실시간 프로세스의 스케줄 가능성 분석을 위한 스케줄링 상태 분석기에 관한 연구

박흥복*

A Study on Scheduling State Analyzer for Schedulability Analysis of Real-Time Processes

Hung-Bog Park

본 연구는 1998년도 부경대학교 기성회 연구비에 의해 수행 되었음

요 약

실시간 스케줄가능성 분석에 대한 기존의 방법들은 프로세스의 모든 상태 공간을 추적하거나 고정 우선 순위 스케줄 방법을 사용하기 때문에 시간 및 공간에 대한 복잡성을 야기한다. 본 논문에서는 프로세스 대수의 전이 규칙을 이용하여 프로세스의 최소 수행 시간, 주기, 마감 시간, 동기화 시간을 고려하여 실시간 프로세스가 마감 시간을 지키는가를 판단하고, GUI 환경을 기반으로 스케줄이 불가능한 프로세스에 대해 스케줄가능하게 하는 회복 알고리즘을 제안한다. 또한 스케줄이 불가능한 실시간 프로세스에 대해 회복 알고리즘을 적용한 결과를 시각적으로 표시해 주는 스케줄링 상태 분석기를 구현, 평가한다.

ABSTRACT

The existed approaches to analyzing real-time schedulability take place exponential time and space complexity of this methods, since these uses a fixed priority scheduling and/or traverse all possible state spaces.

This paper judges whether it is satisfied a given deadlines for real-time processes regarding a minimum execution time of process, periodic, deadline and a synchronizion time of processes by using the transaction rules of process algebra, and proposes a retrieval algorithm for unchedulable processes based on GUI environmnt. And we implement and evaluate the scheduling state analyzer that displays visually the result of schedulability analysis for real-time processes.

* 부경대학교 전자컴퓨터정보통신공학부
접수일자: 2001. 1. 22

Ⅰ. 서 론

실시간 시스템이란 시스템의 운영과 외부 세계와의 상호 관계가 시간과 깊이 관련되어 있어 외부에서 발생한 요구에 대하여 제한된 시간 내에 정확하게 처리될 수 있도록 구축된 시스템이다. 대표적인 예로서, 핵발전, 미사일 제어, 공장 자동화, 항공 교통 제어 등이 있으며, 마감 시간 준수 여부를 판단하기 위한 시스템 기능의 수행 여부는 시스템 전체 운용에 따른 인력과 비용이 소요될 수도 있는 중요한 문제이다. 더욱이 시스템 작업 수행을 위한 프로세스가 마감 시간내에 스케줄불가능할 때 발생하는 문제를 처리하기 위해서는 전체 번역 시간에 대해 최소 수행 시간으로 스케줄링하는 회복 알고리즘의 사용은 불가피하다.

스케줄가능성 분석(schedulability analysis)이란 프로세스가 특정 스케줄링 정책하에서 수행될 때 프로세스가 마감 시간을 만족하는가를 판단하는 것이다. Fredette[3,4]는 프로세스 대수를 이용하여 전체의 프로세스 상태 공간을 탐색하는 스케줄가능성 분석을 제안하였으며, Bruno[1] 등은 페트리 넷(Petri Net)을 이용하여 주어진 시간 내에 전체 프로세스의 스케줄 가능성을 분석하는 방법을 제안했으며[1], Liu와 Layland[2,5]는 비율 단조 스케줄링시 독립적인 프로세스에 대해 주어진 조건들에 대해 마감 시간을 지키는가를 판단하는 방법을 제안하였다.

[6]에서는 Fredette의 방법과는 달리, 시간 제약을 만족하는 스케줄을 구성하는 확률이 높은 최소 슬랙 스케줄링, 최소 마감 시간 스케줄링 , 비율 단조 스케줄링 방법을 이용하여 실시간 프로세스의 스케줄가능성 분석에 대한 알고리즘은 제안하였지만, 스케줄 불가능한 프로세스에 대한 회복 알고리즘은 제안되지 않았다.

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 마감 시간을 지키지 못할 때의 대안을 제시하지 못한다.

본 논문에서는 실시간 명세 언어로 기술된 각 프로세스 정보를 입력댁으로 받아 기존에 제안된 프로세스의 스케줄가능성 분석[4,6]시에 스케줄이 불가능한 프로세스에 대하여 스케줄 가능하도록 하는 회복 알고리즘을 제안하였으며, 스케줄불가능한 프로세스의 상태 공간에 대한 회복 알고리즘을 적용한 결과를 GUI 환경하에서 스케줄가능한 상태 공간을 시각적으로 표시해 주는 스케줄링 상태 분석기를 Visual C++ 6.0으로 구현하고 평가하였다.

