Journal of Korea Multimedia Society Vol. 14, No. 1, January 2011(pp. 76-84)

임베디드 시스템에서 실시간성과 결함허용을 보장하는 스케줄러 설계

전태건[†], 김창수[††]

요 약

임베디드 시스템은 적용되는 분야에 따라 태스크의 완료에 대한 실시간성이 보장되어야 하는 경우가 있다. 또한 실시간성을 제공하는 시스템에서는 다양한 이유로 인해 태스크의 결함이 발생할 수 있다. 그러므로 본 논문에서는 단일 프로세서를 가지는 임베디드 시스템에서 주기적 태스크 집합의 마감시간을 보장하면서 결함이 발생한 태스크의 결함을 허용하는 태스크 스케줄러를 설계한다. 임베디드 시스템에서 실시간성을 제공하기 위해서 태스크를 주기적 및 비주기적 특성으로 분류한 후, 주기적 태스크는 고정 우선순위 실시간 태스크 스케줄링 기법에서 최적의 알고리즘으로 알려진 RMS(Rate Monotonic Scheduling) 기법을 적용하여 실행한다. 주기적 태스크들의 실행 시간을 분석한 후, 결정되는 잉여 시간을 관리함으로써 비주기적 태스크들의 실행을 보장한다. 또한 일시적인 태스크의 단일 결함을 허용하기 위한 결함 허용 기법을 제공한다. 이를 위해 백업 시간을 관리하고 태스크에서 결함이 발생할 경우, 결함이 발생한 태스크를 재실행함으로써 복구 작업을 수행한다.

Design of Scheduler Considering Real-Time Characteristic and Fault-Tolerant in Embedded System

Taegun Jeon[†], Changsoo Kim[††]

ABSTRACT

Embedded Systems need to ensure real-time of the task response time depending on the applied fields of it. And task could be faulty due to various reasons in real time systems. Therefore in this paper, we design a task scheduler that guarantees deadlines of periodic tasks and considers a fault tolerance of defective task in embedded system with a single processor. In order to provide real-time, we classify tasks with periodic/aperiodic tasks and applies RMS(Rate Monotonic Scheduling) method to schedule periodic tasks and can guarantees execution of aperiodic tasks by managing surplus times obtained after analyzing the execution time of periodic tasks. In order to provide fault tolerance, we manage backup times and reexecute a fault task to restore it's conditions.

Key words: Embedded System(임베디드 시스템), Task Scheduling(태스크 스케줄링), Real-Time(실시간), Fault-Tolerance(결함허용)

※ 교신저자(Corresponding Author) : 김창수, 주소 : 부산광역시 남구 대연 3동 599-1 부경대학교 대연캠퍼스 1호관 302호실(608-737), 전화 : 051)629-6245, FAX : 051)629-6230, E-mail : cskim@pknu.ac.kr
접수일 : 2010년 8월 10일, 수정일 : 2010년 10월 15일
완료일 : 2010년 10월 28일
[†] 준회원, 부경대학교 IT융합응용공학과
(E-mail : nicejtg@naver.com)
[††] 종신회원, 부경대학교 IT융합응용공학과

1. 서 론

최근 임베디드 시스템은 하드웨어 기술의 발달로 매우 집적화된 마이크로프로세서가 등장하면서 컴퓨팅 파워가 늘어나 여러 기능을 동시에 수행할 수 있게 되었으며 임베디드 시스템이 갖추어야 하는 사용자의 요구 또한 다양하게 증가하였다. 그러므로 임베디드 시스템에서 처리해야할 작업들이 많아지고 처리해야할 작업의 복잡성 또한 증가하게 되었다. 즉, 임베디드 시스템에서 운영체제에 대한 필요성이 대두되었다.

임베디드 운영체제는 항공기, 우주 왕복선, 산업 프로세스 제어와 스마트 자동차 등과 같은 작업 시간 제약을 만족해야하는 경우가 있다[1-4]. 그러므로 임베디드 운영체제에서 실시간성을 보장할 필요가 있으며 임베디드 실시간 태스크 스케줄러는 효율적인 태스크 관리를 위해 실시간 태스크들을 주기적 태스크와 비주기적 태스크로 구분하여 스케줄링 할 필요가 있다.

