것이라고 가정한다. 하나의 태스크의 두개의 연속된 서브태스크들이 각각 선점불가능, 선점가능할 때 선점불가능한 서브태스크의 시작 시간과 선점가능한 서브태스크의 종료 시간은 선점불가능한 서브태스크를 자신과 같은 크기의 실행 요구 시간을 갖는 선점가능한 서브태스크로 바꾸어 놓고 계산한 경우와 각각 같으며 이는 Harbour등에 의해 증명된 다음의 정리에 기초하여 입증된다.

(정리 1)

τ_i의 연속된 두개의 서브태스크 $\tau_{i,j}$와 $\tau_{i,j+1}$에 대해 $\tau_{i,j}$의 우선순위가 $\tau_{i,j+1}$보다 높다고 가정한다. $\tau_{i,j}$의 우선순위를 $\tau_{i,j+1}$의 우선순위와 같게 놓았을 때의 τ_i와 $\tau_{i,k}(j+1 \le k \le N(i))$들의 종료시간은 태스크들의 위상을 어떻게 주더라도 원래의 우선순위를 갖을 경우와 같다[4].

위의 정리는 주어진 태스크를 그의 정규 형태로 바꾸는데 다음과 같이 이용되며 이로 말미암아 태스크의 스케줄링 가능성을 보다 용이하게 판단할 수 있게 된다.

(정의 2)

하나의 태스크는 그것의 연속되는 서브태스크들이 우선순위에 있어서 줄지 않는(non-decreasing) 순서를 가질 때 정규 형태의 상태에 있다고 말하여 진다. 이때 τ_i의 정규 형태 τ_i'은 다음과 같이 구한다[4]. 아래에서 $P_{i,j}$는 $\tau_{i,j}$의 우선순위를 말한다.

$P_{i,N(i)}' = P_{i,N(i)}$
for l = N(i) downto 2
  if $P_{i,l}' < P_{i,l-1}$ then $P_{i,l-1}' = P_{i,l}'$
  else $P_{i,l-1}' = P_{i,l-1}$
end;

단일 프로세서 환경에서 각 태스크에게 고정적으로 우선순위가 주어진 경우 최악의 경우의 응답시간

을 이용하여 각 태스크의 스케쥴링 가능성을 검사하기 위해서는 최악의 경우의 응답시간이 발생하는 최악의 경우의 위상을 찾는 것이 필요하게 된다. 본 절에서 주어진 태스크 모델의 경우에는 Harbour등의 방법을 이용하여 τ_i의 최악의 경우의 응답시간을 구하고 이를 이용하여 τ_i의 스케쥴링 가능성을 검사하는 것이 다음과 같이 가능하다.

(1) τ_i를 정규 형태로 바꾼다. 이때 τ_i의 마지막 서브태스크는 선점가능한 서브태스크이므로 결국 τ_i는 전체가 하나의 선점가능한 태스크로 간주될 수 있고 이 경우의 우선순위는 τ_i의 선점가능한 서브태스크의 우선순위이다.

(2) τ_i를 제외한 나머지 태스크들은 우선순위별로 다음과 같은 두 가지 종류로 분류된다.

(i)(H) 형태 태스크

(H) 형태에 속한 태스크들은 선점가능한 서브태스크의 우선순위가 τ_i의 선점가능한 서브태스크의 우선순위보다 같거나 높은 것들이다. 이러한 태스크 들은 τ_i를 언제나 선점할 수 있으므로 τ_i와 같은 시점에서 출발시키는 것이 τ_i의 응답시간을 최대로 만드는 경우가 된다.

(ii)$((LH)^+L)$ 형태 태스크

이 형태에 속한 태스크들은 선점가능한 서브태스크의 우선순위가 τ_i의 선점 가능한 서브태스크의 우선순위보다 낮은 것들이다. 그러나 그러한 태스크의 선점불가능한 서브태스크들은 무한히 높은 우선순위를 갖는다고 가정하였고 마지막 서브태스크는 모두 선점가능한 서브태스크이므로 서브태스크들의 우선순위의 구성이 위와 같다. $((HL)^+)$ 형태의 태스크들도 존재하지만 이러한 태스크들은 $((LH)^+L)$ 형태의 태스크로 간주될 수 있다. $((LH)^+)$형태에 속한 모든 태스크들은 그들 중 오직 하나의 선점불가능한 서브태스크만이 τ_i-활동 구간 내에 수행되는 것이 가능하다.

