분산 실시간 프로세스의 스케줄가능성 분석 및 구현

박 흥 복*, 김 춘 배**

A Schedulability Analysis and Implementation of Distributed Real-Time Processes

Hung-Bog Park*, Choon-Bae Kim**

요 약

실시간 스케줄가능성 분석을 위한 여러 가지 방법들이 제안되었지만, 이들은 프로세스의 모든 상태공간을 추적하거나 고정 우선 순위 스케줄링 방법을 사용하기 때문에 시간 및 공간에 대한 복잡성의 증가를 야기한다. 따라서 상태공간의 수를 줄임으로써 보다 빠른 시간에 스케줄가능성을 판단하는 방법이 필요하다. 본 논문에서는 프로세스 대수의 전이규칙을 이용하여 번역시간에 결정될 수 있는 프로세스의 최소 수행시간, 주기, 마감시간, 동기화 시간을 고려하여 실시간 프로세스가 마감시간을 지키는가를 판단하는 개선된 알고리즘을 제안 및 구현한다. 구현 결과를 통하여 모든 상태 공간을 검색하지 않고도 스케줄가능성을 판단하는 효과적인 성능을 보였다.

Abstract

Several approaches to anlayzing real-time schedulability have been presented, but since these used a fixed priority scheduling scheme 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.

This paper proposes and implements an advanced schedulability analysis algorithm to determine that is satisfied a given deadlines for real-time processes. These use a mnimum execution time of process, periodic,

* 정회원 : 부경대학교 컴퓨터·멀티미디어 공학부 부교수

** 준회원 : 부경대학교 대학원 전자계산학과

※ 본 연구는 1998년도 정보통신부의 대학기초연구지원사업에 의하여 수행되었음.

접수일자 : 1998년 12월 29일

deadline, and a synchronization time of processes to detect schedulability at earlier time and dynamic scheduling scheme to reduce state space using the transiton rules of process algebra. From a result of implementation, we demonstrated the effective performance to determine schedulability analysis.

Ⅰ. 서 론

분산 실시간 시스템(distributed real-time system)은 높은 신뢰성, 처리능력의 증가, 자원의 공유 등 여러 가지 장점 때문에 근래에 많은 관심을 받고 있다. 분산 실시간 시스템은 통신망(communication network)으로 연결된 노드라는 처리요소(processing element)로 구성된다. 이러한 시스템에서는 계산의 속도를 증가시키고, 효율적인 수행을 위하여 프로그램이 프로세스라는 단위로 나뉘어져 병렬로 수행된다.

한 노드에서 프로세스는 공유 기억장치(shared memory)를 통하여 통신하고, 다른 노드의 프로세스와는 통신망을 통하여 메세지를 교환한다[5].

시스템의 운행과 외부 실체와의 상호관계가 시간과 밀접한 관련이 있는 실시간 시스템에서 실시간 프로세스가 마감시간을 어기는 것은 매우 큰 문제이므로 번역시간에 시스템의 동작을 예측할 수 있는 능력(predictibility)과 프로세스가 지정된 마감시간을 만족하는 신뢰성을 확인하는 것이 중요한 문제이다.

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

Liu와 Layland는 rate monotonic 스케줄링시 독립적인 프로세스의 계산시간, 주기, 마감시간이 주어졌을 때 프로세스들이 마감시간을 지키는 가를 판단하는 방법을 제안하였으나 이 방법은 두 프로세스간의 동기화를 처리하거나 둘 이상의 시간제약을 갖는 프로세스를 처리할 수 없는 제한이 있다[3,5].

Stoyenko 등[10,11]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 시간범위(time bound)를 갖는다.

CCSR[4]은 동작에 우선순위를 할당하여 고정우선순위 스케줄링(fixed priority scheduling)을 모델링하는데 사용된다. CCSR은 도달가능성 분석(reachability analysis)을 통하여 프로그램을 검증한다. 도달가능성 분석에서 시간제약을 위반하는 상태에 도달하는 가를 결정하기 위하여 프로세스가 정의하는 전이 시스템(transition system)을 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 상태공간(state space)을 탐색하는 스케줄가능성 분석을 제안하였다[1,2,3]. 이 방법은 interleaving 모델에 기초한 단일처리기(uniprocessor) 시스템만을 고려하여 분산 시스템에는 적합하지 않으며, 프로세스의 수에 대하여 지수승으로 상태(state)의 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 모든 상태공간 검색하여야 스케줄이 불가능한 상태를 판단할 수 있는 Fredette의 방법[1,2,3]과 비슷하게 프로세스 대수의 전이규칙을 이용하지만 프로세스의 모든 상태공간을 검색하지 않고도 번역시간에 결정될 수 있는 프로세스들의 수행순서와 프로세스들의 최소 수행시간, 주기, 마감시간, 동기화 시간을 고려하여 남은 계산시간과 마감시간의 차이를 계산하여 실시간 프로세스가 마

감시간을 지키는가를 분석하는 알고리즘을 개선하여 구현한 후, 마감시간을 지키지 못하는 프로세스를 판별하고 기존의 방법[3]과 비교·분석한다

Ⅱ. 실시간 명세언어

실시간 명세언어(real time specification language)는 원시 프로그램의 모든 동작을 표현하지 않고 관심부분만을 요약하여 그 특성을 분석할 수 있어 프로그램의 정적분석(static analysis)에 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[6]에서 볼 수 있다. 프로세스 term P는 그림 1과 같은 BNF 문법으로 주어진다.