임베디드 운영체제가 적용되는 시스템 중에는 수행 중에 발생하는 태스크의 결함이 전체 시스템에 치명적인 결과를 초래할 수 있으며 결함이 발생한 시스템에 대한 즉각적인 대응이 어려운 경우가 있다. 그러므로 태스크 결함으로 인해 발생할 수 있는 문제점들을 해결하여 시스템의 정상적인 동작을 보장하기 위해서는 결함 허용 기법이 제공되어야 한다. 이를 위해 기존의 결함 허용 기법들을 응용하여 이를 임베디드 실시간 시스템에 적용하기 위한 연구가 필요하다.

본 논문에서는 단일 프로세서를 가지는 임베디드 실시간 시스템에서 태스크의 특성에 따라 태스크를 스케줄링 할 수 있는 스케줄러를 설계한다. 주기적 태스크의 마감 시간을 보장하고 비주기적 태스크를 실행하기 위한 잉여 시간을 관리한다. 또한 설계된 스케줄러는 일시적인 태스크 결함 허용을 위한 결함 허용 기법을 제공한다. 이를 위해 태스크에서 일시적인 결함이 발생할 경우, 백업 시간을 이용하여 결함이 발생한 태스크를 재실행함으로써 복구 작업을 수행한다.

본 논문의 구성은 다음과 같다. 2장에서 실시간 및 결함 허용 태스크 스케줄링 알고리즘에 대해 기술한다. 3장에서는 본 논문에서 제안하는 태스크 스케줄러를 설계하고 스케줄러 구성 요소에 대해 기술한다. 4장에서는 결론을 맺는다.

2. 관련 연구

2.1 실시간 스케줄링

고정우선순위를 부여하는 대표적인 태스크 스케줄링 방법은 Liu와 Layland에 의해 제안된 RMS (Rate Monotonic Scheduling)기법이다[5]. RMS는 고정 우선순위 기반 선점형 스케줄링 방식 하에서는 최적의 스케줄링 방법임을 증명하였으며 태스크 집합의 실행 가능 여부를 검사하여 태스크의 실행 가능성을 판단하였다. 전체 처리기 이용률이 0.69보다 작은 태스크 집합은 RM 스케줄링으로 항상 실행 가능함을 증명하였다. 그러나 [5]에서 제시한 처리기 이용률을 만족하지 않는 태스크 집합이라 할지라도 RMS에 의해 실행이 가능함을 Lehoczky[6] 등이 보여 주었다. Lehoczky[6] 등은 Liu와 Layland[5]의 실행 가능성 분석 방법의 부정확성을 보완하여 실행 가능성 검사를 위한 필요충분조건을 제시하였다.

[7]은 비주기적 태스크 스케줄링을 위해서 슬랙 스틸링(slack stealing) 알고리즘을 제안하였으며 비주기적 태스크 처리를 위한 passive 태스크를 생성하고 주기적 태스크의 마감 시간을 보장함과 동시에 비주기 태스크의 응답시간 예측과 실행 시간을 보장하기 위해서 비주기적 태스크에 할당 가능한 처리기 시간을 계산한다. [8]은 고정 우선순위의 슬랙 스틸링 개념을 동적 우선순위 시스템으로 확장하였다.

정경훈[9]와 김병훈[10] 등은 멀티 태스크 기반의 확장성과 주기 및 비주기 태스크 관리 기법을 효율적으로 제공할 수 있는 실시간 센서 노드 플랫폼을 제시하였다. 실시간성을 제공하기 위하여 주기적 태스크의 마감시한을 보장하고 비주기적 태스크의 응답시간을 최소화하는 방법을 제공하였으나 태스크 실행 중에 발생할 수 있는 결함을 복구할 수 있는 방법은 제공하지 못하고 있다.

김희헌[11] 등은 EDF 스케줄링 알고리즘을 사용하는 단일처리기 시스템에서 주기 작업의 단위 수행시간마다 확보할 수 있는 잉여 여유시간을 이용해 온라인으로 비주기 태스크에 마감시간을 부여하는 알고리즘이다. 주기 및 비주기 태스크들이 처리기의 이용률을 모두 이용할 수 있게 하며 주어진 주기 태

스크들의 마감시간을 보장하였다.

