ACSR을 이용한 실시간 운영체제의 Time Partitioning 분석[1)]

남기혁[0*], 방기석[*], 권기춘[**], 최진영[*]

고려대학교 컴퓨터학과[*]
{khnam[0], kbang, choi}@formal.korea.ar.kr[*]

원자력연구소[**]
kckwon@kaeri.re.kr[**]

Korea University

Korea Atomic Energy Research Institute

Ki-Hyuk Nam[0*], Ki-Seok Bang[*], Jin-Young Choi[*]
Dept. of Computer Science and Engineering, Korea University[*]

Ki-Chun Kwon[**]
Korea Atomic Energy Research Institute[**]

요 약

다중 쓰레드를 지원하는 실시간 운영체제 스케줄러의 time partitioning 속성을 프로세스 알제브라 언어인 ACSR을 이용하여 분석한다. 본 논문에서는 먼저 실시간 운영체제의 전반적인 스케줄러 동작에 대한 ACSR 모델을 작성하고, 이를 두 가지 시나리오에 대해 분석한다. 이러한 방법을 사용함으로써 실시간 시스템의 설계 및 구현 작업을 보다 쉽고 정확하게 수행할 수 있다.

1. 서 론

실시간 시스템의 설계와 구현의 정확성을 검증하는 것은 매우 중요하다. 그러나, 시스템의 복잡도로 인해 검증 및 분석 작업이 쉽지 않다. 현재까지도 실시간 시스템의 설계 및 검증에 대한 많은 연구가 진행되고 있는데, NASA에서는 IMA(Integrated Modular Avionics) 시스템용 운영체제인 Honeywell의 DEOS가 쓰레드 레벨의 time partitioning을 만족하는지를, 모델 체커 Spin과 여러 가지 프로그램 분석 기법을 적용하여 설계 및 검증한 바 있다[5]. 여기서 time partitioning이란, 하나의 쓰레드에 할당된 CPU 시간이 다른 쓰레드들의 동작으로 인해 훼손되지 않는다는 것을 의미한다. 본 논문에서는 NASA의 time partitioning 분석에서 사용된 모델 및 시나리오를 토대로 ACSR의 관점에서 분석한다.

2. ACSR

ACSR[1,2,3]은 CCS[4]를 기반으로 이산 시간과 공유자원, 우선 순위 등의 개념이 추가된 프로세스 알제브라 언어이다. ACSR에서는 프로세스의 행위를 1 시간 단위를 소모하는 timed action과 시간 소모없이 즉시 발생하는 instantaneous event으로 구분한다. 또한, CCS와 마찬가지로 ACSR의 동치관계(bisimulation)에 대한 complete axiomatization을 제공하기 때문에 equational law를 이용하여 동치관계를 증명할 수 있다. ACSR의 문법은 [그림1]과 같이 BNF로 표현할 수 있다. [그림2]는 본 논문에서 스케줄러 분석에 사용되는 주요 law들을 보여주고 있다.

$$P ::= \mathrm{NIL} \mid A:P \mid e.P \mid P+P \mid P \| P \mid$$
$$P \triangle_t^b (P, P, P) \mid [P]_I \mid P \backslash F \mid rec\ X.P \mid X$$

[그림1. ACSR Syntax]

Choice(5) $A_1 : P_1 + A_2 : P_2 = A_2 : P_2$ if $A_1 \prec A_2$

Choice(6) $(a_1, n_1).P_1 + (a_2, n_2).P_2 = (a_2, n_2).P_2$ if $(a_1, n_1) \prec (a_2, n_2)$

Choice(7) $A : P + (\tau, n).Q = (\tau, n).Q$ if $n > 0$

Par(2) $(A : P) \| \mathrm{NIL} = \mathrm{NIL}$

Par(3) $(a, n).P \| \mathrm{NIL} = (a, n).(P \| \mathrm{NIL})$

Par(6)
$$(\sum_{i \in I} A_i : P_i + \sum_{j \in J} (a_j, n_j).Q_j) \| (\sum_{k \in K} B_k : R_k + \sum_{l \in L} (b_l, m_l).S_l)$$
$$= \left[\begin{array}{l} \sum_{i \in I, j \in J, A_i \asymp a(B_j) = \emptyset} (A_i \cup B_j) : (P_i \| Q_j) \\ + \sum_{j \in J} (a_j, n_j).(P_j \| (\sum_{k \in K} B_k : R_k + \sum_{l \in L} (b_l, m_l).S_l)) \\ + \sum_{l \in L} (b_l, m_l).((\sum_{i \in I} A_i : P_i + \sum_{j \in J} (a_j, n_j).Q_j) \| S_l) \\ + \sum_{j \in J, l \in L, a_j = \bar{b}_l} (\tau, n_j + m_l).(Q_j \| S_l) \end{array} \right.$$