P ::= nil | X::=P | a.P | P+Q | P ‖ Q

그림 1. CCS의 구문
Fig. 1 Syntax of CCS

대문자는 프로세스, 소문자는 프로세스를 구성하는 동작(action)이다. 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 병렬조합(parallel compositon)을 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[7]. 프로세스의 동작을 나타내는 프로세스의 전이 관계는 [정의 1]과 같다.

[정의 1] 전이 시스템은 4-튜플 <Q, A, →, q_0>로 정의된다.
- Q는 상태의 집합이다.
- A는 동작의 집합이다.
- → : Q × A × Q는 상태와 동작으로 부터 새로운 상태를 만드는 전이 관계이다.
- q_0는 초기상태이다.

2.2 명세언어 CCS-R

본 논문에서 사용하는 실시간 명세언어 CCS-R은 RTSL[1, 2, 3]과 비슷하지만 RTSL에서 예외처리기를 나타내는 시간구성자 **h**와 재귀호출을 제거하여 단순화시킨 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 만족하는가를 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 a^d의 세가지 종류가 있다. d는 동작을 수행하는데 필요한 최대 수행시간이다. 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 2와 같은 BNF 문법으로 정의된다.

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

그림 2. CCS-R 프로세스
Fig. 2. Processes of CCS-R

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

실시간 프로세스가 마감시간을 지키는가를 번역시간에 판단하기 위해서는 프로세스의 동작뿐만 아니라 프로세스의 수행시간을 번역시간에 계산할 수 있거나 프로그램상에 명시되어야 한다. 프로세스의 수행시간을 나타내는 기간함수(duration function)를 [정의 2]와 같이 정의한다.

[정의 2] 기간함수 d

- **D**는 기간정의역(duration domain)이다. 기간정의역 의 원소를 d라 하면 ∀d, d > 0이다.
- *A*는 모든 동작의 집합 { a^d, γ^d, δ^d | d ∈ **D** }이다. a ∈ *A*에 대하여 $\bar{a}$는 보동작으로 *A*의 원소 이다.
- 기간함수 $d : A \rightarrow$ **D**는 모든 동작의 기간이다. ∀ a∈*A*, $d(a)=d(\bar{a})$이다.

2.3 CCS-R의 연산의미

전이관계 →는 프로세스의 전이를 정의한다. $P \xrightarrow{\alpha} Q$는 P 형태의 프로세스가 α라는 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 3과 같이 프로세스의 종료를 나타내는 종료술어와 프로세스의 상태전이를 나타내는 그림 4의 전이규칙으로 정의된다. ↓는 성공적으로 수행된 프로세스에 대해서 참(true)인 술어이다.

$T1$: $nil\!\downarrow$ $T2$: $a^0\!\downarrow$ $T3$: $P\!\downarrow \wedge Q\!\downarrow \Rightarrow (P+Q)\!\downarrow$

$T4$: $P\!\downarrow \wedge Q\!\downarrow \Rightarrow (P;Q)\!\downarrow$ $T5$: $P\!\downarrow \Rightarrow sw(P,d)\!\downarrow$

$T6$: $P\!\downarrow \Rightarrow fw(P,d)\!\downarrow$ $T7$: $P\!\downarrow \Rightarrow e(P,0)\!\downarrow$

그림 3. CCS-R의 종료술어

Fig. 3. Termination predicates of CCS-R

$A1$: $a^d \xrightarrow{a^1} a^{d-1}$ $A2$: $a \xrightarrow{a^1} a^{d(a)-1}$ $A3$: $d>0 \Rightarrow a^d \xrightarrow{\delta^1} a^d$

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

(a) 동작(Action)의 전이 규칙

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

(b) 선택(choice)의 전이 규칙

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

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

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

(c) 순차조합의 전이 규칙

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

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

(d) 시간구성자 sw의 전이 규칙

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

(e) 시간구성자 fw의 전이 규칙

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

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

(f) 시간구성자 e의 전이 규칙

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

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

(g) 지연동작의 전이 규칙

SE: $P \xrightarrow{a^1} P' \wedge Q \xrightarrow{\bar{a}^1} Q' \wedge d=1 \Rightarrow P\langle a^d\rangle Q \xrightarrow{a^1} P' \wedge P\langle a^{d'}\rangle Q \xrightarrow{\bar{a}^1} Q'$