Ⅱ. 실시간 명세 언어

실시간 명세 언어는 실시간 프로세스의 마감 시간 만족 여부를 번역 시간에 판단하기 위해 프로세스의 시간 특성과 프로세스간의 관련성을 표현하는 명세 언어이다.

2.1 프로세스 대수

프로세스 대수의 구문과 연산의미(operational semantics)는 다음과 같이 CCS[1,5]에서 찾아 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-R

nil은 프로세스 종료를 나타내고, X::=P는주어진 환경내에서 변수 X에 한정되는 프로세스 P를 의미한다. a.P는 동작 a를 수행한 후, 프로세스 P와 같이 동작하며, P+Q는 P 또는 Q 두 프로세스 사이의 선택을 나타내며, P||Q는 P와 Q 두 프로세스의 병렬 조합을 의미한다. 대문자는 프로세스, 소문자는 프로세스를 구성하는 동작이며, 두 프로세스는 독립적으로 동작을 수행할 수도 있고, 보동작(complementary action)으로 동기화 될 수 있다. 동작 a의 보동작은 $\overline{a}$로 표현된다.

2.2 명세 언어 CCS-R

실시간 명세 언어 CCS-R은 CCS에 시간 개념을 추가한 것으로 수행할 시간과 수행 시간에 대한 제약을 갖는 실시간 프로그램의 특성을 추출하기 위한 실시간 명세 언어이다. CCS-R로 기술된 프로세스는 수행을 위한 명령 동작들로 구성되며, 각 동작은 그 동작을 수행하는데 필요한 시간 간격 d를 갖는다[1,6].

CCS-R 프로세스의 구문은 그림 2와 같은 BNF 문법으로 정의된다[3,6]. 프로세스가 포함하는 시간 제약과 시간 특성은 시간 구성자 sw, fw, e로 표현된다. d를 마감 시간이라 할 때, 마감 시간은 프로세스가 시작된 시간으로부터 상대적으로 계산된다. sw(P, d)는 프로세스 P가 d 단위 시간이내에 시작해야 한다는 것을 나타내며, fw(P, d)는 프로세스 P가 d 단위 시간 이내에 끝나야 한다. e(P, d)는 정확히 d 단위 시간 이내에 종료하는 프로세스로 주기 프로세스를 표현하는데 사용된다. 만약 P가 d'에 끝나고 d>d'이면, P는 단위 시간 동안 지연되어야 한다.

$P::=nil|\ \gamma^{d}|\ \delta^{d}|\ a^{d}|P+Q|P;P|sw(P,d)|fw(P,d)|\ e(P,d)$

그림 2. CCS-R의 구문
Fig. 2 Syntax of CCS-R

nil은 프로세스 종료를 나타내고, γ^{d}는 d 단위 시간이 걸리는 동작을 의미하며, δ^{d}는 d 단위 시간동안 지연하는 동작, a^{d}은 d 단위 시간이 걸리는 동기화 동작을 나타낸다. P+Q는 P와 Q 프로세스 사이의 선택을 나타내며, P;P는 P와 P의 순차조합을 의미한다. sw(P,d)는 d 단위 시간 내에 시작하는 주기 프로세스를 나타내고, fw(P,d)는 d 단위 시간 내에 종료하는 주기 프로세스를 의미하고, e(P,d)는 d 단위 시간 내에 정확히 종료하는 프로세스를 나타내고 있다. 실시간 프로세스가 마감 시간을 지키는가를 번역 시간에 판단하기 위해서는 프로세스의 동작뿐만 아니라 프로세스의 수행 시간을 번역 시간에 계산할 수 있거나 프로그램상에 명시되어야 한다. 프로세스의 수행 시간을 나타내는 기간 함수(duration function)는 [정의 1]과 같다.

[정의 1] 기간 함수 d

- $\mathbb{D}$는 기간 정의역(duration domain)이다. 기간 정의역의 원소를 d라 하면 $\forall d,\ d > 0$이다.
- A는 모든 동작의 집합 $\{ ad, \forall d, \delta d \mid d \in \mathbb{D} \}$이다. $a \in A$에 대하여 $\overline{a}$는 보동작으로 A의 원소 이다.
- 기간 함수 $d : A \rightarrow \mathbb{D}$는 모든 동작의 기간이다. $\forall a \in A,\ d(a)=d(\overline{a})$이다.