따라서 τ_i의 스케쥴링 가능성은 다음의 조건이 만족되면 보장될 수 있다. 즉,

$$W(i) = \min_{t \le D_i} \{ t \mid \sum_{j=1}^{i} \left(\sum_{k=1}^{npj} CP_{j,k} + \sum_{k=1}^{nnj} CN_{j,k} \right) \left\lceil \frac{t}{T_j} \right\rceil + \max_{j>i \text{ and } k \le N(j)} \{ CN_{j,k} \} \le t \}$$

이라 할 때 위의 조건을 만족하는 최악의 경우의 응답시간 W(i)가 존재하면 τ_i는 스케쥴링이 가능한 것이고 그렇지 못한 경우에는 스케쥴링이 불가능한 것으로 판정된다. 이때 각 태스크의 W(i)는 다음과 같은 식을 이용하여 계산한다.

$$\begin{cases} wrt_i^{n+1} = \sum_{k=1}^{i} \left\lceil \frac{wrt_i^n}{T_k} \right\rceil C_k + \max_{j>i \text{ and } k \le N(j)} \{ CN_{j,k} \} \\ wrt_i^0 = C_i \end{cases}$$

$wrt_i^{n+1} = wrt_i^n$이고 $wrt_i^{n+1} \le D_i$이면 $W(i) = wrt_i^{n+1}$인 동시에 τ_i는 종료시한 이내에 스케쥴링이 가능한 것이고 그렇지 않은 경우에는 τ_i는 스케쥴링이 불가능한 것으로 판정된다.

4.2 각 태스크의 마지막 서브태스크가 선점불가능한 서브태스크인 경우

본 절에서는 태스크 시스템을 구성하는 각 태스크가 서브태스크들로 구성되며 마지막 서브태스크는 모두 선점불가능한 서브태스크들이라고 가정된다. 스케쥴링 가능성을 보고자하는 각 태스크 τ_i의 마지막 서브태스크가 선점불가능한 서브태스크인 경우에도 4.1절의 식을 이용하여 스케쥴링 가능성을 검사할 수는 있다. 그러나 이 경우에는 앞에서 지적한 바와 같이 τ_i보다 높은 우선순위의 태스크들의 실행시간이 불필요하게 τ_i의 종료 시간을 구하는데 더 해질 수 있으며 이를 피하기 위해서 다음과 같은 방법을 제시한다.

주어진 태스크 모델에서 태스크들이 임계 순간에 시작된 경우, 각 태스크의 첫 번째 작업의 종료 시간은 최악의 경우의 응답시간이 아닐 수도 있다. Harbour등의 방법에서는 τ_i에 의해 시작된 τ_i-활동 구간 내에서 τ_i의 종료 시간을 최대로 하는 최악의 경우의 위상을 이용하여 τ_i의 스케쥴링 가능성을 검사한다. 본 절에서 제시된 방법은 Harbour등의 방법에 따른 τ_i의 종료시간의 계산을 개선하고자 하며 이에 대한 방법은 다음과 같다.

먼저 τ_i를 정규 형태 τ_i'으로 바꾼다. 이때 τ_i'은 τ_i가 몇 개의 서브태스크들로 구성되었던 간에 두개의 서브태스크로 구성되게 되며 첫 번째 것은 선점가능한 서브태스크(실행 요구시간은 $C_{i,1}+C_{i,2}+\ldots+C_{i,N(i)-1}$)이고 두 번째 것은 선점불가능한 서브태스크(실행 요구시간은 $C_{i,N(i)}$)이다. 위에서 $C_{i,j}$는 $\tau_{i,j}$의 실행 요구시간이다. 먼저 τ_i'의 첫 번째 작업의 첫 번째 서브태스크의 종료시간을 구한다. 이것은 τ_i'의 첫 번째 서브태스크와 그보다 우선순위가 높은 태스크들을 임계 순간에서 시작시켰을 때의 종료 시간으로 구한다. 이것을 이용하여 첫 번째 서브태스크에 대한 활동 구간을 구하며 이 활동 구간의 끝이 되는 시점에다 두 번째 서브태스크의 실행 요구 시간과 τ_i'보다 우선순위가 낮은 모든 태스크들의 선점불가능한 서브태스크들중에서 가장 큰 실행 요구 시간을 더한다. 이것이 τ_i'의 첫 번째 작업의 종료 시간이 된다. 이 종료 시간이 τ_i'의 종료시한보다 크다면 τ_i'은 물론 스케줄링이 불가능한 것으로 판정된다. 그렇지 않은 경우에는 이 값으로부터 τ_i'의 활동 구간을 구하되 그 구간의 끝이 τ_i'의 주기보다 작다면 더 이상의 계산이 필요없이 τ_i'은 스케줄링이 가능하다고 판정되며 그렇지 않은 경우에는 τ_i'의 활동 구간이 계속되므로 두 번째 작업에 대해서 위의 과정을 되풀이한다. 이상의 방법에 대한 알고리즘은 다음과 같다.