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

(h) 동기화 동작의 전이 규칙

그림 4. CCS-R 프로세스의 전이규칙

Fig. 4. Transition rule of CCS-R process

그림 4의 동작의 전이 규칙(a)은 d>0이고 a^d가 한 시간 단위동안 수행되면, a^{d-1}로 전이됨을 나타낸다. (h)에서 $P<a^d>Q$는 프로세스 P와 Q가 수행시간이 d인 동기화 동작 a를 수행하고 있음을 나타낸다. 규칙 SP(Synchronizing Pair)는 프로세스 P와 Q가 동기화 동작 a를 수행할 때의 전이규칙이다. SE (Synchronization End)는 수행시간이 1 남은 동기화 동작을 수행하는 경우의 전이규칙으로 동기화 동작이 끝나면 동기화 동작에 참여한 프로세스는 각각 수행을 진행한다.

Ⅲ. 스케줄가능성 분석

본 논문에서는 주기 프로세스가 순차수행한다고 가정하여 프로세스가 하나의 처리요소에서 동작할 때 마감시간을 만족하는 가를 판단한다. 그림 5는 Fredette의 방법에 따라 두 프로세스 P_a, P_b의 스케줄가능성 분석을 위한 상태공간이다. 아래의 P_a는 3 단위시간마다 채널 a에 메시지를 보내고, 1 단위시간 동안 수행한 후 다시 P_a를 수행하는 것이며, P_b는 6 단위시간마다 메시지를 채널 b에서 받고 3단위시간 동안 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 5에서 프로세스가 마감시간을 지키지 못함을 판단하는 것은 마감시간이 0인 상태에 도달하는 가를 판단하는 것이다. 이를 위해서는 발생가능한 모든 상태를 탐색하여 마감시간이 0인 상태에 도달하면 스케줄불가능하다고 판단한다. 이러한 도달가능성 분석은 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 5에서 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.

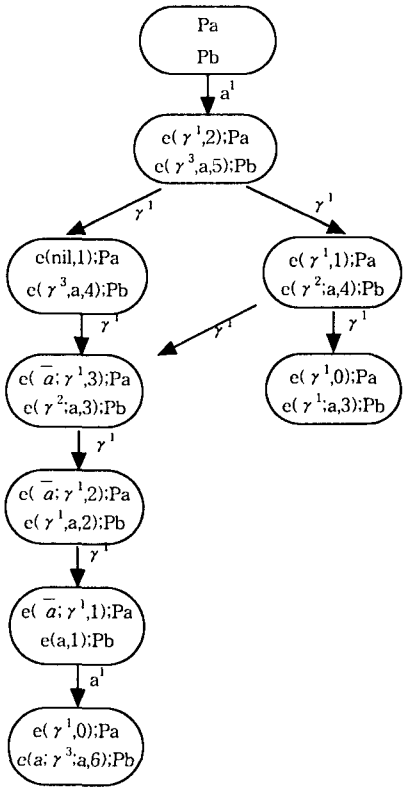


그림 5. 스케줄불가능한 프로세스의 예
Fig. 5. Example of unschedulable processes

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 하나의 처리기에 할당된 프로세스들의 동기화 동작을 포함한 계산시간이 마감시간을 초과하는 가를 계산하여 프로세스의 모든 상태를 탐색하지 않고 마감시간이 0이 되기 전에 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 추출하여 SPT(Synchronizing Process Table)를 구성하는 방법은 [알고리즘 1]과 같다. 이 알고리즘에서 **e, fw, sw**와 같은 프로세스 구성자는 고려하지 않는다.

[알고리즘 1] SPT의 구성 알고리즘

```
Π : 프로세스의 집합
n : |Π|
pi : Π의 원소, 1 ≤ i ≤ n
d : 기간함수

1 for all pi ∈ Π do
2  α ← pi의 첫 동작; sti = 0;
3  while α ≠ nil do
4   case α do
5     γ : sti = sti + d(γ);
6     a : sti = sti + d(a);
7     if (a, pi, d(a), st)가 SPT에 존재하면,
8       then (a, pi, d(a), st)에 sti를 추가한다
9       else SPT에 (a, pi, d(a), sti)를 등록한다 endif;
10      sti = 0;
11      ā : sti = sti + d(ā);
12    if (ā, pi, d(a), st)가 SPT에 존재하면,
13      then (ā, pi, d(a), st)에 sti를 추가한다
14      else SPT에 (ā, pi, d(ā), sti)를 등록한다 endif;
15     sti = 0;
16    γ1+γ2 : sti = sti + max(d(γ1), d(γ2));
17    a + γ : sti = sti + max(d(a), d(γ));
18     if (a, pi, d(a), st)가 SPT에 존재하면,
19      then (a, pi, d(a), st)에 sti를 추가한다
20      else SPT에 (a, pi, d(ā), sti)를 등록한다 endif;
21     sti = 0;
22    a1+a2 : if (a1, pi, d(a), st)가 SPT에 존재하면,
22              then (a1, pi, d(a), st)에 sti+d(a1)를 추가한다
23              else SPT에 (a1, pi, d(ā), sti+d(a1))를 등록한다
                endif;
24            if (a2, pi, d(a), st)가 SPT에 존재하면,
25              then (a2, pi, d(a), st)에 sti+d(a2)를 추가한다
26              else SPT에 (a2, pi, d(ā), sti+d(a2))를 등록한다
                endif;
27            sti = 0;
28   endcase;
29   α ← pi의 다음 동작;
30 endwhile
```