2.2 결함 허용

Gosh 등은 태스크를 스케줄링하는 동안에 발생하는 결함을 허용하는 기법을 제안하였다[12]. 이 기법은 같은 처리기 상에서 결함 태스크의 재실행을 통해 일시적인 결함을 허용하는 RMS이며 두 개의 태스크 요청들 사이에 결함이 발생한 태스크의 재실행에 이용될 수 있는 슬랙 타임(slack time)을 확보함으로써 결함 허용이 가능하도록 하였다.

Pandya와 Malek[13]은 RM 기법으로 스케줄되는 주기적 태스크 집합의 실행 가능성을 분석하고 단일 결함을 허용하는 방법을 제공하였다. [13]은 결함 발생 이후의 태스크들 중 완료하지 않은 모든 태스크들을 재실행함으로써 복구 작업을 수행하였다. 또한, 처리기 이용률이 0.5보다 작거나 같으면 하나의 일시적 결함발생에 대해 어떠한 태스크도 마감시한을 넘기지 않는다는 것을 증명하였다.

Gosh[14] 등은 primary/backup 방법을 사용하여 비선점, 독립적이며 비주기적 실시간 태스크들의 결함 허용 스케줄링을 연구하였다. 태스크가 시스템에 도착하면 primary는 가능한 한 빨리 스케줄이 되며 backup은 primary 스케줄링 시간보다 늦고 태스크의 마감시한 보다 빠른 시간에 스케줄된다. Primary가 성공적으로 수행을 종료하면, 그에 대응하는 스케줄링된 backup은 할당 해제된다.

Y. S. Hong[15] 등은 분산 실시간 시스템에서 처리기의 이용률을 높이면서 주기적 태스크들을 위한 결함 허용 스케줄링 방법을 제안하였다. 태스크의 처리기 이용률에 따라 태스크를 분류하여 primary/backup 기반으로 결함이 발생한 primary 태스크의 재실행을 backup 태스크에서 재실행함으로써 결함을 복구하였다.

Ching-Chih[16] 등은 주기적 태스크 집합의 각 태스크를 primary/alternate로 분류하여 정해진 마감시간 안에 임계 태스크의 primary 또는 alternate 중에 하나가 완료됨을 보장하고 가능한한 많은 primary를 완료할 수 있도록 시도하였다. alternate 태스크를 위한 타임 슬롯을 미리 할당하기 위해서 RM 알고리즘과 같은 고정 우선순위 선점형 스케줄링 스키마를 기본 알고리즘으로 사용하며 실행시 primary 태스크를 먼저 실행하도록 한다. primary 태스크가 실행시 bug로 인해 실패하거나 충분하지 못한 처리기 시간으로 인해 실행 완료를 보장할 수 없을 때, alternate 태스크를 활성화하여 실행을 보장한다.

정경훈[17] 등은 결함허용이 가능한 임베디드 실시간 태스크 관리 메커니즘을 제안하였다. 주기적 태스크들의 마감시한과 비주기적 태스크의 실행완료를 보장할 뿐만 아니라 일시적인 결함이 발생한 태스크를 복구함으로써 태스크 결함으로 인한 시스템 고장을 방지하였다. 여유 시간을 이용하여 결함이 발생한 태스크를 재실행하는 결함 허용 방법으로 복구되는 태스크의 완료 시간을 예측하기 어렵다는 특징이 있다.

3. 결함허용 및 실시간 임베디드 태스크 스케줄러

3.1 스케줄러 개관

임베디드 실시간 운영체제에서 스케줄러를 설계할 때 태스크의 우선순위, 태스크의 주기성 및 비주기성, 태스크의 마감시간 등과 같은 태스크의 특성을 고려할 필요가 있다. 또한 시스템의 지속적인 실행을 위한 안정적인 시스템 운영 기술이 필요하다. 이를 위해 태스크의 결함이 발생함에도 불구하고 실시간성과 결함 허용성을 제공할 수 있는 기법이 필요가 있다.

본 장에서는 본 논문에서 제안하는 결함허용을 고려한 실시간 임베디드 스케줄러의 전체 구조와 구성요소에 대해 기술한다.

그림 1은 본 논문에서 제안하는 실시간 태스크 스케줄러에 대한 전체 구성도를 나타낸다.

