실시간 프로그램의 스케줄가능성 분석 방법

박 흥 복† 유 원 희††

요 약

본 논문에서는 분산 실시간 프로그램의 스케줄가능성 분석 방법을 제안한다. 스케줄가능성 분석을 위한 여러가지 방법이 개발되었지만, 이 방법들은 가능한 모든 상태공간을 추적하거나 고정 우선순위 스케줄 방법을 사용했기 때문에 지수적인 시간과 공간의 복잡성을 야기한다. 따라서 상태 공간을 줄여서 더 이른 시간단위에서 스케줄가능성을 조사하는 방법이 필요하다. 본 논문에서 제시한 스케줄가능성 분석 방법은 번역시간에 결정될 수 있는 프로세스들의 최대 수행시간, 주기, 마감시간, 동기화 시간을 고려하여 동기화 동작 이후에 남는 계산시간과 마감시간의 차이를 계산하여 실시간 프로세스가 마감시간을 지키는가를 판단하는 새로운 알고리즘을 제안하고, 실험을 통하여 그 성능을 평가한다. 실험에 의하여 Fredette의 방법과 비교하면 약 50퍼센트 정도 더 이른 단위시간에 스케줄이 불가능함을 판단할 수 있다.

A Schedulability Analysis Method for Real-Time Program

Hung Bog Park† and Weon Hee Yoo††

ABSTRACT

In this paper, we propose a schedulability analysis method for real-time programs. Several approaches to anlayzing schedulability have been developed, but since these approaches use a fixed priority scheduling method and/or traverse all possible state spaces, there take place exponential time and space complexity of these methods. Therefore it is necessary to reduce the state space and detect schedulability at earlier time. Our schedulability analysis method uses a minimum unit time taken to terminate synchronization action, a minimum unit time taken to terminate actions after synchronization, and a deadline of processes to detect unschedulability at earlier time and dynamic scheduling scheme to reduce state space. We conclude that our method can detected unschedulability earlier 50 percent unit time than Fredette's method.

1. 서 론

실시간 시스템이란 시스템의 운행과 외부 실체와의 상호관계가 시간과 밀접한 관련이 있는 시스템이다[16]. 이러한 시스템에서 실시간 프로그램이 마감시간을 어기는 것은 매우 큰 문제이므로 번역시간에 시스템의 동작을 예측할 수 있는 능력(predictability)과 프로세스가 지정된 마감시간을 만족하는 신뢰성을 확인하는 것이 중요한 과제이다. 분산 실시간 시스템(distributed real-time system)은 높은 신뢰성, 처리능력의 증가, 자원의 공유 등 여러가지 장점 때문에 근래에 많은 관심을 받고 있으며, 공정제어, 공장의 자동화 등에 많이 이용되고 있다. 분산 실시간 시스템은 통신망으로 연결된 노드라는 처리요소로 구성된다. 이러한 시스템에서는 계산의 속도를 증가시키고, 효율적인 수행을 위하여 프로그램이 프로세스라는 단위로 나뉘어져 병렬로 수행된다. 한 노드에서 프로세스는 공유 기억장치를 통하여 통신하고, 다른 노드의 프로세스와는 통신망을 통하여 메세지를 주고 받으며 통신한다[7, 14].

실시간 프로세스는 주기 프로세스(periodic process)와 비주기(sporadic process)로 구분된다.

† 정 회 원: 동명전문대학 전자계산과 부교수
†† 정 회 원: 인하대학교 전자계산공학과 교수
논문접수: 1994년 7월 22일, 심사완료: 1995년 1월 5일

주기 프로세스는 주기(interval)라는 상수 시간간격(time interval)마다 반복적으로 호출되며, 비주기 프로세스는 임의의 시간에 도착한다.

스케줄가능성 분석(schedulability analysis)이란 프로세스가 특정 스케줄링 정책하에서 수행될 때 프로세스가 마감시간을 만족하는가를 판단하는 것이다[1, 2, 3, 4, 13].

Liu와 Layland는 비율 단조(rate monotonic) 스케줄링시 독립적인 프로세스의 계산시간, 주기, 마감시간이 주어졌을 때 프로세스들이 마감시간을 지키는가를 판단하는 방법을 제안하였으나 이 방법은 두 프로세스간의 동기화를 처리하거나 둘 이상의 시간제약을 갖는 프로세스를 처리할 수 없는 제한이 있다[3, 6, 7, 13].

Stoyenko 등[5, 6, 12, 13]은 모든 동작의 시간이 알려지는 시스템에서 동작하는 Real-Time Euclid라는 언어를 설계하고, 이 언어로 작성된 실시간 프로그램이 최선 마감시간(earliest deadline) 방법하에서 스케줄될 때 마감시간을 만족하는가를 검사하는 frame superimposition이라는 스케줄가능성 분석기를 설계하였다. Real-Time Euclid는 모든 언어구조가 번역시간에 명시되는 시간범위를 갖는다.

CCSR[4]은 동작에 우선순위를 할당하여 고정 우선순위 스케줄링을 모델링하는데 사용된다. CCSR은 도달가능성 분석(reachability analysis)을 통하여 프로그램을 검증한다. 도달가능성 분석에서 시간제약을 위반하는 상태에 도달하는 가를 결정하기 위하여 프로세스가 정의하는 전이 시스템을 사용한다. CCSR은 단위시간의 동작만을 정의할 수 있으며, 하나의 고정 우선순위 스케줄링 정책만이 사용된다는 단점이 있다.

또, Fredette 등은 프로세스 대수(process algebra)를 이용하여, 프로세스의 동작을 나타내는 상태공간을 탐색하는 스케줄가능성 분석을 제안하였다[1, 2, 3]. 이 방법은 인터리빙 모델에 기초를 둔 단일처리기 시스템만을 고려하였기 때문에 분산 시스템에는 적합하지 않으며, 프로세스의 수에 대하여 지수승으로 상태의 수가 증가하므로 분석의 계산성능이 좋지 못하다. 이상의 방법들은 정적 스케줄링 방법만을 사용하며, 특정 스케줄링 방법에 기반을 두어 스케줄링 정책이 변할 때는 사용할 수 없다는 단점이 있다. 또한 한가지 스케줄링 방법으로는 프로세스들이 마감시간을 지키지 못할 때의 대안을 제시하지 못한다.