[알고리즘 1]은 동기화 동작 a 또는 $\bar{a}$를 포함하는 프로세스 p_i, p_i가 시작한 후 동기화가 끝날 때까지의 최소시간 st(a) 또는 st($\bar{a}$)를 구하여 표 1과 같은 SPT를 구성한다. 동기화 시간 d(a) = $d(\bar{a})$는 주어진다. 한 프로세스내에서 같은 동기화 동작을 여러번 하는 것을 표현하기 위하여 각 동기화 동작의 st(a)/st($\bar{a}$)를 환형 리스트(circular list)로 연결하여 저장한다.

표 1. SPT의 구조

Table 1. Structure of SPT

동기동작	프로세스이름	동기시간	동기화가 끝날 때 까지 시간
a / $\bar{a}$	pi	d(a)/d($\bar{a}$)	st(a)/st($\bar{a}$)

[알고리즘 2] 스케줄가능성 분석

```
입력: 마감시간과 수행 시간을 갖는 프로세스의 집합 Π
출력: 프로세스가 마감시간을 어길 때, 참인 부울 변수 unschable
n : |Π|
pi : Π의 원소, 1 ≤ i ≤ n
p'i : pi ⇒ p'i, 1 ≤ i ≤ n,
     p'i는 pi에 0번 이상의 전이 규칙을 적용한 후 생성되는 프로
     세스의 상태
missed : 마감시간을 어길 때, 참인 부울 변수
missed_p : Π의 부분 집합으로 마감시간을 지키지 못하는 프로세
     스의 집합
π : Π의 부분 집합으로, 다음 단계에서 수행할 동작을 포함하는 프
     로세스의 집합 π { pk, ..., pl | 1 ≤ k, l ≤ n}
Ai : pi의 동기화가 끝나는데 필요한 최소시간
Bi : pi의 동기화가 끝난 후 남은 동작의 최소 수행시간
Ci : pi의 마감시간
STi : 프로세스 pi의 동기화까지의 시간, sti : 프로세스 p'i의 동기
    화까지의 시간
CTi : 프로세스 pi의 계산시간, cti : 프로세스 p'i의 계산시간
DLi : 프로세스 pi의 마감시간, dli : 프로세스 p'i의 마감시간
si : 프로세스 p'i의 slack(dli - cti)
interval : 프로세스 주기의 최소 공배수
m : SPT의 원소수
 1 for i = 1 to n do
 2  sti = STi; cti = CTi; dli = DLi endfor;
 3 loop
 4  unschable = false;
 5  for j = 1 to m do
 6   a = SPT[j].act_name; pi = SPT[j].proc_name;
 7   if sti = 0
 8     then if (sti가 SPT[j]의 마지막 st 값)
 9            then STi = SPT[j]의 첫 st 값;
10            sti = max(cti, dli) + STi; cti = max(cti, dli) + CTi;
11            Ai = sti; Ci = DLi + dli;
12          else STi = SPT[j]의 다음 st 값
13            sti = STi; Ai = sti; Ci = dli;
14       endif;
15   else Ai = sti; Ci = dli endif;
16   si = dli - cti; Bi = cti - sti;
17   if si < 0 then missed = true; missed_p = {pi} ∪missed_p;
18   break endif;
19   for k = j+1 to m do  /*다른 프로세스의 동기화 보동작과
                            비교 */
20     missed = true;
21     if SPT[k].act_name = ā then
22       pj = SPT[k].proc_name;
23       if stj = 0
```

```
24            then if (st_j가 SPT[k]의 마지막 st 값)
25                 then ST_i = SPT[k]의 첫 st 값;
26                      st_j = max(ct_j, dl_j) + ST_j;
27                      ct_j = max(ct_j, dl_j) + CT_j;
28                      A_j = st_j; C_j = DL_j + dl_j;
29                 else ST_j = SPT[k]의 다음 st 값
30                      st_j = ST_j; A_j = st_j; C_j = dl_j;
31               endif;
32       else A_j = st_j; C_j = dl_j endif;
33       s_j = dl_j - ct_j; B_j = ct_j - st_j;
34       if s_j < 0 then missed = true; missed_p
                          = {p_j}∪missed_p;
35       break endif;
36       missed = missed and max(A_i, A_j) > min(C_i-B_i, C_j-B_j);
37       if missed then break endif;
38     endif;
39   endfor;
40     unschable = unschable or missed; /* 전체 프로세스의 스케
                                          줄 가능성 판단 */
41   if unschable then return unschable;
42   endfor;
43   π를 선택;
44   for all p_i ∈ π do ct_i = ct_i - 1, st_i = st_i - 1 endfor;
45   for i = 1 to m do dl_i = dl_i - 1 endfor;
46   interval = interval - 1;
47 until interval = 0;
48 return unschable;
```