(알고리즘 1)

* τ_i는 정규 형태의 상태에 있다고 가정한다. 따라서 τ_i는 두개의 서브태스크로 구성되며 처음 것은 선점가능하고 두 번째 것은 선점불가능하다.

(1) $j:=1$;
  $\tau_1:=\tau_{i,1}$; $C_1:=C_{i,1}$;
  $\tau_2:=\tau_{i,2}$; $C_2:=C_{i,2}$;
  $D:=D_i$; $T:=T_i$;

(2) $\tau_{i,2}$를 고려하지 않은 상태에서 $\tau_{i,1}$의 활동 구간을 구함;
  이때 [0, t_j]가 구해진 활동 구간이면 $\tau_{i,2}$의 j번째 작업의 종료 시간 $ct_i(j)$는
  $ct_i(j):=t_j+C_{i,2}+\max\{CN_{j,k}\}$;
  $ct_i(j)>D_i$이면 unsuccessfully terminate;
  [0, $ct_i(j)$]가 τ_i의 활동 구간이면 successfully terminate;
  $j:=j+1$;
  $C_{i,1}:=C_{i,1}+C_1+C_2$;
  $T_i:=T_i+T$;
  $D_i:=D_i+T$;
  Goto (2);

(알고리즘 1)에서 τ_i의 활동 구간은 다음과 같이 구해질 수 있다. 즉,

$W_i(k, x)=\min_{t\le x}((\sum_{j=1}^{i-1}\left\lceil\frac{t}{T_j}\right\rceil C_j+k*C_i)/t)$ 이라 할 때

$W_i(1, D_i)\le 1$이면 τ_i의 첫 번째 작업은 종료시한 이내에 종료될 수 있고 $W_i(1, T_i)<1$이면 τ_i-활동 구간이 종료되는 t가 존재하며 $W_i(1, D_i)\le 1$이고 $W_i(1, T_i)\ge 1$이면 활동 구간은 계속된다. 이 경우에는 τ_i의 실행 요구시간을 $2*C_i$로, D_i는 D_i+T_i로, T_i는 $2*T_i$로 놓은 후에 위의 과정을 되풀이하면 된다.

(정리 3)

τ_i의 가장 긴 응답시간은 τ_i에 의해 시작된 τ_i-활동 구간 내에서 발생한다.

(증명은 [4] 참조)

(정리 4)

τ_i에 의해 시작된 활동 구간 내에서의 각 작업의 종료시간을 (알고리즘 1)에 의해 구했을 때 그들 중의 최대치가 D_i보다 작거나 같으면 τ_i는 스케줄링이 가능하다.

(증 명)

일반성을 유지한 채 τ_i는 두개의 서브태스크로 구성되었다고 가정하자. 이때 첫 번째 것은 선점가능한 서브태스크이고 두 번째 것은 선점불가능한 서브태스크이다. 첫 번째 서브태스크의 최악의 경우의 종료시간을 구하는 과정과 그 타당성은 4.1절에서 이미 논의된 바 있다. 두 번째 서브태스크의 경우에 Harbour 등의 방법에서는 나머지 태스크들을 두 가지 종류로 분류하여 생각한다. 한 가지는 (H) 형태이며 다른 한 가지는 $((LH)^+)$ 형태이다. (알고리즘 1)은 $((LH)^+)$ 형태의 태스크들에 대하여는 Harbour등과 같은 처리를 하고 있다. (H) 형태의 태스크들에 대해서 Harbour등의 방법은 그들이 언제라도 τ_i를 선점할 수 있다는 가

경 하에 τ_i의 종료시간을 계산하고 있으나 (알고리즘 1)에서는 선점이 불가능한 상황을 가려내어 (H) 형태 태스크들의 실행 요구시간이 불필요하게 더해지는 것을 방지하고 있다. 이 경우에 계산된 τ_i의 종료 시간이 Harbour등의 방법에 의해 계산된 종료 시간보다 작거나 같음은 자명하다. Harbour등의 방법과 본 논문에서 제시된 방법은 같은 가정 하에서 임의의 태스크 τ_i의 종료 시간을 구하고 있으며 Harbour등의 방법에 의해 구해진 τ_i의 종료시간이 τ_i의 스케쥴링 가능성을 판단하는 충분조건으로 사용될 수 있으므로 (알고리즘 1)에 의해 구해진 τ_i의 종료시간도 τ_i의 스케쥴링 가능성을 검사하는 충분조건으로 사용될 수가 있다.□

(따름 정리 5)

Harbour등의 방법에 의해 스케쥴링이 가능하다고 판정된 태스크 집합은 (알고리즘 1)에 의해서도 스케쥴링이 가능하다고 판정된다.