III. 스케줄가능성 분석 및 회복 알고리즘

스케줄가능성을 분석하고 스케줄불가능한 실시간 프로세스에 대해 스케줄가능하게 하는 회복 알고리즘을 제안한다.

3.1 스케줄가능성 분석

Fredette의 알고리즘[3,4]은 실시간 프로세스가 스케줄가능한가를 판단하기 위해 생성되는 실시간 프로세스들의 모든 상태들 중 마감 시간이 0인 프로세스를 포함하는 상태가 있는 지를 검사하여 마감 시간이 0인 상태에 도달하는 프로세스가 있다면 스케줄이 불가능하다고 판단한다. 예를 들어, Fredette의 방법에 따라 아래의 두 프로세스 Pa, Pb에 대한 스케줄가능성 분석을 위한 상태 공간은 그림 3과 같다. Pa는 3 단위 시간마다 채널 a에 메시지를 보내고, 1 단위 시간 동안 수행한 후 다시 Pa를 수행하는 것이며, Pb는 6 단위시간마다 메시지를 채널 b에서 받고 3 단위시간 동안 수행하고 채널 b에 메시지를 받은 다음에 다시 Pb를 수행한다.

$Pa ::= e(\overline{a};\gamma 1,\ 3);Pa$
$Pb ::= e(a;\gamma 3;a,\ 6);Pb$
$d(a) = d(\overline{a}) = 1$

이들 프로세스는 어떤 경우에도 예외상태에 도착하므로 단일 처리기 시스템에서는 마감 시간을 지키지 못한다. 그림 3에서 프로세스가 마감 시간을 지키지 못함을 판단하는 것은 마감 시간이 0인 상태(진한 타원)에 도달하는가를 판단하는 것이다. 이를 위해서는 발생가능한 모든 상태를 탐색하여 마감 시간이 0인 상태에 도달하면 스케줄 불가능하다고 판단한다. 이러한 도달가능성 분석은 프로세스의 수에 따라 지수시간을 갖는 알고리즘으로 수행된다. 따라서 프로세스가 마감 시간을 지키지 못함을 효율적으로 판단하기 위하여 탐색할 상태의 수를 줄이거나 보다 빠른 단위시간 내에 판단하는 방법이 필요하다.

그러나, 기존의 알고리즘[6]을 이용하면 *표시된 상태 공간에서는 3 단위 시간이 지난 후의 상태를 보면, Pa가 마감 시간을 지키려면 2 단위 시간전에 동기화 동작 $\overline{a}$를 수행하여야 한다. 하지만 Pb가 동기화 동작

a를 수행하려면, 최소한 3 단위 시간이 필요하다. 따라서 Pa는 마감 시간을 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 마감 시간보다 길면 그 프로세스는 마감 시간을 지키지 못함을 미리 판단할 수 있다.

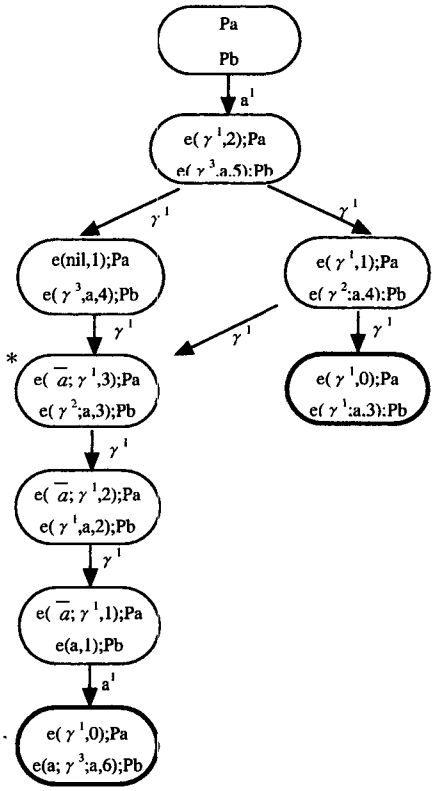

그림 3. 스케줄불가능한 프로세스의 예
Fig. 3 Example of unschedulable processes

또한, 비율 단조 스케줄링 방법과 같은 정적 스케줄링대신 마감 시간을 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 마감 시간을 지키도록 하는 프로세스의 수행 순서를 얻을 수 있다.