이 스케줄가능성 분석 알고리즘은 두 가지 경우에 프로세스가 마감시간을 어긴다고 판단한다. 첫째, [알고리즘 2]의 17과 34번 문장에서 프로세스 p_i의 슬랙(slack) s_i가 0보다 작으면, 즉 남은 계산시간이 마감시간 보다 클 때, p_i가 마감시간을 지키지 못한다고 판단한다. 둘째, 동기화가 끝나는데 필요한 최소 시간 A가 마감시간 C와 동기화가 끝난 후 남은 동작의 최소 수행 시간 B의 차이보다 클 때, 동기화에 참가하는 프로세스가 마감시간을 어긴다고 판단할 수 있다.

[알고리즘 2]는 여러 프로세스들이 하나의 채널에 대하여 동작할 때 그 채널에 대하여 동작하는 모든 프로세스들이 마감시간을 어기는 경우에만 스케줄 불가능하다고 판단한다. 그 이유는 여러 프로세스 중 한 프로세스만이라도 마감시간을 어기지 않으면, 그 프로세스가 수행되도록 스케줄하면 되기 때문이다. 그 결과는 36번 문장에서 missed라는 변수에 저장된다. 그리고 여러 프로세스들이 여러 개의 채널에 대하여 동작하는 경우는 한 채널에 대한 동기화 동작으로 인하여 마감시간을 어기게 되면 그 프로세스는 마감시간을 지키지 못한다고 판단한다. 이것은 38번 문장의 unschable이라는 변수에 결과가 저장된다.

[3]의 방법은 마감시간이 0인 경우에 스케줄 불가능하다고 판단하지만, 본 논문에서는 위에서 설명한 첫번째 경우에 slack이 음수이면, 즉, 남은 계산시간이 마감시간보다 큰 경우 스케줄 불가능하다고 판단한다. 따라서 최소한 마감시간을 어기는 단위 시간만큼 먼저 스케줄이 불가능하다고 판단할 수 있으며, 두번째 경우도 동기화가 끝나는데 필요한 최소 시간 A가 마감시간 C와 동기화가 끝난 후 남은 동작의 최소 시간 B의 차이보다 크면 스케줄 불가능하다고 판단하므로, Fredette의 방법보다 더 이른 단위 시간에 마감시간을 어기는 것을 알 수 있으므로 스케줄이 불가능하다고 판단할 수 있다.

43번 문장에서 수행할 프로세스는 다음과 같은 기준으로 선택된다.

① 동기화 동작
② 동기화 동작까지의 최소 간격
③ 최소 슬랙
④ 최소 주기

이 기준을 본 논문에서 스케줄 가능한 프로세스의 수행 순서를 결정하는 임계 경로를 구성하는 원칙으로 정했으며, 이들 기준이 모두 같고 바로 앞에서 수행된 프로세스가 프로세스의 집합에 있다면, 그 프로세스를 선택한다. 또한, 위에서 제시한 네 개의 임계 경로를 구성하는 원칙을 바꾸어도 스케줄가능성을 판단하는데 거의 영향을 미치지 않으며, 스케줄이 불가능함을 판단하는 단위 시간도 거의 일정하다.

Ⅳ. 적용 및 구현

4.1 적용

본 논문에서 제안된 스케줄링가능성 분석 알고리즘의 정확성를 분석하기 위해 예를 들어, 두 프로세스가 하나의 처리기에 할당되어 수행할 경우의 마감시간을 지키는가의 여부에 대하여 제안된 스케줄링가능성 분석 알고리즘으로 적용해 보면, 기존의 방법인 Fredette의 방법에서는 그림 5와 같이 9개의 상태공간이 생성되고 이 상태를 모두 탐색하여야 프로세스 P_a가 마감시간을 어긴다는 사실을 알 수 있다.

그러나, 본 논문의 방법을 이용하면 알고리즘 1에 의해 표 2가 작성되고, 알고리즘 2에 의해 표 3의

분석 과정이 작성된다. 이 표 3을 살펴보면 3 단위 시간에 $A_b > C_a-B_a$가 되어 P_b가 마감시간을 어김을 알 수 있다. 따라서 프로세스의 상태는 그림 6과 같다.