(증 명)

(정리 4)의 증명에 의해 명백함.□

다음은 Harbour등의 방법에 의해서는 스케쥴링이 불가능한 것으로 판정되지만 (알고리즘 1)에 의해서는 스케쥴링이 가능하다고 판정되는 태스크 집합의 예이다.

(예제 2)

단일 프로세서 환경에서 태스크 집합이 두개의 태스크 τ_1과 τ_2로 구성되어 있고 그들 각각은 두개의 서브태스크들로 구성되며 첫 번째 서브태스크는 선점가능하고 두 번째 서브태스크는 선점불가능하다. τ_1의 우선순위가 τ_2보다 높다고 가정하자. 그들의 실행 요구시간 및 주기는 다음과 같다.

	$C_{i,1}$	$C_{i,2}$	T_i
τ_1	1	2	7
τ_2	2	3	10

임계 순간에서 태스크들이 시작된 경우 Harbour등의 방법을 이용하여 τ_2의 첫 번째 작업의 종료 시간을 구하면 11로 계산되어 종료시한을 놓치게 된다. 그러나 본 논문에서 제시된 방법에 의해 계산을 하면 스케쥴링이 가능한 것으로 판정되고 있다. 본 논문의 방법에 따르면,

(1) $\left\lceil \frac{t}{T_1} \right\rceil C_1 + C_{2,1} = t$를 만족하는 t=5가 존재

(2) t=5 일 때

$\left\lceil \frac{t+\varepsilon}{T_1} \right\rceil C_1 + C_{2,1} < t+\varepsilon$을 만족함

따라서 이 경우의 활동 구간은 [0, 5]

(3) τ_2의 첫 번째 작업의 종료 시간은 5+3=8.

따라서 τ_2의 첫 번째 작업은 종료시한 이내에 종료될 수 있음.

(4) τ_2-활동 구간이 계속되는 가를 검사

t=8, 9, 10 일 때

$\left\lceil \frac{t+\varepsilon}{T_1} \right\rceil C_1 + C_2 < t+\varepsilon$이 성립하는 가를 본다.

위를 만족하는 t가 존재하지 않기 때문에 활동 구간은 계속됨.

(5) $\left\lceil \frac{t}{T_1} \right\rceil C_1 + C_2 + C_{2,1} = t$를 만족하는 t=13이 존재

(6) t=13 일 때

$\left\lceil \frac{t+\varepsilon}{T_1} \right\rceil C_1 + C_2 + C_{2,1} < t+\varepsilon$을 만족함

따라서 이 경우의 활동 구간은 [0, 13]

(7) τ_2의 두 번째 작업의 종료 시간은 13+3−10=6

(8) τ_2-활동 구간이 계속되는 가를 검사

t=16, 17, 18, 19, 20 일 때

$\left\lceil \frac{t+\varepsilon}{T_1} \right\rceil C_1 + 2*C_2 < t+\varepsilon$을 만족하는 t가 있는 가를 본다.

t=19일 때 위의 관계가 성립함.

따라서 이 경우 활동 구간은 [0, 19]

(9) $19 < 2*T_2$이므로 더 이상의 작업에 대한 조사

를 할 필요가 없으며 τ_2의 wrt = max{8, 6} = 8이고 따라서 τ_2는 스케쥴링이 가능한 것으로 판정된다.

5. 타 모델로의 적용및 검토

본 논문에서 제시된 방법들은 선점불가능한 태스크들로 구성된 태스크 집합의 스케쥴링 가능성을 예측하는데 이용되어 질 수 있다. $\tau=\{\tau_1, \tau_2, ..., \tau_n\}$을 선점불가능한 태스크들의 집합이라 하자. 고정 우선순위 전략을 사용하는 경우 이들의 스케쥴링 가능성은 다음과 같이 두 가지로 나누어 검토할 수 있다.