본 논문에서는 이와 같은 목적으로 Fredette의 방법[1, 2, 3]과 비슷하게 프로세스 대수의 전이 규칙을 이용하여 프로세스의 모든 상태공간을 검색하지 않고, 번역시간에 결정될 수 있는 프로세스들의 수행순서와 프로세스들의 최대 수행시간, 주기, 마감시간, 동기화 시간을 고려하여 남은 계산시간과 마감시간의 차이를 계산하여 실시간 프로세스가 마감시간을 지키는가를 판단하는 새로운 알고리즘을 제안하고, 실험을 통하여 그 성능을 평가한다.

2. 실시간 명세언어

실시간 명세언어(real time specification language)는 원시 프로그램의 모든 동작을 표현하지 않고 관심부분만을 요약하여 그 특성을 분석할 수 있어 프로그램의 정적분석에 많이 이용된다. 본 논문의 주관심사는 실시간 프로그램이 마감시간을 만족하는가를 번역시간에 판단하는 것으로 프로세스의 시간특성과 프로세스간의 관련성을 표현하는 명세언어가 필요하다.

2.1 프로세스 대수

프로세스 대수에서는 시스템의 명세와 구현 방법을 설명하기 위해 동일한 언어를 사용한다. 프로세스 대수의 구문과 연산의미(operational semantics)의 예는 다음과 같은 CCS[7]에서 볼 수 있다. 프로세스 term P는 (그림 1)과 같은 BNF 문법으로 주어진다.

$$P ::= \mathbf{nil} \mid X ::= P \mid a.P \mid P+Q \mid P \| Q$$

(그림 1) CCS의 구문
(Fig. 1) CCS Syntax

대문자는 프로세스, 소문자는 프로세스를 구성하는 동작이다. nil은 프로세스의 종료를 나타낸다. X::=P는 주어진 환경내에서 변수 X에 한정되는 프로세스 P를 나타낸다. a.P는 동작 a를 수행한 후에 프로세스 P와 같이 동작한다. P+Q는 P 또는 Q 두 프로세스사이의 선택을 나타낸다. P‖Q는 두 프로세스 P, Q의 병렬조합을 나타낸다. 두 프로세스는 독립적으로 동작을 수행할 수도 있고, 보동작(complementary action)으로 동기화 될 수 있다. 동작 a의 보동작은 $\bar{a}$로 표현된다.

프로세스간의 관련성을 표시하는 도구로는 CCS나 CSP와 같은 프로세스 대수가 많이 사용된다. 프로세스의 시간특성을 표현하기 위한 도구로는 프로세스 대수에 시간의 개념을 추가한 시간 프로세스 대수(temporal process algebra)가 널리 사용된다[8, 13]. 프로세스의 동작을 나타내는 프로세스의 전이 관계는 [정의 1]과 같다.

[정의 1] 전이 시스템은 4-튜플 $<Q, A, \rightarrow, q_0>$로 정의된다.

- Q는 상태의 집합이다.
- A는 동작의 집합이다.
- $\rightarrow$: $Q\times A\times Q$는 상태와 동작으로 부터 새로운 상태를 만드는 전이 관계이다.
- q_0는 초기상태이다.

2.2 명세언어 CCS-R

본 논문에서 사용하는 실시간 명세언어 CCS-R은 Milner의 CCS[7]에 시간의 개념을 추가한 것이다. CCS-R은 수행할 시간과 수행시간에 대한 제약을 갖는 실시간 프로그램의 특성을 추출하기 위한 실시간 명세언어이다. CCS-R은 실시간 프로세스의 동작을 나타내고, 각 동작이나 프로세스가 갖는 시간제약을 표현하고 실시간 프로세스의 시간 특성을 추출하여 마감시간을 만족하는 가를 번역시간에 판단하는데 사용된다.

CCS-R에서 프로세스는 프로세스가 수행하는 명령에 해당하는 동작으로 구성된다. 프로세스의 동작은 프로세스가 수행하는 동작으로 표현된다. 각 동작은 그 동작을 수행하는데 필요한 시간간격 d를 갖는다.

CCS-R의 동작에는 계산을 하는 γ^d, 지연을 나타내는 δ^d, 동기화 동작 a의 세가지 종류가 있다. 동기화 동작은 다른 프로세스가 보동작을 수행하는 경우에만 수행된다. a가 동기화 동작을 나타내면, 이의 보동작은 $\bar{a}$로 표현되고 $\bar{\bar{a}}$ = a이다.

CCS-R 프로세스의 구문은 (그림 2)와 같은 BNF 문법으로 정의된다.

P::=nil | γ^d | δ^d | α^d | P+Q | P;P | sw(P, d) | fw(P, d) | e(P, d)

(그림 2) CCS-R 프로세스
(Fig. 2) CCS-R Process

CCS-R의 기본 동작에는 nil, γ^d, δ^d, α^d가 있다. nil은 지연동작만을 수행할 수 있는 종료 프로세스이다. γ^d는 d 시간단위가 걸리는 동작, δ^d는 d 시간단위동안 지연하는 동작을 나타낸다. α^d는 d 시간단위가 걸리는 동기화 동작이다. 프로세스는 이상의 기본 동작으로 부터, 프로세스 구성자+(선택), ;(순차조합)으로 만들어진다. 프로세스가 포함하는 시간제약과 시간특성은 시간 구성자 sw, fw, e로 표현된다. d를 마감시간이라 할 때, 마감시간은 프로세스가 시작된 시간으로부터 상대적으로 계산된다. sw(P, d)는 프로세스 P가 d 시간단위이내에 시작하여야 함을 나타낸다. fw(P, d)는 프로세스 P가 d 시간단위이내에 끝나야 한다. e(P, d)는 정확히 d 시간단위이내에 종료하는 프로세스로 주기 프로세스를 표현하는데 사용된다. 만일 P가 d'에 끝나고 d > d'이면, P는 d−d' 시간단위동안 지연되어야 한다.