표 2. P_a, P_b의 SPT

Table 2. Structure of P_a, 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 table

P_a				P_b			
A_a	B_a	C_a	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

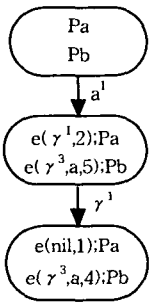


그림 6. 프로세스 상태공간

Fig. 6. Process state space

4.2 구현

본 논문에서는 최소 슬랙 우선, 최선 마감시간 우선, 비율 단조 스케줄링 방법을 혼합한 스케줄링 방법을 기반으로 하여, 부록에 제시된 CCS-R로 표현된 임의의 실험 대상 실시간 프로세스에 대하여 Fredette의 방법과 본 논문에서 제안한 방법을 적용하였을 때, 마감시간을 지키지 못함을 결정하는 단위 시간을 비교하기 위해 PC 상에서 Turbo C 언어로 구현하였다. 제안된 스케줄가능성 분석을 수행하는 단계는 그림 7과 같다.

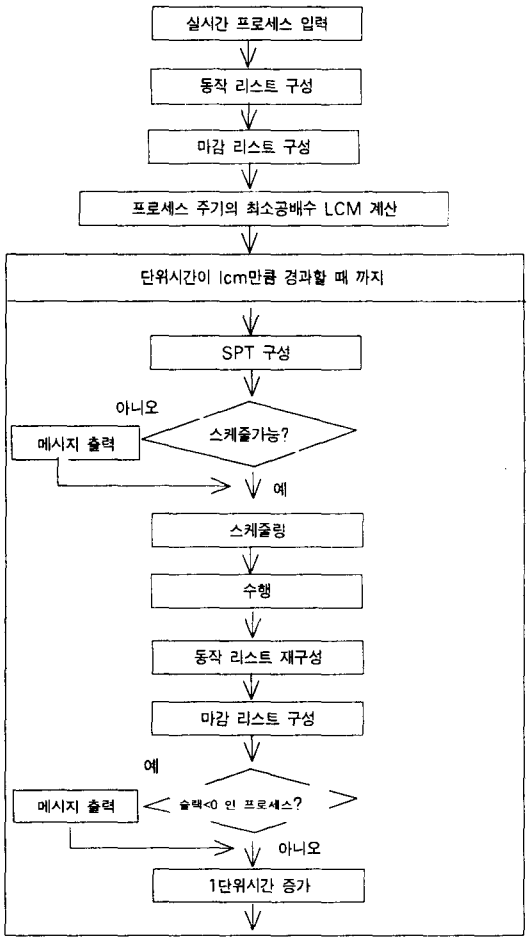


그림 7. 수행과정

Fig. 7. Execution procedure

4.2.1 실시간 프로세스의 입력

실시간 프로세스의 입력은 실험용 언어로 표현된 실시간 프로세스를 입력하는 것이다. 실험용 언어는 CCS-R을 컴퓨터에 입력하기 쉽도록 변환한 것으로 프로세스 term을 실험용 언어로 표현하는 방법은 다음과 같다.

(1) 프로세스의 이름은 'P'로 시작하고 뒤의 문자로 구분한다.

(2) 동기화 동작은 'a'로 시작하는 식별자(identifier) 로, a 이후에 번호를 붙여 동기화 동작을 구분 하며, send 동작은 '!'로 receive 동작

은 '?'를 동기화 동작의 이름 뒤에 붙여 구분한다.

(3) 계산 동작은 c로 표현한다.

(4) 최대 예상 수행 시간(앞으로 수행 시간이라 표현한다)은 각 프로세스 term 뒤에 표현한다.

실험용 언어의 표현 방법에 따라 예를 들면, 그림 8과 같다.

P_a ::= e(a^1;a^1;γ^4, 6);P_a

P_b ::= e(γ^1;a^1;sw(γ^1; $\overline{b}^1$, 4), 10)

(a) CCS-R로 작성된 실시간 프로세스

```
Pa ::= e(a1!1;a1!1;c4, 6);Pa;
Pb ::= e(c1;a1!1;sw(c1;a2?1, 4), 10);Pb;
```

(b) 실험용 언어로 작성된 실시간 프로세스

그림 8. CCS-R과 실험용 언어

Fig. 8. Experimental language and CCS-R

입력된 실시간 프로세스를 저장하는 구조는 다음과 같다.

```
enum symbol { proc, rcv, snd, comp, start, finish,
      exact, seq, alt, nil, lparent, rparent, num,
      assign, comma, slash, nosy };
struct proc_term {
 enum symbol type; /* 동작의 종류 */
 char name[4];       /* 동작의 이름 */
 int dur;   /* 동작의 수행 시간 *
 struct proc_term *term; /*부분 동작에 대한 포인터*/
 int dl;    /* 동작의 마감시간 */
 struct proc_term *link; /*다음 동작에 대한 포인터*/
 };
```

4.2.2 동작 리스트와 마감시간 리스트의 구성

동작 리스트는 시간이 지남에 따라 변화하는 실시간 프로세스의 전이를 구현하기 위하여 시간 구성자를 제외한 계산과 동기화 동작을 수행 시간과 함께 저장하는 리스트이며, 다음과 같은 노드 구조로 구성된다.