5.1 각 태스크의 뒤에 ε 만큼의 선점가능한 서브태스크를 삽입하는 경우

이 경우에는 태스크 모델이 다음과 같이 바뀌어 지는 경우이다. 즉, 각 태스크는 두개의 서브태스크들로 구성되며 첫 번째 것은 선점불가능, 두 번째 것은 선점가능하다. 이 경우의 태스크 모델은 4.1절의 모델과 같으며 태스크들의 스케쥴링 가능성도 4.1절에서 제시된 방법을 적용할 수가 있게 된다. 즉, τ_i의 스케쥴링 가능성을 보기 위해 τ_i를 정규 형태로 바꾸어 놓으면 τ_i는 전체가 하나의 선점가능한 태스크로 간주되어 질 수 있다. 이때 τ_i를 제외한 나머지 모든 태스크들은 τ_i에 대해 (H) 형태 또는 (HL) 형태중의 어느 한 형태에 속하게 된다. (H) 형태에 속한 태스크들은 τ_i를 언제라도 또한 몇 번이고 선점할 수 있으며 (HL) 형태에 속한 모든 태스크들의 선점불가능한 서브태스크들 중에서 실행 요구 시간이 제일 큰 것이 τ_i에 대한 블로킹(blocking) 요인으로 작용하게 된다.

5.2 각 태스크의 앞에 ε 만큼의 선점가능한 서브태스크를 삽입하는 경우

각 태스크의 앞에 ε만큼의 선점가능한 서브태스크를 삽입한다면 각 태스크는 5.1절에서와 같이 두개의 서브태스크로 구성된 것으로 간주될 수 있다. 이 경우에는 처음 것이 선점가능하고 두 번째 것이 선점불가능하다. 이와 같은 태스크 모델은 4.2절의 모델과 같으며 태스크들의 스케쥴링 가능성도 4.2절에서 제시된 방법을 적용할 수가 있다. 즉, 첫 번째 서브태스크의 종료 시간을 구하고 이를 이용하여 첫 번째 서브태스크에 대한 활동 구간을 구한다. 활동 구간이 끝나는 시점에 두 번째 서브태스크의 실행 요구 시간과 우선순위가 낮은 모든 태스크들의 선점불가능한 서브태스크들중에서 가장 큰 실행 요구 시간을 더하면 그것이 두 번째 서브태스크의 종료 시간이 된다. 이때 해당 태스크의 활동 구간을 구하여 활동 구간의 끝이 해당 태스크의 주기보다 작다면 그때까지 구해진 종료 시간중 제일 큰 것이 최악의 경우의 종료 시간이 되는 것이고 그렇지 않으면 해당 태스크의 다음 작업에 대하여 지금까지의 일을 반복하면 된다.

5.3 예 제

위에서 언급된 방법들에 대한 구체적인 예를 살펴보기로 한다. 단일 프로세서 환경에서 선점 불가능한 두 개의 태스크의 실행 요구 시간및 주기가 다음의 표와 같다고 하자. 주어진 태스크들은 임계 순간에서 시작하여 주기의 공배수인 40까지를 살펴보았을 때 스케쥴링 가능한 것으로 판명된다. 물론 그렇다고 해서 어떤 위상하에서도 스케쥴링이 가능함을 보장하는 것은 아니다.

	C_i	T_i
τ_1	3	8
τ_2	4	10

$U = 3/8 + 4/10 = 0.775$

주어진 태스크들을 5.1절에서 제시된 방법에 의해 변환하면 다음과 같은 표로 나타낼 수 있다. 이때 각 태스크의 첫번째 서브태스크는 선점 불가능하고 두 번째 서브태스크는 선점 가능하다.

	$C_{i,1}$	$C_{i,2}$	T_i
τ_1	3	ε	8
τ_2	4	ε	10

각 태스크의 스케쥴링 가능성은 다음과 같이 검사될 수 있다.

(1) τ_1의 스케쥴링 가능성 검사

$W(1) = (3 + \varepsilon) + 4 = 7 + \varepsilon \leq 8$

따라서 τ_1은 스케쥴링 가능한 것으로 판정됨.

(2) τ_2의 스케쥴링 가능성 검사

$$W(2)=\left\lceil\frac{7+2\varepsilon}{8}\right\rceil(3+\varepsilon)+(4+\varepsilon)=7+2\varepsilon\le 10$$

따라서 τ_2도 스케쥴링 가능한 것으로 판정됨.

원래의 태스크들을 5.2절에서 제시된 방법을 이용하기 위하여 변환하면 다음과 같은 표로 나타낼 수 있다. 이때 각 태스크의 첫번째 서브태스크는 선점 가능하고 두번째 서브태스크는 선점 불가능하다.

	$C_{i,1}$	$C_{i,2}$	T_i
τ_1	ε	3	8
τ_2	ε	4	10

각 태스크의 스케쥴링 가능성은 다음과 같이 검사될 수 있다.