본 논문에서 제안한 스케줄러는 실행 가능성 검사 모듈, 타임 관리 모듈, 결함 허용 모듈, 비주기적 태스크 스케줄러 모듈과 주기적 태스크 모듈로 구성되어 있다. 실행 가능성 검사 모듈은 결함이 발생한 태스크의 결함 허용을 위한 백업 시간과 주기적 태스크들의 실행 시간을 고려하여 실행 가능성 여부를 검사한다. 타임 관리 모듈은 결함 허용을 위한 백업 시간과 비주기적 태스크를 위한 잉여 시간을 관리한다. 백업 시간 할당 모듈에서 결함이 발생한 태스크를 재실행하기 위한 백업 시간을 할당하고 잉여 시간 할당 모듈에서 비주기적 태스크를 위한 타임 슬롯을 할당한다.

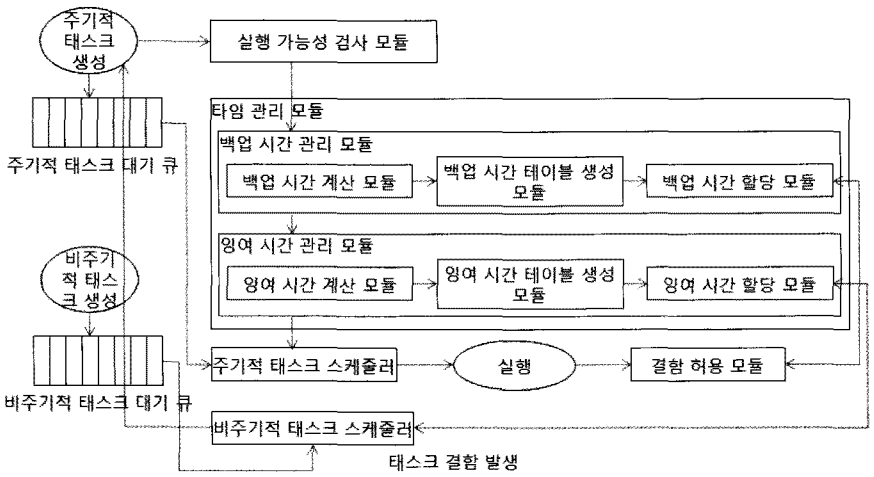

그림 1. 결함 허용 실시간 태스크 스케줄러 구성도

본 논문에서 주기적 태스크 스케줄링 알고리즘은 RMS(Rate Monotonic Scheduling)를 이용하였으며, 결함이 발생하였을 경우 결함을 복구하기 위하여 [12]에서 제시한 백업 시간을 이용한다. 또한 비주기 태스크를 위해 Slack Stealing 알고리즘을 적용하였다.

3.2 실행 가능성 검사 모듈

실행 가능성 검사 모듈은 주기 태스크의 처리기 이용률과 결함 태스크를 위한 백업 시간 이용률을 기반으로 하여 전체 태스크의 처리기 이용률을 계산한다. 주기적 태스크는 $\tau_i=(C_i, T_i)$과 같이 정의된다. 이 때, C_i는 태스크의 실행시간이고 T_i는 태스크의 실행 주기(마감 시간)를 나타낸다. 태스크의 우선순위는 RMS를 따르며 우선순위가 높을수록 낮은 태스크에 비해 먼저 수행된다.

태스크의 결함을 고려하지 않았을 경우 태스크 집합의 스케줄 가능성은 RMS를 따르며 다음 수식 (1)~(5)에 의해 결정된다.

$$t = S_i = \{k \cdot T_j \mid j=1,...,i; k=1,..., \lfloor T_i/T_j \rfloor\} \quad (1)$$