실시간 프로세스가 마감시간을 지키는가를 번역시간에 판단하기 위해서는 프로세스의 동작뿐만 아니라 프로세스의 수행시간을 번역시간에 계산할 수 있거나 프로그램상에 명시되어야 한다. 프로세스의 수행시간을 나타내는 기간함수를 [정의 2]와 같이 정의한다.

[정의 2] 기간함수 d

- D는 기간 정의역이다. 기간 정의역의 원소

를 d라 하면 ∀d, d > 0이다.

- A는 모든 동작의 집합 $\{\alpha^d, \gamma^d, \delta^d \mid d \in D\}$ 이다. $a \in A$에 대하여 $\bar{a}$는 보동작으로 A의 원소이다.
- 기간함수 $d : A \rightarrow \mathbb{D}$는 모든 동작의 기간이다. $\forall a \in A,\ d(a) = d(\bar{a})$이다.

$T1$: nil
$T2$: α^0
$T3$: $P \wedge Q \Rightarrow (P+Q)$
$T4$: $P \wedge Q \Rightarrow (P;Q)$
$T5$: $P \wedge Q \Rightarrow (P \parallel Q)$
$T6$: $P \Rightarrow sw(P,d)$
$T7$: $P \Rightarrow fw(P,d)$
$T8$: $P \Rightarrow e(P,0)$

(그림 3) CCS-R의 종료술어
(Fig. 3) Termination Predicates of CCS-R

$A1$: $\alpha^d \xrightarrow{\alpha^1} \alpha^{d-1}$

$A2$: $a \xrightarrow{\alpha^1} a^{d(a)-1}$

$A3$: $d > 0 \Rightarrow a^d \xrightarrow{\delta^1} a^d$

$A4$: $a \xrightarrow{\delta^1} a$

CL: $P \xrightarrow{\alpha^1} P' \wedge \alpha \neq \delta \Rightarrow P+Q \xrightarrow{\alpha^1} P'$

CR: $Q \xrightarrow{\alpha^1} Q' \wedge \alpha \neq \delta \Rightarrow P+Q \xrightarrow{\alpha^1} Q'$

CI: $P \xrightarrow{\delta^1} P' \wedge Q \xrightarrow{\delta^1} Q' \wedge \neg(P+Q) \Rightarrow P+Q \xrightarrow{\delta^1} P+Q'$

$Sq1$: $P \xrightarrow{\alpha^1} P' \wedge \neg P \wedge \neg P' \Rightarrow P;Q \xrightarrow{\alpha} P';Q$

$Sq2$: $P \xrightarrow{\alpha^1} P' \wedge \neg P \wedge P' \Rightarrow P;Q \xrightarrow{\alpha^1} Q$

$Sq3$: $Q \xrightarrow{\alpha^1} Q' \wedge P \Rightarrow P;Q \xrightarrow{\alpha^1} Q'$

$SW1$: $P \xrightarrow{\delta^1} P' \wedge \neg P \wedge d > 0 \Rightarrow sw(P,d) \xrightarrow{\delta^1} sw(P', d-1)$

$SW2$: $P \xrightarrow{\alpha^1} P' \wedge \alpha \neq \delta \wedge d > 0 \Rightarrow sw(P,d) \xrightarrow{\delta^1} P'$

FW: $P \xrightarrow{\alpha^1} P' \wedge \neg P \wedge d > 0 \Rightarrow fw(P,d) \xrightarrow{\alpha^1} fw(P', d-1)$

$E1$: $P \xrightarrow{\alpha^1} P' \wedge d > 0 \Rightarrow e(P,d) \xrightarrow{\alpha^1} e(P', d-1)$

$E2$: $d > 0 \Rightarrow e(P,d) \xrightarrow{\delta^1} e(P, d-1)$

I: $P \Rightarrow P \xrightarrow{\delta^1} P$

SP: $P \xrightarrow{\alpha^1} P' \wedge Q \xrightarrow{\bar{\alpha}^1} Q' \wedge d > 0 \Rightarrow P\langle a^d \rangle Q \xrightarrow{\alpha^1} P'\langle a^{d-1} \rangle Q'$

SI: $P \xrightarrow{\delta^1} P' \wedge Q \xrightarrow{\delta^1} Q' \Rightarrow P\langle a^d \rangle Q \xrightarrow{\delta^1} P'\langle a^d \rangle Q'$

(그림 4) CCS-R 프로세스의 전이 규칙
(Fig. 4) Transiton Rules for CCS-R process

2.3 CCS-R의 연산의미

전이관계 →는 프로세스의 전이를 정의한다. $P \xrightarrow{\alpha} Q$는 P 형태의 프로세스가 라는 동작을 취하면 Q 형태의 프로세스로 변환됨을 나타낸다. 프로세스의 연산의미는 다음과 같은 형태이다.

전제 ∧ 조건 ⇒ 결론

프로세스가 전제와 조건을 동시에 만족하면 프로세스는 결론으로 전이된다. 프로세스의 연산의미는 (그림 3)과 같이 프로세스의 종료를 나타내는 종료술어와 프로세스의 상태전이를 나타내는 (그림 4)의 전이 규칙으로 정의된다. √ 는 성공적으로 수행된 프로세스에 대해서 참이 되는 술어이다.