동작 리스트는 프로세스당 하나씩 할당되며, 해당 프로세스가 수행될 때마다 수행 시간을 1씩 감소하며, 수행 시간이 0이 되면 첫 노드를 제거한다.

```
struct a_node {
  enum symbol type;          /* 동작의 종류 */
  char name[4];     /* 동작의 이름 */
  int dur;          /* 동작의 수행 시간 */
  struct a_node *link;/* 다음 동작에 대한 포인터 */
  };
```

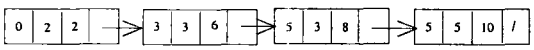
마감시간 리스트는 실시간 프로세스가 수행될 때, 마감시간을 관리하기 위하여 사용되는 리스트로, 다음과 같은 노드 구조로 구성된다. 여기서는 프로세스당 하나의 리스트가 할당되며, 리스트 내의 각 노드는 하나의 마감시간에 제약을 받는 연속된 동작의 모임인 블럭마다 하나씩 할당된다.

```
struct d_node {
  int dur;  /* 경과 시간 */
  int slack;          /* 슬랙 */
  int dl;   /* 마감시간 */
  struct d_node *link; /* 다음 노드에 대한 포인터 */
  };
```

위에서 기술한 마감시간 리스트의 경과 시간 dur은 프로세스의 시작으로 부터 지정된 동작이 수행되는데 걸리는 최소 시간이다. 마감시간 리스트의 예는 그림 9와 같이 구성된다.

P ::= e(sw(γ^1;a^1, 3);γ^1;fw(γ^1;b^1, 8), 10);P

(a) 실시간 프로세스



(b) 마감시간 리스트

그림 9. 실시간 프로세스의 마감시간 리스트

Fig. 9. Deadline list for real-time process

그림 9의 (a)에서 sw(γ^1;a^1, 3)는 3단위 시간 이내에 수행을 시작해야 하므로 sw(γ^1;a^1, 3) 앞의 동작들은 마감시간이 2이다. 앞에 동작이 없으므로 수행 시간은 0이다. 따라서 이 프로세스의 마감시간 리스

트의 첫 노드는 (0, 2, 2)이다. fw(γ^1;b^1, 8)가 마감시간을 지키기 위해서는 앞의 동작들이 6단위 시간 이내에 수행을 마쳐야 한다. 따라서 sw(γ^1;a^1, 3)의 동작과 다음의 γ^1로 구성되는 블럭의 마감시간은 6이다. 이 블럭의 마감시간 리스트의 노드는 (3, 3, 6)이다. fw(γ^1;b^1, 8)로 구성되는 블럭은 마감시간이 8이므로 마감시간 리스트 노드는 (5, 3, 8)이다. fw(γ^1;b^1, 8)이 끝난 뒤에는 10단위 시간까지 대기하여야 하므로 경과 시간은 앞 노드와 같고 마감시간은 10이다.

마감시간 리스트는 매 단위 시간마다 갱신된다. 스케줄된 프로세스의 마감시간 리스트 노드의 경과시간과 마감시간은 1씩 감소되며, 그 외 프로세스의 마감시간 리스트 노드의 마감시간과 슬랙은 1씩 감소된다. 이 때 슬랙이 음수가 되면, 해당 프로세스는 마감시간을 지키지 못한다고 판단한다.

4.2.3 스케줄링과 SPT의 구성

수행될 프로세스를 선정하는 스케줄링은 Ⅲ장에서 언급한 바와 같이 슬랙이 최소인 프로세스, 마감시간이 가장 빠른 프로세스, 주기가 가장 짧은 프로세스를 선택한다. 이를 위하여 위의 세 가지 기준으로 프로세스를 순서 배열한 우선 순위 리스트를 운영한다. 그 우선 순위 리스트의 노드 구조는 다음과 같다.

```
struct sch_node {
  int pno;    /* 프로세스 번호 */
  int slack;          /* 슬랙 */
  int dl;   /* 마감시간 리스트내 첫 노드의 마감시간 */
  int interval;       /* 프로세스의 주기 */
  struct sch_node *link;
  };
```

동기화 프로세스 테이블 SPT의 구성 방법은 Ⅲ장에서 설명하였으며, SPT의 노드 구조는 다음의 페이지에 나타내었다.

```
struct sss {
  int pno; /* 프로세스 번호 */
  int dl;   /* 마감시간 */
  struct sss *link;
  };
```

```
struct spt_node {
  char name[4];       /* 동기화 동작의 이름 */
  enum symbol type; /* 동기화 동작의 종류 */
  struct sss *s_ndx; /* 동기화 동작을 수행하는 프
                        로세스, 그 프로세스의 경
                        과시간, 마감시간을 갖는
                        노드에 대한 포인터 */
  };
```

4.2.4 구현 결과 분석

표 4는 프로세스 주기의 최소공배수 그리고 Fredette와 본 논문에서 제안한 방법으로 분석하였을 경우에 걸리는 단위 시간을 나타낸다.