[그림2. ACSR Law]

---

1) 본 연구는 " 원자력 연구개발 중장기사업인 원전계측제어 사업단" 에 의해 지원되었음.

3. Rate Monotonic 스케줄러 모델링 및 분석

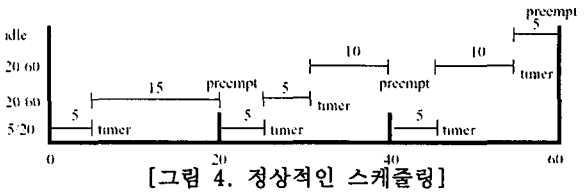
3.1. ACSR 모델

```
-------------------------------------------------------------------
System = [(Dispatch||Threads)₩{s[1-5],d[1-5],delete}]{cpu}
Dispatch = (D1 || D2 || D3 || D4 || D5)
Threads = (T1 || T2 || T3 || T4)
-------------------------------------------------------------------
D1 = ('s[1],1).(((d2,1).{})^o:D1
D2 = ('s[2],1).(((d2,1).{})^p:D2)
D3 = ('s[3],1).(((d3,1).{})^q:D3)
D4 = ('s[4],1).(((d4,1).{})^r:D4 + (delete,1).Nil)
D5 = ('s[5],1).(((d5,1).{})^r:D5)
-------------------------------------------------------------------
T1 = (s[1],1).T1[0]
T2 = (s[2],2).T2[0]
T3 = (s[3],3).T3[0]
T4 = (s[4],4).T4[0]
T5 = (s[5],4).T5[0]
-------------------------------------------------------------------
T1[0] = ('d1,1).{(cpu,1)}:('idle,1).T1[0]
T2[i] = ('d2,1).{}:T2[i]                    ( 0<=i<l )
      + ('d2,2).('d4,2).('d3,2).('d1,2).{(cpu,2)}:T2[i+1]
T2[l] = ('d2,1).{}:T2[l] + T2
T3[i] = ('d3,1).{}:T3[i]                    ( 0<=i<m )
      + ('d3,3).('d4,3).('d2,3).('d1,3).{(cpu,3)}:T3[i+1]
      + ('delete,3).T5
T3[m] = ('d3,1).{}:T3[k] + T3 + ('delete,7).T5
T4[i] = ('d4,1).{}:T4[i]                    ( 0<=i<n )
      + ('d4,4).('d3,4).('d2,4).('d1,4).{(cpu,4)}:T4[i+1]
T4[n] = ('d4,1).{}:T4[n] + T4
T5[i] = ('d5,1).{}:T5[i]                    ( 0<=i<k )
      + ('d5,4).('d2,4).('d1,4).{(cpu,4)}:T5[i+1]
T5[k] = ('d5,1).{}:T5[k] + T5               ( k=|(m/q)*r|+n )
-------------------------------------------------------------------
```

[그림 3. RM 스케줄러의 ACSR 모델]

System은 Dispatch와 Threads라는 컴포넌트로 구성되어 있으며, s[1]~s[5], d[1]~d[5], delete라는 이벤트에 대해 동기화하여 동작하고, cpu라는 자원을 독점한다. Threads는 T1, T2, T3, T4, T5가 있으며, 각각 idle, user1, user2, main, user2의 budget을 넘겨받은 main 쓰레드를 나타낸다. T2와 T3의 cpu budget은 20/60(cpu 시간/주기)이고, T4는 5/20이다. idle 쓰레드는 우선 순위가 가장 낮으며, 다른 쓰레드가 실행될 수 없을 때 동작한다. T2와 T3는 T4에서 동적으로 생성 또는 삭제될 수 있지만, 여기서는 시작과 동시에 s[1]~s[5]와 's[1]~'s[5] 이벤트의 동기화를 통해 T1, T2, T3, T4가 모두 초기화된다고 가정한다. T3가 실행중에 delete라는 이벤트를 발생시켜 종료할 경우에는, T3의 CPU budget을 T4에게 넘겨주고, T4 대신 T5가 실행된다.