그리고 동기화 동작을 수행하는 프로세스에 대하여 동기화 동작을 수행하기 위해 지나야 하는 시간, 동기화 시간, 동기화 동작을 수행한 후 나머지 동작을 수행하는 시간의 합이 한 프로세스의 마감 시간보다 길면 그 프로세스는 마감 시간을 지키지 못한다는 것을 미리 판단한다는 방법을 이용하여 스케줄이 불가능할 경우의 스케줄 회복 알고리즘을 제안한다.

3.2 회복 알고리즘

기존의 실시간 스케줄 가능성 분석에서 제시된 스케줄 가능성 분석을 위한 식(1)과 제시된 스케줄 불가능한 프로세스를 스케줄가능하게 하는 표 1의 처리기 할당 규칙을 이용하여 스케줄 불가능한 프로세스에 대해 스케줄이 가능하게 하는 회복 알고리즘을 제안한다. 아래의 식(1)은 기존에 제시된 스케줄가능성 분석을 위한 판단식이다[6]. 실시간 프로세스가 스케줄가능할 경우에는 식(1)을 만족해야 한다.

$$\sum_{i=0}^{n}(Ci+\frac{Ai}{2})\times\frac{W}{Ti}\le W \quad \cdots\cdots (1)$$

Ci는 프로세스 Pi의 계산 시간이며, Ai는 프로세스 Pi의 동기화 시간이다. W는 프로세스 Pi의 주기의 최소 공배수이며, Ti는 프로세스 Pi의 주기이다.

스케줄이 불가능한 프로세스가 스케줄가능하도록 하기 위해 처리기 할당의 기본 규칙을 표 1과 같이 설정할 수 있다.

표 1. 처리기 할당 규칙
Table 1. Allocation Rules of Processor

식(1)	기존의 알고리즘	처리기 할당 기준
스케줄가능	스케줄불가능	프로그램 코드의 순서를 변경하여 단일 처리기에 할당한다
스케줄불가능	스케줄불가능	프로그램 코드의 순서와 마감 시간을 변경하고 프로세스를 여러 처리기에 분담하여 할당한다

식(1)에서 제시된 판단식과 스케줄가능성 분석 알고리즘을 통해 처리된 결과를 조합하여 적용될 회복 알고리즘을 선택할 수 있다. 따라서 알고리즘 1에서는 기

존의 알고리즘[6]으로 스케줄가능성을 판단한 결과가 스케줄가능하고, 식(1)이 스케줄불가능할 경우, 알고리즘 2는 기존의 알고리즘이 스케줄불가능하고 식(1)도 스케줄불가능할 경우의 회복 알고리즘을 제안한다.

[알고리즘 1] 식(1)의 결과가 스케줄 가능할 때 적용될 회복 알고리즘

```
시스템 변수 초기화;
명령어 순서 초기 조합;
for(factorial(각 프로세스에 포함된 명령어 수-1))
  스케줄가능성 분석;
  if(스케줄 가능)
    회복 처리 작업 종료;
  end of if
  다음 프로세스 명령 순서 조합;
end of for
```

[알고리즘 2] 식(1)의 결과가 스케줄 불가능할 때 적용될 회복 알고리즘

```
시스템 변수 초기화;
프로세스 수만큼 프로세스 처리기 할당;
do
  실행 프로세스 결정;
  for(프로세스 수)
    switch(실행 프로세스의 명령어 코드)
      동기화 동작일 때:
        동기화 대상 프로세스 탐색;
        if(동기화 대상 프로세스가 존재할 때)
          동기화 동작 수행;
        else
          동기화 대기;
        end of if
      계산 동작 일때:
        계산 동작 수행;
      모든 명령어이 종료되었을 때:
        if(마감 시간 == 1)
          프로세스 초기화;
        end of if
    end of switch
  end of for
    마감 시간 및 주기수 감소;
    if(수행될 작업시간>마감 시간)
      스케줄 가능 플래그 = false;
      return
    end of if
  while(마감 시간의 최소 공배수)
    스케줄 가능 플래그 = true;
  end of while
  return;
```

제시된 알고리즘 1과 2를 이용한 회복 처리가 실패될 경우 다시 마감 시간을 1씩 증가시켜 가면서 회복 알고리즘을 재 수행한다.