(1) τ_1의 스케쥴링 가능성 검사

첫번째 서브태스크는 ε에 끝나고 $[0, \varepsilon]$이 첫번째 서브태스크의 활동구간이므로 두 번째 서브태스크는 $(3+\varepsilon)+4=7+\varepsilon$에 끝난다. 이때 $[0, 7+\varepsilon]$이 τ_1의 활동 구간의 끝이고 $7+\varepsilon\le 8$ 이므로 τ_1은 스케쥴링 가능하다고 판정됨.

(2) τ_2의 스케쥴링 가능성 검사

첫번째 서브태스크의 종료 시간은

$$\left\lceil\frac{\varepsilon}{8}\right\rceil(\varepsilon+3)+\varepsilon=3+2\varepsilon$$

이때 $[0, 3+2\varepsilon]$이 첫번째 서브태스크의 활동 구간이므로 두번째 서브태스크는 $(3+2\varepsilon)+4=7+2\varepsilon$에 끝난다. 이때 $[0, 7+2\varepsilon]$이 τ_2의 활동 구간의 끝이고 $7+2\varepsilon\le 10$ 이므로 τ_2도 스케쥴링 가능하다고 판정됨.

6. 모의실험

6.1 모의 실험의 방법

앞에서도 언급한 바와 같이 Harbour등의 방법은 태스크의 최악의 경우의 응답 시간을 구할 때에 응답 시간이 불필요하게 길게 계산되는 경향을 보이고 있다. 따라서 본 논문에서는 제안된 방법과 Harbour등의 방법과의 차이를 비교 분석하기 위하여 다음과 같이 태스크 집합들을 생성하고 그들을 각 방법에 적용하여 보았다. 먼저 태스크 집합은 3개의 태스크들로 구성된 경우와 5개의 태스크들로 구성된 경우를 나누어 생성하였다.

이들에 대하여 각 태스크가 3개, 4개, 그리고 5개의 서브태스크들로 각각 구성되었을 때, 또한 그들 각각에 대하여 태스크 집합이 3개의 태스크들로 구성된 경우에는 각 태스크의 주기가 실행 요구시간의 3배, 4배, 5배, 6배, 그리고 7배인 경우로 나누고 또한 태스크 집합이 5개의 태스크들로 구성된 경우에는 각 태스크의 주기가 실행 요구 시간의 6배, 7배, 8배, 9배, 그리고 10배인 경우를 나누어 태스크 집합을 생성하였다. 고려된 각 경우에 대하여 100개씩의 태스크 집합을 생성하였으며 이들 중에 몇 개가 각 방법에 의해서 스케쥴링이 가능한 것으로 판단되는 지를 조사하였다.

생성된 모든 태스크 집합들은 임계 순간에서 시작하여 태스크들의 주기의 공배수까지 살펴보았을 때 태스크들이 모두 스케쥴링 가능한 집합들이다. 그러나 임계 순간에서 시작하여 스케쥴링이 가능하다고 해서 모든 위상 하에서 스케쥴링이 가능한 것은 아니므로 생성된 태스크 집합들 중에는 그들에 속한 태스크들의 위상을 바꾸어 줌으로 인해 스케쥴링이 불가능해지는 태스크 집합들이 존재할 수도 있다. 이에 반해 본 논문에서 제안된 방법과 Harbour등의 방법은 주어진 조건을 만족하는 경우에는 모든 위상 하에서 스케쥴링이 가능함을 보장할 수 있다.

6.2 모의실험의 결과

(표 1)은 3개의 태스크들로 구성된 태스크 집합에 대해 서브태스크의 수와 각 태스크의 주기의 범위에 변화를 주었을 때 각 방법에 의해 스케쥴링 가능하다고 판단된 비율을 보여주고 있다. 두 가지 방법의 차이가 가장 크게 나타나는 경우는 각 태스크의 주기의 범위가 그것의 실행 요구 시간의 합의 세배 범위 내에서 임의로 선택된 경우이다. 이 경우에는 프로세서 이용률(utilization)이 다른 경우에 비해 상대적으로 큰 경우이며 특히 서브태스크가 3개인 경우에는 실험된 100개의 태스크 집합 중에서 본 논문의 방법에 의

해 스케쥴링이 가능하다고 판단된 것이 92%에 달하는 것에 비해서 Harbour등의 방법에 의해서는 52%만이 스케쥴링 가능한 것으로 판단되었다.