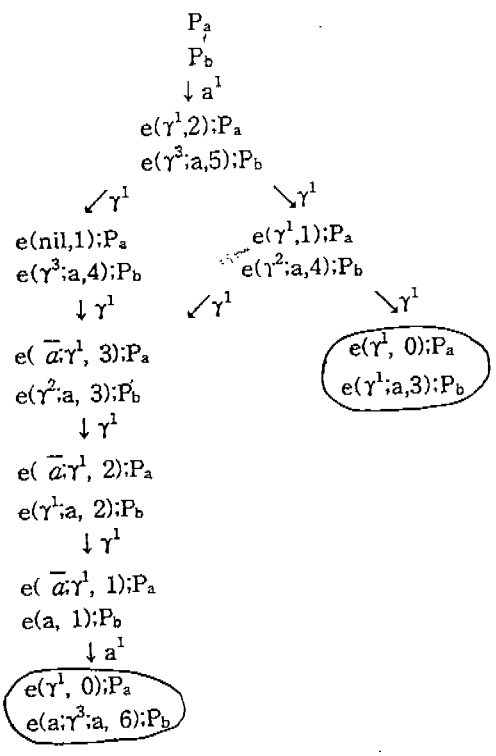
3. 스케쥴가능성 분석

본 논문에서는 주기 프로세스가 순차수행한다고 가정하여 프로세스가 하나의 처리요소에서 동작할 때 마감시간을 만족하는 가를 판단한다. (그림 5)는 Fredette의 방법에 따라 두 프로세스 P_a, P_b의 스케쥴가능성 분석을 위한 상태공간이다. 아래의 P_a는 3 단위시간마다 채널 a에 메시지를 보내고, 1 단위시간동안 수행한 후 다시 P_a를 수행하는 것이며, P_b는 6 단위시간마다 메시지를 채널 b에서 받고 3단위시간동안 수행하고 채널 b에 메시지를 받은 다음에 다시 P_b를 수행한다.

$P_a ::= e(\bar{a};\ \gamma^1,\ 3);\ P_a$
$P_b ::= e(a;\ \gamma^3;\ a,\ 6);\ P_b$
$d(a) = d(\bar{a}) = 1$

이들 프로세스는 어떤 경우에도 예외상태에 도착하므로 단일 처리기 시스템에서는 마감시간을 지키지 못한다.

(그림 5)에서 프로세스가 마감시간을 지키지 못함을 판단하는 것은 마감시간이 0인 상태에 도달하는 가를 판단하는 것이다. 이를 위해서는 발생가능한 모든 상태를 탐색하여 마감시간이 0인



(그림 5) 스케줄불가능한 프로세스의 예
(Fig. 5) Example of unschedulable processes

상태에 도달하면 스케줄불가능하다고 판단한다. 이러한 도달가능성분석은 프로세스의 수에 따라 지수시간을 갖는 알고리즘으로 수행된다. 따라서 프로세스가 마감시간을 지키지 못함을 효율적으로 판단하기 위하여 탐색할 상태의 수를 줄이거나 보다 빠른 단위시간내에 판단하는 방법이 필요하다.

하지만 (그림 5)에서 3 단위시간이 지난 후의 상태를 보면, P_a가 마감시간을 지키려면 2 단위시간전에 동기화 동작 $\bar{a}$를 수행하여야 한다. 하지만 P_b가 동기화 동작 a를 수행하려면, 최소한 3 단위시간이 필요하다. 따라서 P_a는 마감시간을 지키지 못한다. 이와 같은 사실을 이용하면 동기화 동작을 수행하는 프로세스에 대하여 동기화를 수행하기 위하여 지나야 하는 시간, 동기화 시간, 동기화를 수행한 후 나머지 동작을 수행하는 시간의 합이 한 프로세스의 마감시간보다 길면 그 프로세스는 마감시간을 지키지 못함을 미리 판단할 수 있다.

또 비율 단조 스케줄링 방법과 같은 정적 스케줄링대신 마감시간을 만족할 확률이 가장 높은 최소 슬랙(slack) 우선 방법과 같은 동적 스케줄링 방법을 사용하면, 프로세스가 수행하는 순서를 나타내기 위하여 가능한 모든 프로세스의 상태를 생성하지 않아도, 프로세스가 마감시간을 지키도록 하는 프로세스의 수행순서를 얻을 수 있다.

3.1 스케줄가능성 분석 알고리즘

본 논문에서는 프로세스의 주기, 계산시간, 마감시간, 동기화 동작의 수행시간이 알려진 경우 계산시간이나 동기화 동작을 위하여 대기하는 시간으로 인하여 마감시간을 지키지 못하는 경우를 판단하는 방법을 제안한다. 이 방법은 같은 처리기에 할당된 프로세스들의 동기화 동작을 포함한 계산시간이 마감시간을 초과하는 가를 계산하여 프로세스의 모든 상태를 탐색하지 않고 Fredette의 방법보다 더 이른 단위시간내에 프로세스의 스케줄가능성 분석을 수행한다.

이와 같은 방법으로 프로세스의 스케줄가능성 분석을 수행하기 위해서는 먼저 한 채널을 통하여 동기화하는 프로세스의 쌍을 파악하여야 한다. CCS-R로 표현된 실시간 프로세스의 집합에서 한 채널을 통하여 동기화 동작을 수행하는 프로세스의 쌍을 추출하는 알고리즘은 [알고리즘 1]과 같다. 이 알고리즘에서 e, fw, sw와 같은 프로세스 구성자는 고려하지 않는다.