실험 대상 실시간 프로세스 중 N1, N9, N10, N11, N13은 시간 구성자를 포함하지 않은 주기 프로세스이며, 나머지는 **sw**, **fw**와 같은 시간 구성자를 포함하는 주기 프로세스이다. 본 논문에서 제안한 방법으로 부록에서 제시한 13개의 실험 대상 실시간 프로세스를 적용했을 때, 모두가 Fredette의 방법을 적용한 경우보다 이른 시간 이내에 스케줄이 불가능함을 판단한다.

표 4. 실험 결과 비교

Table 4. Experimental result

구분 / 프로세스 번호	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	

그리고 N8, N12, N13은 슬랙이 음수일 때 스케줄이 불가능한 경우이고, 나머지는 동기화가 끝나는데 필요한 최소 시간이 마감 시간과 동기화가 끝난 후 남은 동작의 최소 수행 시간의 차이 보다 크지 않을 경우에 스케줄이 불가능하다고 판단하는 것이다. 전체적인 절감률은 $\left(1-\frac{41}{85}\right)\times 100$ = 52%로 Fredette의 방법에 비하여 약 1/2 단위 시간만으로 스케줄이 불가능함을 판단하였다. 또한 절감된 단위 시간은 평균적으로 최악의 경우에 수행해야 할 주기의 최소 공배수의 약 1/3인 $\left(\frac{41}{122}\right)\times 100$ = 35%라는 결과를 얻었다. 이 방법은 시간 구성자를 포함하여 시간 제약이 많은 프로세스에 대하여 특히 높은 성능 향상을 보였다.

V. 결 론

본 논문에서 분산 실시간 프로그램의 스케줄가능성 분석 방법을 제안하고 구현하였다. 본 논문의 스케줄가능성 분석기는 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 프로세스의 상태공간을 생성하기 위하여 프로세스들이 마감시간을 지킬 확률이 높은 동적 스케줄링을 사용한다. 따라서 고정 스케줄링 방법을 사용하고, 발생가능한 모든 프로세스의 상태공간을 탐색하는 방법을 사용하기 때문에 시간, 공간적 복잡도가 높은 Stoyenko, Gerber, Fredette 등의 스케줄가능성 분석기들과는 달리 적은 수의 상태를 생성하고, 이른 단위시간내에 스케줄이 불가능한 프로세스를 판별할 수 있다.

향후 연구 계획으로는 스케줄링이 불가능한 실시간 프로세스의 상태공간을 PC 화면상에 시각적으로 표시해 주는 추가의 연구가 필요하다.

부 록

실험 대상 실시간 프로세스는 다음과 같다.

(1) N1

Pa ::= e(a^1, 3);Pa;

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

(2) N2

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;

(3) N3

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; $\overline{d}^1$, 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;

(4) N4

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;

(5) N5

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;

(6) N6

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($\overline{d}^1$;c^1, 8);Pd;

(7) N7

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;

Pc ::= e(fw(a^1;γ^1, 4);γ^1, 10);Pc;

(8) N8

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;

(9) N9

Pa ::= e(a^1;γ^1, 3);Pa;

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

(10) N10

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

Pb ::= e(a^1;a^1;γ^4, 6);Pb;

(11) N11

Pa ::= e($\overline{a}^1$;γ^1, 3);Pa;

Pb ::= e(a^1;γ^3;a^1, 6);Pb;

(12) N12

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;

(13) N13

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;

참고문헌

[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, 1993, pp.274-283.

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

[4] 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, Sep. 1992, pp.768-784.

[5] 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, 1992, pp.146-155.

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

[7] F. Moller and C. Tofts, "A Temporal Calculus of Communicating Systems," Proceedings of CONCUR '90, LNCS 458, 1990, pp.401-415.

[8] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," IEEE Transactions on Computers, Vol.38, No.8, pp.1110-1123, 1989.

[9] J.A. Stankovic, K. Ramamritham, and S. Cheng, "Evaluation of Flexible Task Scheduling for Distributed Hard Real-Time Systems," IEEE Transactions on Computers, Vol.34, No.12, pp.1130-1143, 1985.

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

[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.

박 흥 복(Hung-Bog Park)

1981년 2월 경북대학교 컴퓨터 공학(공학사)

1984년 2월 경북대학교 대학원 컴퓨터공학(공학석사)

1995년 8월 인하대학교 대학원 전자계산학(이학박사)

1984년~1995년 동명대학 전자계산학과 부교수

1996년~현재 부경대학교 컴퓨터·멀티미디어 공학부 부교수

*관심분야 : 멀티미디어 프로그래밍, 컴파일러, 실시간 시스템, 자동화 시스템

김 춘 배(Choon-Bae Kim)

1993년 2월 부산외국어 대학교 컴퓨터 공학과 졸업 (공학사)

1996년 2월 부산외국어 대학교 대학원 컴퓨터 공학과 졸업 (공학석사)

1998년 3월~현재 부경대학교 대학원 전자계산학과 박사과정

*관심분야 : 멀티미디어 프로그래밍, 실시간 시스템 응용