$$W_i(t) = \sum_{j=1}^{i} C_j \cdot \lceil t/T_j \rceil \quad (2)$$

$$L_i(t) = W_i(t)/t \quad (3)$$

$$L_i = \min_{\{0<t\le T_i\}} L_i(t) \quad (4)$$

$$L = \max_{\{1\le i\le n\}} L_i \quad (5)$$

$t(=S_i)$는 τ_i에 대한 스케줄링 포인트(scheduling point)이다. 또한 τ_i의 마감 시간이며 τ_i의 마감 시간 전에 τ_i보다 높은 우선순위를 가진 태스크의 도착 시간이다. 즉, 주기 T_j의 배수 중의 하나가 되며 처리기 이용률을 계산하는 기준 시간이 된다. 수식 (2)는 시간 t에 대한 태스크 τ_i가 수행완료를 위해 요구되는 처리기의 시간요구량이며 수식 (3)은 시간 t에 대한 태스크들의 처리기 이용률이다. 그리고 L_i는 수식 (3)에서 계산된 처리기 이용률 중에서 최소값을 나타내며, 추가된 태스크 집합의 실제 처리기 이용률이 된다. 수식 (5)는 수식 (4)에서 구한 처리기 이용률 중에서 최대값이며 전체 주기 태스크 집합의 처리기 이용률이다.

결함 허용 기법을 적용하기 위해서 결함이 발생한 태스크의 재실행에 필요한 추가적인 프로세서 처리 시간을 요구하게 된다. 즉, 결함이 발생한 태스크의 재실행을 위한 백업 시간이 필요하며 이 시간은 다음 수식 (6)~(7)에 의해 결정된다.

$$U_B = \max\{U_i\} \quad (6)$$

$$B_L = U_B(lT_j - kT_i) \quad (7)$$

수식 (6)은 태스크들의 처리기 이용률 중에서 최대값을 의미하며, 백업 시간을 위해 계산된 수식 (7)은 태스크들의 주기와 주기 사이에 결함 허용을 위한 백업 시간을 의미한다.

예를 들어, τ={(2,8), (3,10)}의 태스크 집합이 있을 경우, 태스크의 처리기 이용률은 각각 25%, 30% 이

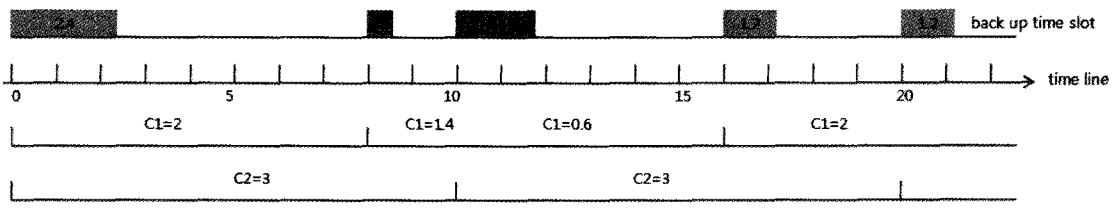

그림 2. 백업 시간 할당

므로 U_B=30%가 되며 각 태스크들 주기 사이의 백업 시간은 시간 0에서 T_1사이는 2.4, T_1과 T_2 사이는 0.6, T_2와 $2T_1$ 사이는 1.8, $2T_1$에서 $2T_2$ 사이는 1.2의 백업 시간을 할당한다. 결함 허용을 위해 각 주기마다 할당된 백업 시간은 그림 2에서 회색 박스로 표시된다.

τ_i보다 더 높은 우선순위를 가진 태스크의 재실행으로 인해 τ_i를 실행하기 위해서 추가적인 프로세서 요구(HR_i)가 필요하고 τ_i보다 같거나 낮은 우선순위를 가진 태스크의 재실행으로 인한 τ_i의 지연 시간(LR_i)이 필요하며 그 각각은 다음 수식 (8)~(9)와 같다.

$$HR_i = \max_{j=1}^{i} C_j \tag{8}$$

$$LR_i = \min(T_i \cdot U_B, \max_{j=i+1}^{n} C_j) \tag{9}$$

그러므로 결함 허용을 고려한 실시간 스케줄러의 실행 가능성은 수식 (10)~(13)에 의해 결정되며, $LR \leq 1$인 경우 태스크 집합은 실행이 가능함을 나타낸다.