[알고리즘 1] SPT의 구성 알고리즘

$\prod$: 프로세스의 집합
n : $|\prod|$
pi : $\prod$의 원소, $1 \le i \le n$
d : 기간함수

```
1 for all p_i ∈ do
2 α ← p_i의 첫 동작; st_i = 0;
3 while α ≠ nil do
4   case α do
```

```
5    γ: st_i = st_i + d(γ);
6    a: st_i = st_i + d(a);
7       if(a, p_i, d(a), st)가 SPT에 존재하면,
8         then(a, p_i, d(a), st)에 st_i를 추가한다
9         else SPT에 (a, p_i, d(a), st_i)를 등
          록한다 endif;
10      st_i = 0;
11   ā: st_i = st_i + d(ā);
12      if(ā), p_i, d(a), st)가 SPT에 존재하면,
13        then(ā p_i, d(a), st)에 st_i를 추가한다
14        else SPT에 (ā, p_i, d(ā), st_i)를 등
          록한다 endif;
15      st_i = 0;
16   γ1+γ2: st_i = st_i + max(d(γ1), d(γ2));
17   a + γ: st_i = st_i + max(d(a), d(γ));
18        if(a, p_i, d(a), st)가 SPT에 존재하면,
19           then(a, p_i, d(a), st)에 st_i를 추
             가한다
20           else SPT에 (a, p_i, d(ā), st_i)를
             등록한다 endif;
21           st_i = 0;
22   a1+a2: if(a1, p_i, d(a), st)가 SPT에 존재
     하면,
22           then(a1, p_i, d(a), st)에 st_i+d(a
             1)를 추가한다
23           else SPT에 (a1, p_i, d(ā) st_i+d
             (a1))를 등록한다 endif;
24        if(a2, p_i, d(a), st)가 SPT에 존재하면,
25           then(a2, p_i, d(a), st)에 st_i+d(a
             2)를 추가한다
26           else SPT에 (a2, p_i, d(ā), st_i+
             d(a2))를 등록한다 endif;
27        st_i = 0;
28   endcase;
29   α ← p_i의 다음 동작;
30 endwhile
```

이 알고리즘 1은 동기화 동작 a 또는 ā를 포함하는 프로세스 p_i, p_j가 시작한 후 동기화가 끝날 때까지의 최소시간 st(a) 또는 st(ā) 를 구하여 〈표 1〉과 같은 SPT(Synchronizing Process Table)를 구성한다. 동기화 시간 d(a) = d(ā)는 주어진다. 한 프로세스내에서 같은 동기화 동작을 여러 번 하는 것을 표현하기 위하여 각 동기화 동작의 st(a)/st(ā)를 환형리스트로 연결하여 저장한다.

〈표 1〉 SPT의 구조
〈Table 1〉 Synchronizing Process Table

동기화동작	프로세스이름	동기화시간	동기화가 끝날 때까지의 시간
a/ā	p_i	d(a)/d(ā)	st(a)/st(ā)

[알고리즘 2] 스케줄가능성 분석

입력 ∏: 마감시간과 수행시간을 갖는 프로세스의 집합

출력 unschable: 프로세스가 마감시간을 어길 때 참인 부울 변수

n : |∏|

p_i : ∏의 원소, $1 \le i \le n$

p'_i : $pi \rightarrow^* p'_i$, $1 \le i \le n$

missed: 마감시간을 어길 때 참인 부울 변수

π : ∏의 부분집합으로 수행할 동작을 포함하는 프로세스의 집합 $\{p_{i1}, \cdots, p_{ik} \mid 1 \le k \le l \le n\}$

A_i : p_i의 동기화가 끝나는데 필요한 최소시간

B_i : pi의 동기화가 끝난 후 남은 동작의 최소수행시간

C_i : p_i의 마감시간

ST_i : 프로세스 p_i의 동기화까지의 시간,
st_i: 프로세스 p'_i의 동기화까지의 시간

CT_i : 프로세스 p_i의 계산시간,
ct_i: 프로세스 p'_i의 계산시간

DL_i : 프로세스 p_i의 마감시간,
dl_i: 프로세스 p'_i의 마감시간

s_i : 프로세스 p'_i의 슬랙($dl_i - ct_i$)

interval: 프로세스 주기의 최소공배수

```
1 for all p_i ∈ do
2   st_i = ST_i; ct_i = CT_i; dl_i = DL_i; B_i
    = CT_i − ST_i endfor;
3 loop
4   unschable = false;
5   for SPT의 각 entry do
6       p_i는 a를 포함하는 프로세스;
        missed = true; st_i = ST_i;
7       if st_i = 0 then A_i = max(ct_i, dl_i) +
        st_i; C_i = DL_i + dl_i
8                 else A_i = st_i; C_i = dl_i endif;
9       s_i = dl_i − ct_i;
10      if si < 0 then unschable = true;
        return unschable endif;
11      for ā_i를 갖는 SPT의 entry do
12          p_i는 ā를 포함하는 프로세스;
```

```
13          if st_i = 0 then A_i = max(ct, dl_i) +
            st_i; C_i = DL_i + dl_i; st_i = ST_i;
14                  else A_i = st_i; C_i =
                    dl_i endif;
15          s_i = dl_i − ct_i;
16          if s_i < 0 then unschable =
            true ; return unschable endif;
17          missed = missed and max(A_m, A
            _j) > min(C_i−B_m, C_j−B_j);
18          if not missed then break;
19        endfor
20      unschable = unschable and missed;
21      if !unschable then return unschable;
22  endfor;
23  π = {p_k, ···, p_l | 1 ≤ k ≤ l ≤ n}를 선택
    한다;
24  for all p_i ∈ π do ct_i = ct_i − 1, st_i = st_i −
    1 endfor;
25  for all p_i ∈ Π do dl_i = dl_i − 1 endfor;
26  interval = interval − 1;
27 until interval = 0;
28 return unschable;
```

이 알고리즘 2는 Fredette의 방법에 따라 마감시간이 0인 프로세스가 존재하는가를 찾는 방법을 개선한 것으로 두가지 경우에 프로세스가 마감시간을 어긴다고 판단한다. 첫째, 10과 16번 문장에서 프로세스 p_i의 슬랙 s_i가 0보다 작으면 p_i가 마감시간을 지키지 못한다고 판단한다. 둘째, 동기화가 끝나는데 필요한 최소시간 A가 마감시간 C와 동기화가 끝난 후 남은 동작의 최소 수행시간 B의 차이보다 클 때 동기화에 참가하는 프로세스가 마감시간을 어긴다고 판단한다.