3.2. 시나리오 1 : 정상적인 RM 스케줄링 분석



[그림 4. 정상적인 스케줄링]

여기서는 System의 o, p, q, r이 각각 60ms, 60ms, 60ms, 20ms으로, l, m, n이 각각 20ms, 20ms, 5ms로 할당된 경우를 살펴본다. 또한, ACSR의 1시간 단위는 1ms라고 가정한다. System의 Dispatch와 Threads는 할당된 주기와 cpu 시간이 제대로 맞을 경우에만 다음 단계로 계속 진행할 수 있게 작성되었다. 따라서 스케줄링이 제대로 되지 않은 경우에는 데드락이 발생하게 된다.

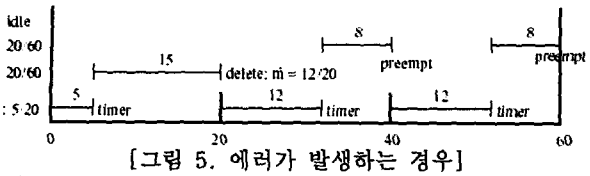
ACSR로 작성한 스케줄러는 다음과 같은 단계를 거쳐 진행한다. 각 단계는 ACSR의 Law들을 적용한 것이다.

```
Sys = (t,5).(t,4).(t,3).(t,2).
      ( P^60:D1 || Q^60:D2 || R^60:D3 || S^20:D4 || D5 ||
        T1[0] || T2[0] || T3[0] || T4[0] )
// Par3, Choice6 law 적용
// P=((d1,1).{}), Q=((d2,1).{}), R=((d3,1).{}), S=((d4,1).{})
                    ...
    = Init.((t,5).(t,5).(t,5).(t,5).{cpu,4})^5:
      ( P^55:D1 || Q^55:D2 || R^55:D3 || S^15:D4 || D5 ||
        T1[0] || T2[0] || T3[0] || T4[5] )
// Par3, Choice5, Choice6 적용, T4가 5ms 동안 진행
// Init == (t,5).(t,4).(t,3).(t,2)
                    ...
    = Init.((t,5)^4.{cpu,4})^5:((t,4)^4.{cpu,3})^15:
      ( P^40:D1 || Q^40:D2 || R^40:D3 || D4 || D5 ||
        T1[0] || T2[0] || T3[15] || T4 )
// Par3, Choice5, Choice6 적용,
```

여기서 delete와 s[4] 이벤트 모두 발생할 수 있는데, 쓰레드가 삭제되지 않는 경우를 살펴보기 위해 s[4] 이벤트가 발생하여 다시 T4가 진행한다. 이런 식으로 ACSR law를 반복적으로 적용하면 [그림 4]와 같이 동작한다. T4가 5ms 수행한 후에는, T3는 이미 15ms만큼의 cpu 시간을 소모했기 때문에, 남은 5ms만 진행하고 T2에게 cpu를 넘겨준다. T3의 주기인 60ms가 도달하기 전까지는 T4와 T2가 서로 번갈아가면서 실행되다가 T2의 cpu 시간을 다 소모하면 남은 시간은 idle이 수행된다.

이처럼 쓰레드가 중간에 종료되지 않으면 데드락 없이 계속 진행할 수 있다. 변수 p, q, r, l, m, n의 값을 대입하여 다른 주기 및 cpu 시간에 대해서도 스케줄 가능성을 검사할 수 있다.

3.3. 시나리오 2 : T3가 20ms 후에 삭제되는 경우



[그림 5. 에러가 발생하는 경우]

이번에는 [그림 5]와 같이, s[4] 대신 delete 이벤트가 발생하여 T3가 종료하는 경우를 살펴본다. 처음 단계는 앞 절에서 설명한 세 번째 표현식까지 똑같이 진행한다.

```
...
= Init.((t,5)^4.{cpu,4})^5:((t,4)^4.{cpu,3})^15:
  ( P^40:D1 || Q^40:D2 || R^40:D3 || D4 || D5 ||
    T1[0] || T2[0] || T3[15] || T4 )
// Par3, Choice5, Choice6 law 적용,
```