$$WR_i(t) = \sum_{j=1}^{i} C_j \cdot \lceil t / T_j \rceil + \max(HR_i, LR_i) \tag{10}$$

$$LR_i(t) = WR_i(t)/t \tag{11}$$

$$LR_i = \min_{\{0 < t \leq T_i\}} LR_i(t) \tag{12}$$

$$LR = \max_{\{1 \leq i \leq n\}} LR_i \tag{13}$$

표 1은 주기적 태스크 집합 τ={(2,10), (3,15), (5,30)}가 주어진다고 가정할 경우, RMS 및 결함 허용에 따른 실행 가능성을 보여 준다. 먼저 RMS에 따른 주기 태스크 집합 τ의 처리기 이용률을 살펴보면 L_1=0.20, L_2=min(0.50, 0.47)=0.47, 그리고 L_3=min(1.00, 0.80, 0.75, 0.57)=0.57이 된다. 따라서 태스크 집합 τ의 전체 처리기 이용률은 L=max(0.20, 0.47, 0.57)=0.57이 되며 L<1이므로 주기 태스크 집합 τ는 실행이 가능하다.

결함 허용을 위해 필요한 백업 시간을 고려한 처리기 이용률을 살펴보면 LR_1=0.40, LR_2=min(0.70, 0.70)=0.70, 그리고 LR_3=min(1.20, 1.00, 0.95, 0.77)=0.77이 된다. 따라서 태스크 집합 τ의 전체 처리기 이용률은 LR=max(0.40, 0.70, 0.77)=0.77이 되며 LR<1이므로 주기 태스크 집합 τ는 실행이 가능하다.

3.3 결함 허용 모듈

결함 허용 모듈은 결함이 발생하였을 경우 타임

표 1. RMS 및 결함 허용을 고려한 주기 태스크 집합에 대한 전체 처리기 이용률 비교

			RMS 처리기 이용률				결함허용을 고려한 처리기 이용률			
i	τ_i	t	$Wi(t)$	$Li(t)$	Li	L	$WRi(t)$	$LRi(t)$	LRi	LR
1	τ_1	10	$C1 = 2$	0.20	0.20		$B1+C1 = 4$	0.40	0.40	
2	$\tau_1 \sim \tau_2$	10	$C1+C2 = 5$	0.50			$B1+C1+C2 = 7$	0.70		
		15	$2C1+C2 = 7$	0.47	0.47		$B1+B2+2C1+C2 = 10$	0.70	0.70	
3	$\tau_1 \sim \tau_3$	10	$C1+C2+C3 = 10$	1.00			$B1+C1+C2+C3 = 12$	1.20		
		15	$2C1+C2+C3 = 12$	0.80			$B1+B2+2C1+C2+C3 = 15$	1.00		
		20	$2C1+2C2+C3 = 15$	0.75			$B1+B2+B3+$ $2C1+2C2+C3 = 19$	0.95		
		30	$3C1+2C2+C3 = 17$	0.57	0.57	0.57	$B1+B2+B3+$ $B4+3C1+2C2+C3 = 23$	0.77	0.77	0.77

관리 모듈의 백업 시간 할당 모듈에서 제공하는 백업 시간을 이용하여 결함이 발생한 태스크를 재실행한다.

본 논문에서의 결함은 일시적으로 발생하며 단일 태스크에 의한 결함은 다른 태스크에 영향을 미치지 않는다고 가정한다. 그러므로 이러한 결함은 태스크를 재실행함으로써 복구될 수 있다. 또한 복수의 결함에 대한 허용은 단일 결함 허용에 비해 상대적으로 많은 비용이 필요하며 이로 인해 태스크 처리 응답 시간이 늦어 질 수 있고 처리기 이용률 측면에서 비효율적이기 때문에 단일 결함에 대한 복구만을 고려한다.

표 2는 결함이 발생한 태스크가 복구 모드 상태에서 수행하는 알고리즘을 나타낸다. 복구 모드에서 태스크의 스케줄링은 우선순위에 의한 선점 방식에 의해 이루어지지만 다음의 경우는 예외이다. 새로운 태스크 τ_i의 우선순위가 복구 태스크 τ_r의 우선순위보다 높고 마감시간이 큰 경우(①)는 우선순위에 의해 스케줄링 되지 않고 복구 태스크의 수행이 완료될 때까지 기다렸다가 실행한다. 즉, 비록 τ_i의 우선순위가 τ_r보다 높더라도 τ_r의 마감시간을 보장하기 위해서 τ_i를 블록 태스크 큐로 전환(②)한 후 기존의 τ_r을 실행한다. 그리고 τ_r의 실행이 완료(③)되면 블록 상태에 있던 태스크 τ_i를 우선순위 태스크 큐로 이동(④)한 후 우선순위 방식에 따라 태스크들을 스케줄링(⑤)한다.