스케줄가능성 분석 및 처리된 결과는 다음 과정의 처리 혹은 출력을 위해 구조체에 저장된다. 본 논문에서 각 단계의 분석을 통해 구축된 구조체의 내용을 화면상에 시각적으로 나타내는 방법은 알고리즘 3과 같다.

[알고리즘 3] 화면 출력 알고리즘

```
디바이스 초기화 및 초기 출력 x, y 좌표값 설정;
 for(처리된 interval의 가중치)
    상태 도형 출력 위치 설정;
    if(초기 출력)
      시작 문자열과 화살표를 표시;
    end of if
    if(마지막 출력도형)
      if(스케줄 불가능하다면)
       반전된 상태 도형 출력;
       '*'문자 출력;
      else
       초기 상태 화살표 및 문자 출력;
    else
      상태 도형 및 해당 명령어 출력;
      다음 상태 지정 화살표 및 실행 명령어 출력;
    end of if
    다음 상태 출력을 위한 y축 좌표값을 보정;
end of for
```

Ⅳ. 스케줄링 상태 분석기의 구현 및 평가

실시간 프로세스가 스케줄가능한지를 분석하고, 스케줄불가능할 경우 스케줄가능하게 하는 스케줄링 상

태 분석기를 구현하고 평가한다.

4.1 구현

본 논문에서 제안된 스케줄링 회복 알고리즘의 정확성을 분석하기 위해 그림 5와 같은 또다른 예를 들어 Visual C++6.0을 이용하여 구현하는 스케줄링 상태 분석기는 그림 4와 같다.

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, ε 이후에 번호를 붙여 동기화 동작을 구분 하며, send 동작은 '!'로 receive 동작은 '?'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.

그림 5의 (a)는 CCS-R로 표현된 입력 프로세스들이며, (b)는 (a)를 실험용 언어로 변환시킨 프로세스들이다.

스케줄링 상태 분석기는 실시간 명세 언어를 입력덱으로 받아 처리한 후, 출력을 위한 구조체에 결과를 저장한다. 저장 구조체는 그림 6과 같은 인자를 가진다.

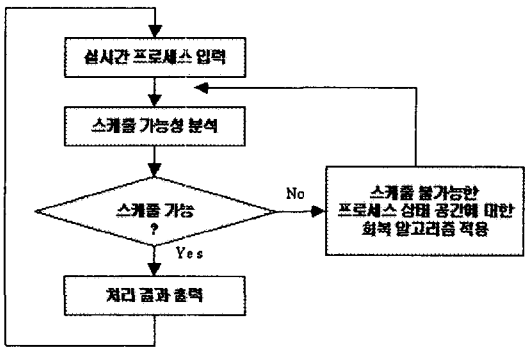

그림 4. 스케줄링 상태 분석기의 흐름도
Fig. 4 Flowchart of scheduling state analyzer

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

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

(a)CCS-R로 작성된 실시간 프로세스

Pa::=e(a1!1,3);Pa;
Pb::=e(a1?1;a1?1;r4,6);Pb; --- 프로세스 (1)

Pa::=e(a1!1;r1,3);Pa;
Pb::=e(a1?1;r3;a1?1,6);Pb; --- 프로세스 (2)

(b) 실험용 언어로 작성된 실시간 프로세스

그림 5. CCS-R과 실험용 언어
Fig. 5 Experimental language and CCS-R

```
typedef struct NODE_LIST{
  int   Pi_Num;            //프로세스번호
  char Pi[2];     //프로세스명
  OP_  OP_CODE[Max];//계산동작들의 집합
  int   St;        //슬랙
  int   Limit;     //제한시간
  char OP_array[MAX_OP_CODE_NUM];//코드 배열
  int OP_index; //현재 수행중인 OP코드의 위치
  int   OP_num;       //현재 가진 OP코드의 수
  int   ct;        //총 계산 시간(Cti)
  int   at;        //총 동기화 시간    }NODE_;
```

(a) 실시간 명세 언어 입력덱 저장 구조체

```
typedef struct ANALY_TABLE{
  int  A;  //프로세스의 동기화가 끝나는 시간
  int  B;  //동기화 동작 후, 남은 동작의 최소수
           행 시간
  int  C;  //프로세스의 마감 시간
  int  si;  //슬랙
  CString exec_code;  //수행 명령어
  CString exec_flag;  //명령어 처리 플래그
  CString OP_str;    //프로세스 상태 공간 명령
           문자열
```

```
char  Pi[2];      //프로세스명     }ANALY_;
```

(b) 스케줄가능성 분석 테이블 저장 구조체

그림 6. 저장 구조체
Fig. 6 Storage structure

프로세스 상태 분석기는 이상의 구조체들을 통해 자료를 입·출력한다.