여기서는 시나리오 1의 경우와 달리, T3에서 delete 이벤트를 발생하여 D4의 동작을 멈추게하고 T5로 전이한다. 여기서 T3은 곧바로 T5로 진행하였기 때문에 T3의 초기 상태인 (s[3],3).T3[0]로 갈 수 없으며, 따라서 T3에 대응되는 D3도 진행할 수 없게 된다. 또한 T5로 넘어가면서 T3의 cpu budget이 T4의 cpu budget과 합쳐져서 T5의 cpu budget k는 12(|(m/q)*r|+n)ms가 된다.

```
= Init.((t,5)^4.{cpu,4})^5:((t,4)^4.{cpu,3})^15:(t,4).
  ( P^40:D1 || Q^40:D2 || R^40:D3 || Nil || D5 ||
    T1[0] || T2[0] || T5 || T4 )
// Par3, Choice5, Choice6 law 적용,
```

이 시점에서 D5 || T5의 우선 순위가 가장 높기 때문에[('d5,4)와 (d5,1)], T5가 12ms만큼 계속 진행한다.

```
...
= Init.((t,5)^4.{cpu,4})^5:((t,4)^4.{cpu,3})^15:(t,4).
  ((t,5)^4.{cpu,4})^12:
  ( P^28:D1 || Q^28:D2 || R^28:D3 || Nil || S^8:D5 ||
    T1[0] || T2[0] || T5 || T4 )
// Par3, Choice5, Choice6 law 적용,
```

그런 다음, T5의 남은 주기(8ms)만큼 T2가 진행되고, 그 다음 주기 20ms 동안 한번 더 T5와 T2가 각각 12ms와 8ms만큼 번갈아 진행하게 된다.

```
...
= Init.((t,5)^4.{cpu,4})^5:((t,4)^4.{cpu,3})^15:(t,4).
  (((t,5)^4.{cpu,4})^12:((t,3)^4.{cpu,2})^8)^2:
  ( D1 || D2 || D3 || Nil || D5 ||
    T1[0] || T2[16] || T5 || T4 )
// Par3, Choice5, Choice6 law 적용
```

그런데 여기서 전체 주기가 한번 끝나고 새로운 주기를 시작해야 하는데, T2는 전체 cpu 시간 20ms 중 16ms만 진행된 상태에서 다시 우선 순위를 따라 T5를 진행시키다보면,

```
('d5,4).('d2,4).('d1,4).{(cpu,4)}:T5[1]
```

중에서 ('d5,4)만 진행하고 ('d2,4)는 D2가 ('s[2],1)에 의해 막혀있기 때문에 더 이상 진행하지 못하고 데드락을 발생하게 된다. 따라서, 이 경우에는 time partitioning을 보장해주지 못하게 된다.

4. 결론 및 향후 연구 계획

본 논문에서는 실시간 운영체제의 Rate Monotonic 스케줄러의 time partitioning 속성을 프로세스 알제브라인 ACSR로 분석하였다. 기본적인 RM 스케줄링 분석 뿐만 아니라, 실행 시간에 쓰레드가 종료할 수 있는 경우에 대해서도 검증하였다. 실시간 시스템의 개발에 ACSR을 이용하여 명세하고 검증함으로써, 미묘한 에러를 좀 더 쉽게 찾아낼 수 있다. 향후 연구 계획으로는 모델의 상세한 명세를 통해 다양한 속성들을 분석하고자 한다.

5. 참고 문헌

[1] Insup Lee, Hanne Ben-Abdallah, Jin-Young Choi, A Process Algebraic Method for the Specification and Analysis of Real-Time Systems, Formal Methods for Real-Time Computing, John Wiley & Sons Ltd, 1996

[2] Patrice Brmond-Grgoire, Jin-Young Choi, and Insup Lee, A Complete Axiomatization of Finite-state ACSR Processes, Information and Computation 138, 124-159, 1997

[3] Duncan Clarke, Insup Lee, Hong-liang Xie, VERSA: A Tool for the Specification and Analysis of Resource-Bound Real-Time Systems, Journal of Computer Software Engineering, 3(2), 1995

[4] Robin Milner, Communication and Concurrency, Prentice Hall, 1989

[5] John Penix, Willem Visser, Eric Engstrom, Aaron Larson, Nicholas Weininger, Verification of Time Partitioning in the DEOS Scheduler Kernel, International Conference on Software Engineering, 2000