각 태스크의 주기가 실행 요구시간의 합의 여섯 배 이내에서 임의로 선택된 경우까지는 두 방법간에 상당한 차이가 있음을 볼 수 있으며 주기가 실행 요구시간의 합의 일곱 배 또는 그 이상에서 임의로 선택된 경우에는 두 방법간의 차이가 줄고 있음을 볼 수 있다. 이 경우는 프로세서 이용률에 상당한 여유가 있는 경우이며 이때에는 선점불가능한 서브태스크가 다른 서브태스크의 실행을 막는 것이 거의 발생하지 않는 것으로 분석될 수 있다.

(표 2)는 5개의 태스크들로 구성된 태스크 집합에 대한 모의 실험의 결과이다. 각 태스크의 주기가 그의 실행 요구 시간의 여섯배 이내에서 임의로 선택되고 각 태스크가 3개의 서브태스크로 구성된 경우는 본 논문의 방법이 74%를 스케쥴링 가능한 것으로 판정한 것에 비해 Harbour등의 방법에 의한 결과는 29%에 그치고 있다. 이 경우는 프로세서 이용률이 다른 경우에 비해 높을 때이며 따라서 프로세서 이용률이 높은 태스크 집합일수록 본 논문의 방법과 Harbour등의 방법에 반응하는 차이가 커짐을 다시 한번

〈표 1〉 각 태스크 집합이 3개의 태스크들로 구성된 경우
〈Table 1〉 Results for the case with n = 3

		$C_i \le T_i \le 3C_i$	$C_i \le T_i \le 4C_i$	$C_i \le T_i \le 5C_i$	$C_i \le T_i \le 6C_i$	$C_i \le T_i \le 7C_i$
서브태스크 3개의 경우	M 1	53	79	83	87	96
	M 2	92	97	98	96	99
서브태스크 4개의 경우	M 1	60	82	81	87	93
	M 2	83	96	91	97	99
서브태스크 5개의 경우	M 1	67	89	89	92	92
	M 2	84	95	94	97	96

* M 1 : Harbour등에 의해 제안된 방법
* M 2 : 본 논문에서 제안된 방법
* 각 경우에 사용된 태스크 집합의 수는 100
* 각각의 수는 100개의 태스크 집합중 해당 방법에 의해 스케쥴링이 가능한 것으로 판정된 태스크 집합의 수

〈표 2〉 각 태스크 집합이 5개의 태스크들로 구성된 경우
〈Table 2〉 Results for the case with n = 5

		$C_i \le T_i \le 6C_i$	$C_i \le T_i \le 7C_i$	$C_i \le T_i \le 8C_i$	$C_i \le T_i \le 9C_i$	$C_i \le T_i \le 10C_i$
서브태스크 3개의 경우	M 1	29	49	57	73	71
	M 2	74	87	94	91	90
서브태스크 4개의 경우	M 1	54	60	72	83	79
	M 2	87	90	90	95	95
서브태스크 5개의 경우	M 1	51	68	76	82	93
	M 2	83	86	91	95	97

* M 1 : Harbour등에 의해 제안된 방법
* M 2 : 본 논문에서 제안된 방법
* 각 경우에 사용된 태스크 집합의 수는 100
* 각각의 수는 100개의 태스크 집합중 해당 방법에 의해 스케쥴링이 가능한 것으로 판정된 태스크 집합의 수

있 수 있다.

조사된 경우들을 볼 때 본 논문에서 제안된 방법과 Harbour등의 방법은 최대 45%의 효율성의 차이를 보이고 있으며 대부분의 경우에 있어서 20%이상의 차이를 보이고 있다. 각 태스크의 주기가 그의 실행 요구 시간의 10배 이상에서 점점 크게 선택되는 경우에는 프로세서 이용률이 점진적으로 떨어지게 된다. 프로세서 이용률이 매우 작은 값으로 주어지는 태스크 집합에 속한 태스크들은 그것이 선점가능한 서브태스크와 선점불가능한 서브태스크들로 구성된 경우와 전체가 선점가능한 서브태스크들로 구성되었거나 또는 전체가 선점불가능한 서브태스크들로 구성된 경우의 실행되는 모습의 차이가 별로 나타나지 않게 된다. 이러한 경우들에 대하여는 본 논문의 방법과 Harbour등의 방법간에 발생하는 효율성의 차이가 줄어들 것으로 예상된다.