4.2 평가

본 연구에서 구현된 스케줄링 상태 분석기의 결과를 분석하기 위해 그림 5에서 제시된 2개의 프로세스를 적용한다.

우선 프로세스 (1)을 프로세스 상태 분석기를 통해 적용하면, 프로세스 (1)의 실험 프로세스들은 스케줄가능성 분석을 통해 두번째 interval에서 스케줄이 불가능하다고 판단된다. 즉, Pa가 Pb와 동기화 동작을 수행하기 위해서는 Pb가 $\gamma 3$ 를 수행하는 시간인 3 단위 시간 동안 기다려야 한다. 다음에 동기화 동작 $\overline{a}$를 수행하고 남은 계산 시간 $\gamma 1$ 을 수행하면 모두 5 단위 시간이 필요한데 마감 시간은 4 단위 시간이므로 마감 시간을 지키지 못한다. 이러한 분석 과정에 대한 처리 결과 및 현재 상태의 출력은 그림 7과 같이 표시된다.

스케줄링 상태 분석기는 입력덱 관리를 위한 추가 및 삭제, 선택 기능이 주어지고, 스케줄가능성 분석 기능과 회복 기능, 그리고 출력 기능을 개별적으로 분리해 둠으로써 각 단계별 처리 상태를 자세히 분석할 수 있도록 하였다. 그리고 처리 과정과 결과를 유도하는 과정에서 도출되는 유용한 정보는 화면 하단의 Analysis Information 컨트롤과 Status Displayer 컨트롤에 텍스트로 출력되며, 프로세스 상태 공간을 화면에 표시 할 수 있다.

스케줄가능성 분석에 대한 첫번째 과정 처리 결과(SPT 테이블)는 수행 후, 각 행에 대해 메시지 박스를 이용해서 화면에 출력되며, 두번째 과정에 대한 처리 결과(분석 테이블) 및 기타 정보는 상태 출력 윈도우 하단의 Analysis Information 컨트롤에 문자열로서 출력된다.

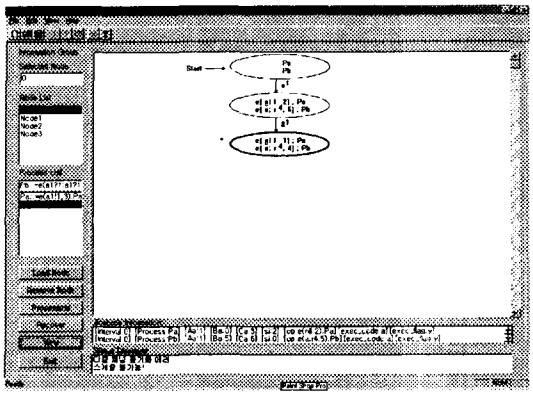

그림 7. 프로세스 (1)의 스케줄링 상태 분석기 윈도우
Fig. 7 Window of scheduling state analyzer for processes (1)

스케줄링 상태 분석기는 입력덱 관리를 위한 추가 및 삭제, 선택 기능이 주어지고, 스케줄가능성 분석 기능과 회복 기능, 그리고 출력 기능을 개별적으로 분리해 둠으로써 각 단계별 처리 상태를 자세히 분석할 수 있도록 하였다. 그리고 처리 과정과 결과를 유도하는 과정에서 도출되는 유용한 정보는 화면 하단의 Analysis Information 컨트롤과 Status Displayer 컨트롤에 텍스트로 출력되며, 프로세스 상태 공간을 화면에 표시할 수 있다.

스케줄가능성 분석에 대한 첫번째 과정 처리 결과(SPT 테이블)는 수행 후, 각 행에 대해 메시지 박스를 이용해서 화면에 출력되며, 두번째 과정에 대한 처리 결과(분석 테이블) 및 기타 정보는 상태 출력 윈도우 하단의 Analysis Information 컨트롤에 문자열로서 출력된다. 스케줄불가능한 프로세스의 상태 공간에 대해 적용될 회복 알고리즘을 선택하기 위해서 스케줄가능성 판단을 위한 식(1)을 적용하여 스케줄가능성 여부를 판단한다. 즉, C3=0, A3=1, T3=3, C4=4, A4=2, T4=6이고, W=6이므로 결과는 $(0+1/2)\times 6/3+(4+2/2)\times 6/6=6$이 된다. 따라서 프로세스(1)의 Pa, Pb는 마감 시간을 만족하므로 스케줄가능하다. 그 결과는 해당 처리 함수의 BOOL형 지역 변수 sch_jug에 저장된다.