3.4 타임 관리 모듈

3.4.1 백업 시간 관리 모듈

결함 허용 알고리즘은 백업으로 할당된 타임 슬롯을 모든 태스크가 결함 허용을 위해 사용하기 위해서 Overloading 기법[18]을 이용한다. 그렇지 않을 경우, 각 태스크에 대한 백업 시간 요구가 증가할 것이며 처리기 이용률은 감소한다.

RMS 기법에 결함 허용을 적용하기 위해서 결함이 발생한 태스크에 대한 재실행이 마감시간 내에서 보장되어야 한다. 이를 위해서는 재실행에 필요한 추가적인 시간이 필요하다.

백업 시간은 3.2절의 수식 (7)에 의해서 설정된다. 복구 태스크의 마감시간을 보장하면서 재실행에 충분한 실행 시간을 확보하기 위해서 교체(Swapping) 방법을 이용한다. 즉, 결함이 발생하지 않는 상태에서 태스크들이 실행할 경우 이미 할당된 백업 타임은 실행되는 태스크의 시간 뒤로 이동한다. 그러므로 향후 태스크 결함이 발생할 경우 복구에 필요한 시간을 확보할 수 있다.

예를 들어, τ = {(2, 8), (4,16)}라는 태스크가 있을 경우, 초기 스케줄 타임 테이블은 (a)와 같다. 결함이 발생하지 않을 경우 RMS에 의해 수행된다. 먼저, τ_1이 수행하기 위해서 B1과 C1을 교체하고 첫 번째 주기의 τ_1이 작업을 수행한다(b). 이 후, C2가 수행할 때, B1은 다시 C2와 교체된다((c), (d)). 그리고 두 번째 주기의 C1을 실행하기 위해서 B2와 C1이 C2의 수행하기 위해서 B2와 C2가 교체된다((e), (f)).

3.4.2 잉여 시간 관리 모듈

주기적 태스크의 실행 시간과 백업 시간을 할당한 후, 처리기 잉여시간을 계산하고 확보된 잉여시간을 비주기적 태스크의 요청에 할당함으로써 비주기적 태스크를 실행할 수 있다. 잉여 시간을 계산하는 과정은 다음과 같다. 주기 태스크 τ_i의 j번째 요구를 τ_{ij}라고 하면, τ_{ij}의 도착시간은 $(j-1)T_i$이고, 마감시한은 D_{ij}, 그리고 실행시간은 c_{ij}가 된다. $B(t)$는 시간 [0, t]에서 결함 허용을 위해 할당된 백업시간이며 $P_i(t)$는 시간 [0, t]에서 우선순위가 i인 주기 태스크가 실행되는 시간이 된다. $A_i(t)$는 시간 [0, t]에 우선순위가 i보다 크거나 같은 비주기 태스크들의 전체 실행시간이 된다. $I_i(t)$는 시간 [0, t]에서 우선순위

표 2. Execution Algorithm in Recovery Mode of τ_r

New task τ_i arrives	Is recovery task τ_r finished?

if ($priority-\tau_i > priority-\tau_r$) and
($deadline-\tau_i > deadline-\tau_r$) then ····· ①
move τ_i to block_task_queue ····· ②
else
add τ_i to priority_task_queue
execute task
	if $StatusOfRecovery_r$==false then ····· ③
	move blocked task τ_is
	to priority_task_queue ····· ④
	else
	continue Recovery Mode
	execute task ····· ⑤

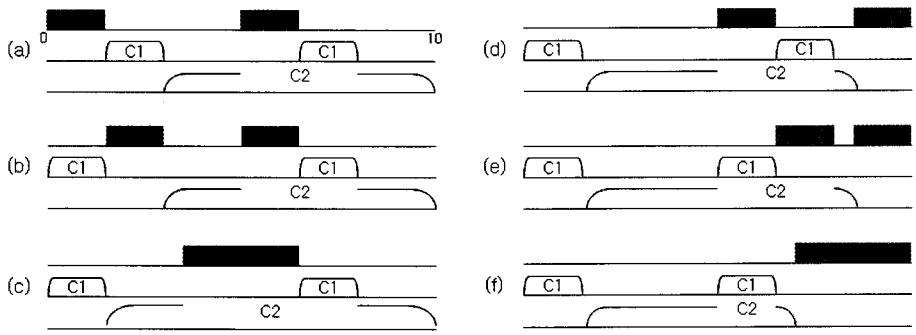

그림 3. 교체 기법

i보다 낮은 태스크들이 수행되거나 처리기가 유휴상태인 시간이 된다. 태스크들의 전체 처리기 이용시간은 다음과 같다.