그 결과는 17번 문장에서 missed라는 변수에 저장된다. 그리고 여러 프로세스들이 여러개의 채널에 대하여 동작하는 경우는 한 채널에 대한 동기화 동작으로 인하여 마감시간을 어기게 되면, 그 프로세스는 마감시간을 지키지 못한다고 판단한다. 이것은 20번 문장의 unschable이라는 변수에 결과가 저장된다.

23번 문장에서 수행할 프로세스는 다음과 같은 기준으로 선택된다. 이것은 스케줄가능한 프로세스의 수행순서를 결정하는 임계경로를 구성하는 원칙이 된다.

① 동기화 동작
② 동기화 동작까지의 최소간격
③ 최소 slack
④ 최선 마감시간
⑤ 최소주기

3.2 알고리즘의 운영예

스케줄링가능성 분석 알고리즘에 따라 두 프로세스가 하나의 처리기에 할당되어 수행할 경우의 마감시간을 지키는가의 여부에 대한 예를 적용하여 결과를 분석한다.

[예 1] 다음 두 프로세스 P_a와 P_b가 하나의 처리기에 할당되어 수행할 때 마감시간을 지키는 가를 판단하자.

$$P_a :: = e(\bar{a}; \gamma', 3); P_a$$
$$P_b :: = e(a; \gamma'; a, 6); P_b$$
$$d(a) = d(\bar{a}) = 1$$

Fredette의 방법[1, 2, 14]을 이용하면, 9개의 상태가 생성되고, 이 상태를 모두 탐색하여야 프로세스 P_a가 마감시간을 어긴다는 사실을 알 수 있다. 하지만, 본 논문의 방법을 이용하면 3 단위시간에 A2 > C1-B1이 되어 P_b가 마감시간을 어김을 알 수 있다. SPT는 〈표 2〉, 분석 결과는 〈표 3〉이고, 프로세스의 상태는 (그림 6)과 같다.

〈표 2〉 Pa와 Pb의 SPT
〈Table 2〉 SPT of P_a and P_b

동기화동작	프로세스 이름	d(a)	st(a)
$\bar{a}$	P_a	1	1
a	P_b	1	1, 4

〈표 3〉 분석과정
〈Table 3〉 Analysis Phase

P_a				P_b			
A_a	B_a	C_i	s_a	A_b	B_b	C_b	s_b
1	1	3	1	1	4	6	1
3	1	5	1	4	0	5	1
2	1	4	1	4	0	4	0

P_a
P_b
$\downarrow a^1$
$e(\gamma^1,2);P_a$
$e(\gamma^3;a,5);P_b$
$\downarrow \gamma^1$
$e(nil,1);P_a$
$e(\gamma^3;a,4);P_b$

(그림 6) 프로세스 상태 공간
(Fig. 6) Process State Space

4. 실험 및 평가

4.1 실험 환경 및 방법

본 장에서는 최소 슬랙 우선, 최선 마감시간 우선, 비율 단조(rate monotonic) 스케줄링 방법을 혼합한 스케줄링 방법을 기반으로 하여 CCS-R로 표현된 실시간 프로세스에 대하여 Fredette의 방법과 본 논문에서 제안한 방법을 IBM PC 상에서 C 언어로 실험하여, 마감시간을 지키지 못함을 결정하는 단위시간을 비교한다. 스케줄가능성 판단 과정은 아래와 같고, 실험 대상 실시간 프로세스는 부록에 제시하였다.

스케줄가능성 판단 과정

[단계 1] 실시간 프로세스를 입력한다.
[단계 2] 동작 리스트를 그성한다.
[단계 3] 마감시간 리스트를 구성한다.
[단계 4] 프로세스 주기의 최소 공배수(LCM)를 구한다.
[단계 5] 단위시간이 최소 공배수만큼 경과할 때까지 다음을 반복한다.
1) SPT를 구성한다.
2) 스케줄가능성을 판단하여 불가능하면, 출력한 후에 스케줄링을 한 다음에 수행하고, 가능하다면, 스케줄링한 후에 수행한다.
3) 동작 리스트를 재구성한다.
4) 마감시간 리스트를 재구성한다.
5) 슬랙이 0보다 작은 프로세스가 존재하는가를 검색하여, 존재하면 출력 후에 단위시간을 1 증가시키고, 존재하지 않으면 곧바로 단위시간을 1 증가시킨다.

4.2 실험 결과 분석

이 절에서는 13개의 실시간 프로세스에 대하여, 실험한 결과를 비교 분석한다. 대상 실시간 프로세스는 부록에 제시하였고, 그 실험 결과는 〈표 4〉와 같다.

실험 대상 실시간 프로세스 중 N1, N9, N10, N11, N12, N13은 시간 구성자를 포함하지 않는 주기 프로세스이며, 나머지는 sw, fw와 같은 시간 구성자를 포함하는 주기 프로세스이다. 본 논문에서 제안한 방법은 모두 Fredette의 방법을 적용한 경우보다 이른 시간이내에 스케줄이 불가능함을 판단한다.

전체적인 절감률은 51%로 Fredette의 방법에 비하여 약 반 정도의 단위시간만을 소모하여 스케줄이 불가능함을 판단하였다. 또한 절감된 단위시간은 평균적으로 최악의 경우 수행하여야 할 주기의 최소 공배수의 약 1/3인 35%라는 결과를 얻었다.