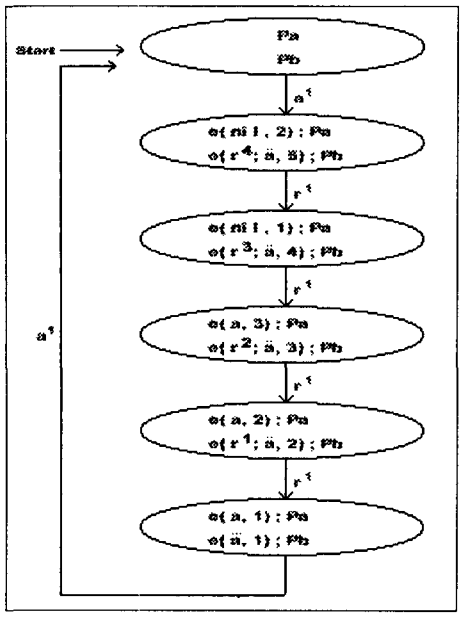

그림 8. 스케줄불가능한 프로세스 (1)에 대한 회복 알고리즘의 적용 결과
Fig. 8 Result of retreival algorithm for unschedu- lable processes(1)

그리고 스케줄가능성 분석 알고리즘을 적용한 결과인 스케줄불가능이란 판단은 전역 구조체인 T2_GLV의 unschable에 저장된다. 그러므로 두 변수가 가진 값의 경우가 조합되어 분기되므로, 여기에서는 스케줄가능성 분석 결과가 저장된 unschable이 부정이며, 식(1)의 처리 결과는 긍정이다. 따라서, 표 1에서 제시된 스케줄링 할당 규칙을 적용하면 프로그램 코드의 순서를 변경하여 단일 처리기를 할당하는 회복 알고리즘을 선택할 수 있다. 선택된 회복 알고리즘을 적용한 결과는 그림 8과 같다.

두번째 실험 프로세스인 프로세스 (2)를 스케줄링 상태 분석기로 수행하면, 스케줄가능성 분석 결과는 그림 9와 같이 프로세스 상태 공간에 이중원(*표시)으로 표시된 네번째 상태 공간에서 스케줄이 불가능하다는 것을 판단할 수 있다.

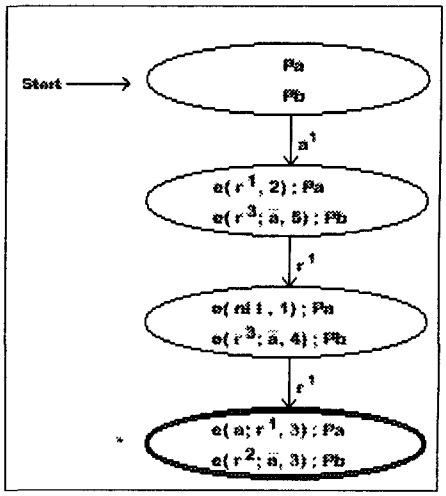

그림 9. 프로세스 (2)의 스케줄 불가능한 상태 공간
Fig. 9 Unschedulable States Space of Processes(2)

따라서 프로세스 (2)에 회복 알고리즘을 적용하면 프로세스 (1)의 처리와 같이 수행된다. 스케줄링 상태 분석기는 프로세스 (2)에 대해서 스케줄가능성 판단을 위해 식(1)을 적용하면, C1=1, A1=1, T1=3, C2=3, A2=2, T2=6이고, W=6이므로 결과는(1+1/2)×6/3+(3+2/2)×6/6=7>6이 된다. 따라서 프로세스(2)의 Pa, Pb는 하나의 처리기에서 수행될 때 마감 시간을 지키지 못하기 때문에 스케줄이 불가능하다. 또한, 기존의 알고리즘[6]에 적용해도 스케줄링 불가능함을 판단함으로 실시간 프로세스(2)는 스케줄이 불가능하다.