7. 결 론

선점불가능한 태스크들로 구성된 태스크 집합의 스케쥴링 가능성을 검사하는 것과 관련된 연구는 선점가능한 태스크들로 구성된 태스크 집합의 스케쥴링 가능성을 검사하는 것과 관련된 연구에 비해 별로 알려진 것이 없다. 이는 실시간 시스템에 대한 스케쥴링 알고리즘들이 선점가능한 알고리즘들인 RMS 알고리즘과 EDS 알고리즘을 기반으로 하여 발전해 온 것에 기인한다. Jeffay등은 주기적이고 선점불가능한 태스크들에게 동적 알고리즘인 EDS 알고리즘을 적용했을 때 그들이 스케쥴링 가능하기 위한 필요충분 조건을 제시하였다. 그러나 실제의 환경에서는 하나의 태스크가 선점가능한 부분과 선점불가능한 부분을 모두 포함하는 것이 보다 현실적이다. Harbour 등은 태스크들이 여러 개의 서브태스크들로 나뉘어져 있고 그들이 서로 다른 우선순위를 갖는 경우에 주어진 태스크 집합이 스케쥴링 가능하기 위한 충분 조건을 제시하였다. 또한 그들은 이 모델을 선점불가능한 서브태스크를 포함하는 경우에 적용하였는데 적용 과정에 다소의 애매함이 있고 또한 응답 시간이 불필요하게 길어지는 문제가 있음을 살펴보았다.

본 논문의 방법은 Harbour등의 방법에 의한 각 태스크의 최악의 경우의 응답시간을 구하는 과정을 개선하였으며 이때 다음과 같은 결과를 얻을 수 있었다. 첫째, Harbour등의 방법에 의해서 스케쥴링이 가능하다고 판단된 태스크 집합은 본 논문에서 제안된 방법에 의해서도 스케쥴링이 가능하다. 둘째, 모의 실험의 결과를 볼 때 Harbour등의 방법에 의해 스케쥴링이 가능하다고 판단된 태스크 집합의 비율과 본 논문에 의해 판단된 비율은 상당한 차이를 보이고 있으며 특히 프로세서 이용률이 1에 가까워질수록 그 차이가 더욱 벌어지고 있음을 볼 수 있었다.

향후 연구로는 서브태스크들의 우선순위가 서로 다르고 또한 하나의 태스크가 선점가능한 서브태스크와 선점불가능한 서브태스크를 모두 포함하는 경우의 모델에 고정 우선순위 전략을 적용하여 스케쥴링 가능성을 검사하는 것과 이러한 모델의 다중 프로세서 환경으로의 확장 등을 들 수 있다.

참 고 문 헌

[1] R. Bettati and J.W.S. Liu, "Algorithms for end-to-end scheduling to meet deadlines," Proc. of the 2nd IEEE conf. on Parallel and Distributed processing, Dec. 1990.

[2] J. Blazewicz, M. Drabowski and J. Welgarz, "Scheduling multiprocessor tasks to minimize schedule length," IEEE trans. on computers C-35, 1986, pp. 389-393.

[3] D.W.Gillies and J.W.S. Liu, "Scheduling tasks with AND/OR precedence relations," Proc. of the 2nd IEEE conf. on Parallel and Distributed processing, Dec. 1990.

[4] M.G. Harbour, M.H. Klein and J.P. Lehoczky, "Timing analysis for fixed-priority scheduling of hard real-time systems," IEEE Trans. on Software Engineering, Vol. 20, No. 1, Jan. 1994.

[5] K. Jeffay, D. Stanat and C. Martel, "On non-preemptive scheduling of periodic and sporadic task," Proc. of the 12th Real Time Systems Symposium, Dec. 1991, pp. 129-139.

[6] J.P. Lehoczky, L. Sha and Y. Ding, "The rate monotonic scheduling algorithms:exact characterization and average case behavior," Proc. of

the 10th IEEE Real Time Systems Symposium, Dec. 1989.

[7] J.P. Lehoczky, L. Sha, J.K. Strosnider, and H. Tokuda, "Fixed priority scheduling for hard real-time systems," in Foundations of Real-Time Computing: Scheduling and Resource management, A. van Tilborg and G.M. Koob, Eds. New York: Kluwer, 1991, pp. 1-30.

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

[9] J.M. Moore, "Sequencing n jobs on one machine to minimize the number of tardy jobs," Management Science 15, 1968, pp. 102-109.

10] B. Simons, "Multiprocessor scheduling of unit time jobs with arbitrary release times and deadlines," SIAM Journal of computing, 12(2), 1983.

김 인 국

1982년 단국대학교 수학교육학과 졸업(이학사)

1985년 미국 Emory Univ. 대학원 졸업(이학석사)

1995년 아주대학교 대학원 컴퓨터공학과 졸업(공학박사)

1986년~현재 단국대학교 전자계산학과 부교수

관심분야: 운영 체제, 분산 시스템, 실시간 시스템, 실시간 스케줄링