한국컴퓨터정보학회 하계학술대회 논문집 제30권 제2호 (2022. 7)

혼합 중요도 시스템의 주기 변환과 스케줄링 오버헤드간의 트레이드오프 관계 분석

윤상운[O], 임지섭[*], 강경태[*]
[O]한양대학교 인공지능융합학과 바이오인공지능융합전공,
[*]한양대학교 인공지능융합학과 바이오인공지능융합전공
e-mail: {swyun, jseoup, ktkang}@hanyang.ac.kr[O*]

Analysis of Trade-off between Period Transformation and Scheduling Overhead in Mixed-Criticality System

Sangwoon Yun[O], Jiseoup Lim[*], Kyungtae Kang[*]
[O]Dept. of Applied Artificial Intelligence, Major in Bio Artificial Intelligence, Hanyang University,
[*]Dept. of Applied Artificial Intelligence, Major in Bio Artificial Intelligence, Hanyang University

● 요 약 ●

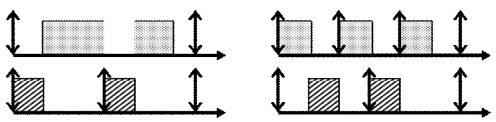
혼합 중요도(mixed criticality) 시스템은 안전에 중요한 기능을 우선시하도록 하는 추가적인 안전 요구 사항이 존재한다. 그러나 기존 실시간 시스템의 설계로는 이를 만족하지 못하며, 높은 중요도 태스크가 다른 낮은 중요도 태스크로부터 간섭을 받아 데드라인 미스와 같은 문제를 일으키는 중요도 역전(criticality inversion) 문제가 발생할 수 있다. 이러한 중요도 역전 문제를 해결하기 위해 주기 변환(period transformation) 기법을 사용할 수 있지만, 스케줄링 오버헤드의 증가로 인해 오히려 전반적인 태스크의 응답시간이 증가하는 또 다른 문제가 발생하게 된다. 본 논문에서는 주기 변환과 스케줄링 오버헤드 간의 트레이드오프 관계를 설명하고, 실시간 리눅스 시스템에서 해당 문제점을 재연한 후 주기 변환의 적정선을 분석하고자 실험을 진행하였다. 그 결과, 중요도 역전 문제를 해결하기 위한 주기 변환을 그대로 적용할 경우, 문맥 교환이 48.7% 증가 및 스케줄링 오버헤드가 28.7% 증가로 인해 전반적인 응답시간이 증가하여 데드라인 미스가 다수 발생하는 결과를 확인할 수 있었다.

키워드: 실시간 시스템(real-time system), 혼합 중요도 시스템(mixed-criticality system), 중요도 역전(criticality inversion), 주기 변환(period transformation)

I. Introduction

최근에는 비용, 중량 및 전력 소비를 제한하기 위해 ECU의 수를 더 이상 늘리지 않는 추세이며, 이는 대부분 고도로 통합된 다기능 단일 ECU로 이어진다[1, 2]. 즉, 단일 ECU에 다양한 중요도(criticality) 수준을 가진 기능이 공존하는 혼합 중요도(MC, mixed criticality) 시스템은 점점 보편화 될 것이며, 이는 추가적인 안전 요구 사항을 충족해야 한다. 이로 인해 혼합 중요도 시스템은 더 이상 기존 설계 패턴으로 안정적으로 처리할 수 없는 요구 사항을 제시하게 된다.

본 연구에서 혼합 중요도 안전 요구 사항에 위배되는 중요도 역전(criticality inversion)이 실제 환경에서 유발하는 문제를 살펴본다. 또한, 중요도 역전 문제를 해결하기 위한 방법인 주기 변환(period transformation)과 이로 인해 발생되는 스케줄링 오버헤드를 측정하고, 주기 변환의 문제점을 확인한다.



(a) RM Scheduling (b) Period Transformation

Fig 1. Summary

II. Background

혼합 중요도 시스템에서는 다음과 같은 안전 요구 사항을 충족해야 한다[1, 2].

- 모든 상황에서 ECU의 과부하를 방지하고 모든 타이밍 요구 사항을 충족하기 위해 관련 기능에 충분한 컴퓨팅 시간을 사용할 수 있어야 한다.
- 안전 관련 기능은 방해될 수 없다.

하지만 전통적인 실시간 시스템은 낮은 중요도 태스크가 높은 중요도 태스크의 실행을 간섭하는 중요도 역전 문제에 직면하게 된다[1].