그러므로 제시된 회복 알고리즘 2를 적용하면, 그림 10과 같은 스케줄가능한 프로세스 상태 공간으로 표시된다. 특히, 스케줄링 상태 분석기는 프로세스 (2)의 처리를 위해 각 프로세스 당 한 개의 처리기를 할당하여 작업을 수행한다. 따라서, 본 논문에서 제안한 회복 알고리즘을 이용한 스케줄링 상태 분석기는 기존의 방법에 비해 보다 빠른 시간에 효율적으로 스케줄링 가능성 분석을 판단할 수 있음을 알 수 있다.

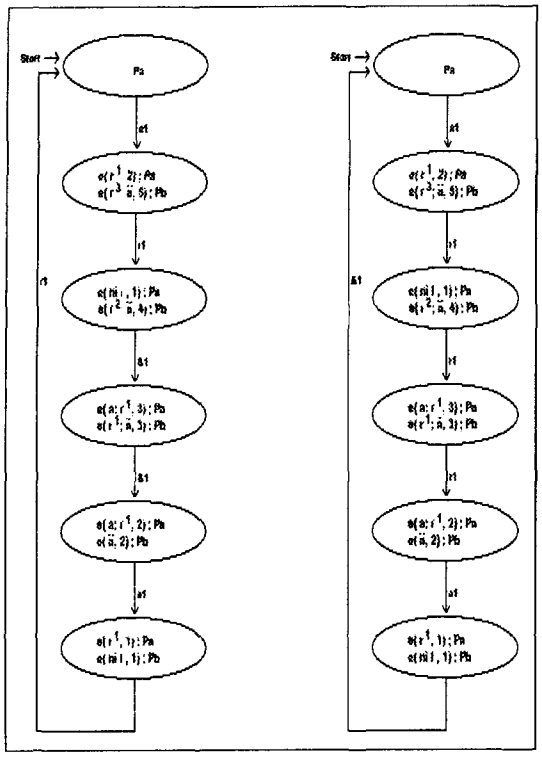

그림 10. 스케줄불가능한 프로세스 (2)에 대한 회복 알고리즘의 적용 결과
Fig. 10 Result of retreival algorithm for unschedulable processes(2)

V. 결 론

본 논문에서는 스케줄가능성 분석 방법을 이용하여 스케줄이 불가능한 프로세스를 스케줄이 가능하도록 하는 스케줄 회복 알고리즘을 제안하고, Visual C++6.0으로 스케줄가능성 분석 결과를 시각적으로 표시하는 스케줄링 상태 분석기를 구현하였다.

이 상태 분석기를 이용하면, 발생가능한 모든 프로세스의 상태 공간을 탐색하는 방법을 사용하기 때문에 시간, 공간적 복잡도가 높은 Fredette 등의 스케줄가능성 분석기들과는 달리 상태 공간의 수를 적게 생성하고, 보다 빠른 단위 시간내에 스케줄가능을 판별할 수 있다.

따라서 이 스케줄링 상태 분석기는 실시간 프로세스의 스케줄가능성 판단에 효과적으로 응용될 것이라고 생각되며, 향후의 연구 분야는 분산 실시간 시스템에 적용하는 방법을 구현한다.

참고문헌

[1] G. Bruno, A. Castella, I. Pavesio and M. P. Pescarmona, "A New Petri Net Based Formalism for Specification, Design and Analysis of Real-Time Systems", Proceeding Real-Time Systems Symposium, pp. 294-301, 1993.

[2] C. L. Liu and J. Layland, "Scheduling Algorithm for Multiprogramming in a Hard Real-Time-Environment", JACM Vol20, No1, pp.46-61, Jan.1973.

[3] 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.

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

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

[6] 박홍복, 김춘배, "분산 실시간 프로세스의 스케줄 가능성 분석 및 구현", 한국해양정보통신학회 논문지, Vol.3, No.1, pp. 209-221, 1999.

박 홍 복(Hung-Bog Park)
1982년 경북대학교 컴퓨터공학(공학사)
1984년 경북대학교 대학원 컴퓨터공학(공학석사)
1995년 인하대학교 대학원 전자계산학(이학박사)
1984년~1995년 동명대학 전자계산학과 부교수
1996년~현재 부경대하교 전자계산학과 부교수
※주관심분야: 실시간 시스템, 프로그래밍 언어 및 컴파일러, 멀티미디어 시스템, 객체지향 프로그래밍