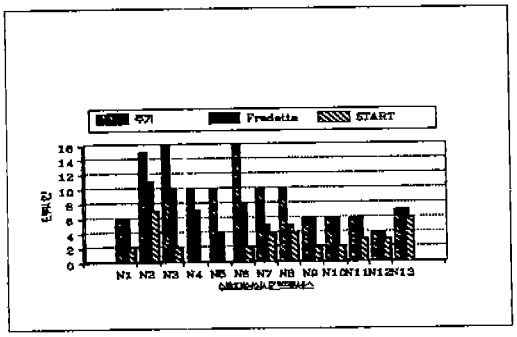
(그림 7)은 부록에 제시한 각 실험 대상 프로세스의 분석결과를 그래프로 표시한 것이다. 주기 항목은 프로세스의 주기의 최소 공배수이며, Fredette는 Fredette의 방법으로 분석하였을 경우

〈표 4〉 실험 결과 비교
〈Table 4〉 Comparision of Experimetal Results

	주기의 최소 공배수	Fredette의 방법	본 논문의 방법	절감률(%)	비고
N1	6	6	2	67	
N2	15	11	7	33	
N3	16	10	2	80	
N4	10	7	0	100	
N5	10	4	0	100	
N6	16	8	2	75	
N7	10	5	4	20	
N8	10	5	4	20	Slack < 0
N9	6	6	2	67	
N10	6	6	2	67	
N11	6	6	3	50	
N12	4	4	3	25	Slack < 0
N13	7	7	6	14	Slack < 0
합계	122	85	41	52	

에 걸리는 단위시간, START는 본 논문에서 제안한 방법으로 분석하였을 경우에 걸리는 단위시간을 나타낸다.

시간 구성자를 포함하지 않는 프로세스의 집합인 A군과 시간 구성자를 포함하는 프로세스의 집합 B군을 별도로 비교한 결과인 〈표 5〉에서 볼 수 있듯이 B 군에 대하여 더 좋은 성능을 보였음으로 시간 구성자를 포함하는 실시간 프로세스에 더 적합할 것으로 판단된다.



(그림 7) 실험결과 비교 그래프
(Fig. 7) Comparision Graph of Experimental Results

〈표 5〉 프로세스 군별 실험 결과
〈Table 5〉 Experimental Results of Process Group

	평균 주기 최소 공배수(a)	Fredette의 방법(b)	1-b/a	본 논문의 방법(c)	1-c/a	절감률 (1-b/c)*100%
A군	5.8	5.8	1	3.3	0.43	43
B군	12.1	6.9	0.57	3.1	0.74	55

5. 결 론

본 논문에서는 실시간 프로그램의 새로운 스케줄가능성 분석 방법을 제안하고, 실험을 통하여 그 결과를 비교 분석한다. 본 논문의 스케줄가능성 분석은 동기화 동작을 포함하는 프로세스가 마감시간을 지키는가를 판단하기 위하여 프로세스 대수의 전이규칙에 따라 생성되는 프로세스의 상태의 동기화를 수행하는데 필요한 최소 수행시간, 프로세스의 마감시간, 그리고 동기화 동작이 끝난 후 수행되어야 하는 동작의 최소 수행시간을 계산한다. 또 프로세스의상태공간을 생성하기 위하여 프로세스들이 마감시간을 지킬 확률이 높은 동적 스케줄링을 사용한다.

따라서 본 논문에서 제안한 스케줄가능성 분석기는 고정 스케줄링 방법을 사용하고, 발생가능한 모든 프로세스의 상태공간을 탐색하는 방법을 사용함으로써 시간, 공간적 복잡도가 높은 Stoyenko, Gerber, Fredette 등의 스케줄가능성 분석기들과는 달리 적은 수의 상태를 생성하고, 이른 단위시간내에 스케줄이 불가능한 프로세스를 판별할 수 있음이 실험 결과를 통하여 밝혀졌다.

Stoyenko, Gerber, Fredette의 방법은 모두가 발생가능한 모든 상태를 생성하여 검색하기 때문에 실험 결과와 같이 Fredette의 방법만을 비교하였지만, Stoyenko나 Gerber의 방법 역시 실험을 통하여 비교해 보면, 본 논문에서 제안한 방법이 Stoyenko, Gerber의 방법보다도 더 이른 단위시간 내에 스케줄이 불가능한 프로세스를 판별할 수 있을 것이다.

향후 연구 과제로는 여러 채널을 통하여 다수의 프로세스와 동기화를 수행하는 프로세스는 음수가 되는 슬랙을 이용하여 스케줄이 불가능함을 판별할 수 있었지만, 다른 채널을 통한 동기화 동작간의 관계를 분석하여 스케줄가능성 분석을 수행하는 방법과 스케줄이 불가능할 경우의 처리 방법에 대한 추가의 연구가 필요하다.

부 록

실험 대상 실시간 프로세스는 다음과 같다.

Pa ::= e(a^1, 3);Pa;
Pb ::= e($\overline{a^2}$; $\overline{a^2}$;γ^3, 6);Pb;

(a) N1

Pa ::= e(γ^1; $\overline{a^1}$;sw(γ^1;b^1, 10), 15);Pa;
Pb ::= e(γ^1;a^1;γ^2; $\overline{b^1}$;γ^1, 15);Pb;
Pc ::= e(γ^1; $\overline{a^1}$;fw(b^1;γ^1, 13), 15);Pc;
Pd ::= e($\overline{b^1}$;γ^3;fw(γ^1;a^1, 13), 15);Pd;

(b) N2

Pa ::=e(d^1;c5;sw($\overline{c^1}$;γ^1, 12), 16);Pa;
Pb ::= e(γ^3;a^1;sw(γ^3; $\overline{b^1}$, 12), 16);Pb;

Pc ::= e(c[1];γ[1];fw(γ[3]; $\overline{a}^1$, 14), 16);Pc;
Pd ::= e(γ[2]; $\overline{b}^1$;fw(γ[1];a[1], 30), 16);Pd;
Pe ::= e(γ[4];a[1], 8);Pe;
Pf ::= e($\overline{a}^1$;γ[1], 16);Pf;

(c) N3

Pa ::= e(sw(γ[3]; $\overline{a}^1$, 3);sw(γ[1];b[1], 6), 10);Pa;
Pb ::= e(sw(γ[1];a[1], 3);γ[4];fw(γ[1]; $\overline{b}^1$, 9), 10);Pb;