Table 1. Task Set of the Example

Tasks	Criticality	Period	Runtime	Release
T1	H	12	6	0
T2	M	10	2	1
T3	L	6	1	3

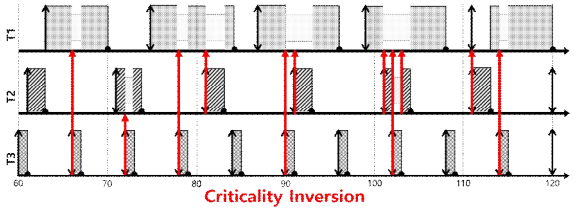


Fig 2. Gantt Chart of RM Scheduling

비율 단조(RM, Rate-Monotonic) 스케줄링[3] 기법은 태스크의 우선순위를 수행 주기(period)가 가장 짧은 태스크에 가장 높은 우선순위를 부여하는 방법이다. 태스크 셋에 대한 스케줄링 가능성 확인은 CPU 최소 상한(LUB, least upper bound) 혹은 응답 시간 분석(response time analysis)[4, 5]을 통해 확인한다. 응답 시간 분석은 모든 태스크에 대해 Eq 1을 만족하는지 확인하는 것이다.

$$R_i = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \leq D_i \qquad (1)$$

하지만 이러한 전통적인 스케줄링 기법은 태스크의 타이밍 보장이 주된 목적이기 때문에 혼합 중요도 시스템의 안전 요구 사항을 충족하기 어렵다. 반면에, 중요도를 우선순위로 사용하는 CAPA(criticality aware priority assignment) 스케줄링 기법은 높은 중요도 태스크를 우선적으로 스케줄링하기 때문에 혼합 중요도의 안전 요구 사항을 만족할 수 있다. 하지만 고정 우선순위(Fixed Priority) 스케줄링 기법의 단점을 똑같이 가지고 있기 때문에, 태스크의 타이밍 보장이 불가능한 경우가 있다는 문제가 있다.

이러한 문제점을 해결하기 위해 주기 변환을 적용한 RM 스케줄링 기법을 사용하는 방법이 있다[6, 7]. RM 스케줄러 정책에서 높은 우선순위를 할당하기 위해 높은 중요도 태스크를 짧은 주기로 여러 번 나누어 스케줄링 하는 방법이다. 하지만 이 과정에서 더 많은 태스크의 선점(preemption)이 일어나기 때문에 문맥 교환(context switch)로 인한 스케줄링 오버헤드가 증가하게 된다. 이는 태스크의 응답시간 증가로 이어지며, 데드라인 미스와 같은 또 다른 문제를 유발할 수 있다.

Table 2. Task Set of the Period Transformation

Tasks	Criticality	Period	Runtime	Release
T1′	H	4	2	0
T2′	M	5	1	1
T3	L	6	1	3

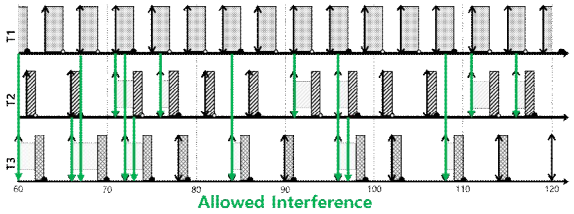


Fig. 3. Gantt Chart of Period Transformation

III. Experiments

태스크 셋 Table 1에 대한 응답 시간 분석을 통해 스케줄링 가능성 분석을 하면, Eq 2와 같이 모든 태스크의 응답 시간이 데드라인 이내이기 때문에 스케줄링이 가능함을 알 수 있다.

$$\begin{aligned} R_3 &= 1 < 6 \\ R_2 &= 3 < 10 \\ R_1 &= 10 < 12 \end{aligned} \qquad (2)$$

태스크 셋 Table 1을 RM 스케줄러를 통해 스케줄링하게 되면 Fig 2처럼 모든 태스크는 데드라인을 충족한다. 그러나 태스크 T3가 더 높은 중요도 태스크 T1과 T2의 실행에 간섭을 하는 중요도 역전 문제가 발생하게 되며, 발생 지점들을 붉은 색 화살표를 통해 표시하였다.

중요도 역전 문제를 해결하기 위해, Table 2와 같이 주기 변환을 통해 태스크 T1과 T2의 우선순위를 높이는 과정을 수행하였다. 태스크 T1을 4 ms의 주기를 가진 3개의 태스크로 분할하고, 태스크 T2를 5 ms의 주기를 가진 2개의 태스크로 분할하였다. 변환된 태스크 셋으로 RM 스케줄링을 하게 되면 Fig 3처럼 중요도 역전 문제가 해결되며, 모든 태스크에 대해 타이밍 보장이 가능하다. 하지만 문맥 교환 횟수가 간트 차트 상에서 36 회에서 45 회로 25% 증가하였다.

본 연구는 실제 환경에서의 스케줄링 오버헤드를 측정하기 위해 Intel Core i7-10700 CPU의 Ubuntu 20.04 데스크탑에서 실험을 진행하였다. RM 스케줄러는 sched_deadline 문서[8]를 참고하여 SCHED_FIFO에서 clock_gettime() 함수와 usleep() 함수를 사용하여 구현하였다.