(d) N4

Pc ::= e(sw(γ[1]; $\overline{c}^1$, 3);fw(d[1];γ[1], 6), 10);Pc;
Pd ::= e(sw($\overline{d}^1$;γ[1], 3);γ[2];fw(γ[1];c[1], 8), 10);Pd;

(e) N5

Pa ::= e(γ[2]; $\overline{a}^1$;fw(b[1];b[1], 10);γ[2]; $\overline{a}^1$; $\overline{a}^1$;b[1], 16);Pa;
Pb ::= e(a[1]; $\overline{b}^1$;γ[4], 8);Pb;
Pc ::= e(d[1];fw(γ[1]; $\overline{c}^1$, 8);γ[3];d[1]; $\overline{c}^1$;sw(d[1]; $\overline{c}^1$, 12), 16);Pc;
Pd ::= e($\overline{d}^1$;c[1], 8);Pd;

(f) N6

Pc ::= e(fw(a[1];γ[1], 4);γ[1], 10);Pc;
Pa ::= e($\overline{a}^1$;γ[1];sw($\overline{a}^1$;γ[1], 6), 10);Pa;
Pb ::= e(γ[4];a[1];γ[4], 10);Pb;

(g) N7

Pa ::= e($\overline{a}^1$;γ[3]; $\overline{a}^1$;γ[1], 10);Pa;
Pb ::= e(γ[1];a[1];γ[4]; $\overline{b}^1$, 10);Pb;
Pc ::= e(γ[2];a[1];γ[2], 10);Pc;
Pd ::= e(γ[1];b[1];c5, 10);Pc;

(h) N8

Pa ::= e(a[1];γ[1], 3);Pa;
Pb ::= e($\overline{a}^1$;γ[4]; $\overline{a}^1$, 6);Pb;

(i) N9

Pa ::= e($\overline{a}^1$, 3);Pa;
Pb ::= e(a[1];a[1];γ[4], 6);Pb;

(j) N10

Pa ::= e($\overline{a}^1$;γ[1], 3);Pa;
Pb ::= e(a[1];γ[3];a[1], 6);Pb;

(k) N11

Pa ::= e($\overline{a}^1$, 2);Pa;
Pb ::= e(γ[2];a[1], 4);Pb;
Pc ::= e(fw(a[1];γ[1], 3), 4);Pc;

(l) N12

Pa ::= e($\overline{a}^1$; $\overline{b}^1$;γ[2], 7);Pa;
Pb ::= e(a[1];γ[2];c[1], 7);Pb;
Pc ::= e(γ[2];b[1]; $\overline{c}^1$, 7);Pc;

(m) N13

참 고 문 헌

[1] A. N. Fredette and R. Cleaveland, "A Generalized Approach to Real-Time Schedulability Analysis," 10th IEEE Workshop on Real-Time Operating Systems and Software, 1993.

[2] A. N. Fredette and R. Cleaveland, "RTSL: A Language for Real-Time Schedulability Anaysis," Proceedings of the IEEE Real-Time Systems Symposium, pp. 274-283, 1993.

[3] R. Gerber and I. Lee, "A Layered Approach to Automating the Verification of Real-Time Systems," IEEE Transactions on Software Engineering, Vol. 18, No. 9, pp. 768-784, Sep. 1992.

[4] W. A. Halang and A. D. Stoyenko, "Comparative Evaluation of High-Level Real-Time Programming Languages," Real Time Systems, Kluwer Academic Publishers, Vol. 2, pp. 365-382, 1990.

[5] W. A. Halang and A. D. Stoyenko, *Constructing Predictable Real Time Systems*, Kluwer Academic Publishers, 1991.

[6] C.-J. Hou and K. G. Shin, "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems," Proceedings of the IEEE Real-Time Systems Symposium, pp. 146-155, 1992.

[7] R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.

[8] F. Moller and C. Tofts, "A Temporal

Calculus of Communicating Systems," Proceedings of CONCUR '90, LNCS 458, pp. 401-415, 1990.

[9] X. Nicollin and J. Sifakis, "An Overview ans Synthesis on Timed Process Algebras," Proceedings of the REX workshop on Real-Time: Theory and Practice, LNCS 600, 1991.

[10] A. D. Stoyenko, "The Evolution and State-of-the-Art of Real-Time Languages," The Journal of Systems and Software, Elsevier Science Publishers, pp. 61-84, April 1992.

[11] A. D. Stoyenko, V. C. Hamacher, and R. C. Holt, "Analyzing Hard-Real-Time Programs For Guaranteed Schedulability," IEEE Transactions on Software Engineering, Vol. 17, No. 8, pp. 737-750, Aug. 1991.

[12] W. Tarng and T.-H. Lin, "Fault-Tolerant Task Assignment in Distributed Real-Time Computing Systems," Readings in Real-Time Systems, IEEE, pp. 98-110, 1993.

[13] W. Yi, "CCS+Time = an Interleaving Model for Real Time Systems," Proceedings of ICALP '91, 1991.

[14] A. N. Fredette, "A Generalized Approach to the Analysis of Real-Time Computer System," Ph. D. Thesis, North Carolina State University, 1993.

박 홍 복

1982년 경북대학교 전자공학과(전자계산 전공) 졸업(공학사)
1984년 경북대학교 대학원 전자공학과(전산공학) 전공(공학 석사)
1992년 인하대학교 대학원 전자계산공학과 박사과정 수료
1984년~현재 동명전문대학 전자계산과 부교수
관심분야: 실시간 프로그래밍 언어, 인공지능, 멀티미디어/전자출판.

유 원 희

1975년 서울대학교 공과대학 응용수학과 졸업(이학사)
1978년 서울대학교 대학원 계산학 전공(이학석사)
1985년 서울대학교 대학원 계산학 전공(이학박사)
1979년~현재 인하대학교 공과대학 전자계산공학과 교수
관심분야: 프로그래밍 언어(실시간 프로그래밍 언어, 함수형 언어).