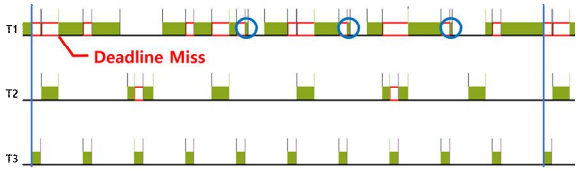


Fig 4. Actual RM Scheduling Example

해당 실험에서는 태스크 사이의 문맥 교환 오버헤드를 측정하기 위해 ftrace[9]의 function_graph 트레이서를 사용하며 enqueue_task와 dequeue_task 함수를 추적하였다.

IV. Results

Fig 4와 Fig 5는 각각 Table 1의 태스크 셋과 주기 변환을 수행한 Table 2의 태스크 셋을 RM 스케줄러로 실제 환경에 적용했을 경우의 예시이다. 초주기 (hyper period) 당 실제 문맥 교환은 태스크 셋 1이 총 39회, 주기 변환 후 총 58회로 간트 차트의 예상 횟수보다 각각 8.3%, 28.9% 증가하였다. 그래서 태스크 셋 Table 1에서 주기 변환을 수행 후 문맥 교환 횟수가 48.7% 증가하였다.

초주기당 측정된 전체 스케줄링 오버헤드는 각각 24.668 μs와 31.747 μs로 주기 변환 수행 이후 28.7% 증가하였다. 중요도 역전 문제와 스케줄링 오버헤드로 인해 Fig 4와 같이 높은 중요도 태스크 T1이 데드라인 미스가 발생하였다. 중요도 역전 문제를 해결하기 위한 주기 변환은 Fig 5와 같이 초주기 당 데드라인 미스가 3회 발생하여 타이밍 보장이 불가능함을 확인 할 수 있다.

V. Conclusions

해당 논문에서는 혼합 중요도 시스템에서 중요도 역전으로 인한 문제점을 실제 실시간 리눅스 환경에 적용하여 확인 할 수 있었다. 또한, 주기 변환을 통해 중요도 역전 문제 해결을 시도하였지만, 이로 인해 문맥 교환 횟수가 48.7% 증가하고, 스케줄링 오버헤드가 28.7% 증가함을 확인하였다. 그 결과 전체 시스템의 응답 시간이 증가하여 데드라인 미스가 다수 발생하게 되고, 타이밍 보장과 같은 혼합 중요도 시스템의 안전 요구사항을 충족하지 못하는 결과를 초래함을 확인할 수 있다.

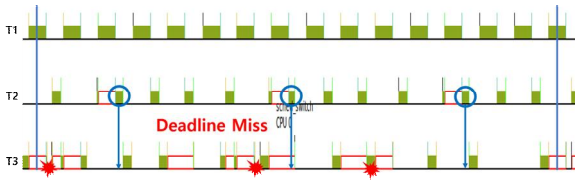


Fig. 5. Actual Period Transformation Example

ACKNOWLEDGMENT

이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.2020-0-01343, 인공지능융합연구센터 지원(한양대학교 ERICA))과2022년도 정부(산업통상자원부)의 재원으로 한국산업기술진흥원의 지원(P0012744, 2022년 산업혁신인재성장지원사업)을 받아 수행된 연구임

REFERENCES

[1] Schmidt, Karsten, et al. "Design patterns for highly integrated ECUs with various ASIL levels." *ATZelektronik worldwide* 7.1 (2012): 22-27.

[2] Ficek, Christoph, Nico Feiertag, and Kai Richter. "Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems." *Embedded Real Time Software and Systems (ERTS2012).* 2012.

[3] Liu, Chung Laung, and James W. Layland. “Scheduling algorithms for multiprogramming in a hard-real-time environment.” *Journal of the ACM (JACM)* 20.1 (1973): 46-61.

[4] Audsley, Neil, et al. "Applying new scheduling theory to static priority pre-emptive scheduling." *Software engineering journal* 8.5 (1993): 284-292.

[5] S. K. Baruah, A. Burns and R. I. Davis, "Response-Time Analysis for Mixed Criticality Systems," *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, pp. 34-43

[6] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239-243

[7] Sha, L. "Solutions for some practical problems in prioritized preemptive scheduling." *Proc. 7th IEEE Real-Time Systems Symposium, 1986.* IEEE Computer Society Press, 1986.

[8] "Deadline task scheduling." *kernel.org*, 2018, https://www.kernel.org/doc/Documentation/scheduler/sched-deadline.txt. accessed 16. Jun 2022.

[9] Steven Rostedt, "ftrace - function tracer." *kernel.org*, 2008, https://www.kernel.org/doc/Documentation/trace/ftrace.txt. accessed 16. Jun 2022.