$$W_i(t) = B(t) + A_i(t) + P_i(t) + I_i(t) \quad (14)$$

A_{ij}(t)를 시간 $[0, C_{ij}]$에서 우선순위가 i보다 크거나 같은 비주기 태스크를 처리하기 위한 최대 잉여시간이라고 하면, 수식 (15)의 조건이 성립된다.

$$A_{ij}(t) = t - B(t) - P_i(t) \quad (15)$$

실행 시간 t에 따른 처리기의 최대 잉여시간(Surplus time of Processor)은 다음과 같다.

$$SP_{all}(t) = \min_{\{1 \le i \le n\}} A_{ij}(t),$$
$$C_{ij-1} \le t < C_{ij},\ j \ge 1 \quad (16)$$

예를 들어 주기 태스크 집합 τ={(2,10), (3,15), (5,30)}가 주어진다고 가정할 경우, 잉여 시간 할당 테이블인 그림 4에서 검은 상자는 처리기의 이용 상태를 의미한다. 즉, 결함이 발생한 태스크의 복구를 위한 백업 시간과 태스크 실행으로 인한 처리기 이용 상태를 나타낸다. 표 1로부터 τ의 전체 처리기 이용률은 77%이고 하이퍼주기는 30이므로 비주기 태스크를 실행할 수 있는 처리기 여유시간은 7(23%)임을 알 수 있다.

3.5 비주기적 태스크 스케줄러

비주기 태스크 스케줄러는 비주기적 태스크들을 실행하기 위해서 잉여시간 관리 모듈로부터 할당 받은 잉여 시간 할당 테이블의 잉여시간을 비주기 태스크에게 제공한다.

그림 5는 주기 태스크 집합 τ={(2,10), (3,15), (5,30)}과 실행시간이 5인 비주기 태스크 τ_{ap}에 대한 예를 보여준다. 비주기 태스크를 실행할 수 있는 시간 중에서 가장 빠른 잉여시간은 19가 되며 이 시간부터 비주기 태스크는 스케줄된다. 그림 5에서 보는 바와 같이 본 논문에서 제안한 주기 및 비주기 태스크 관리 기법은 비주기 태스크인 τ_{ap}의 실행을 보장함과 동시에 모든 주기 태스크들의 실행을 마감시간 내에 완료한다.

3.6 주기적 태스크 스케줄러

주기 태스크 스케줄러는 주기적 태스크의 실행 주기에 따른 우선순위 기반의 스케줄링 기능을 제공한다. 그러나 결함이 발생한 태스크의 결함을 복구하고 있는 상황에서는 실행주기만을 이용하여 우선순위

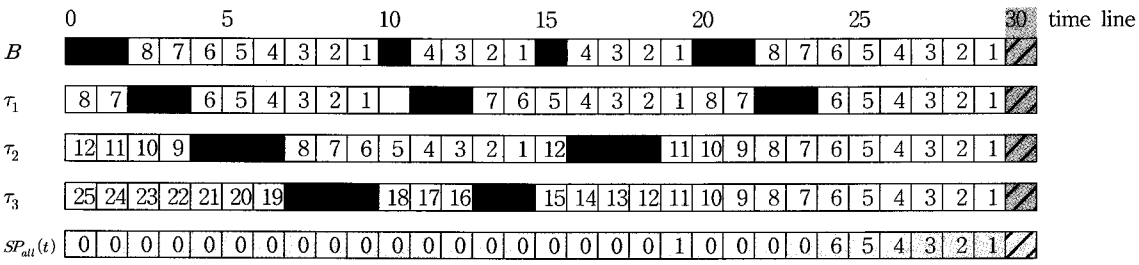

그림 4. 잉여시간 할당 테이블

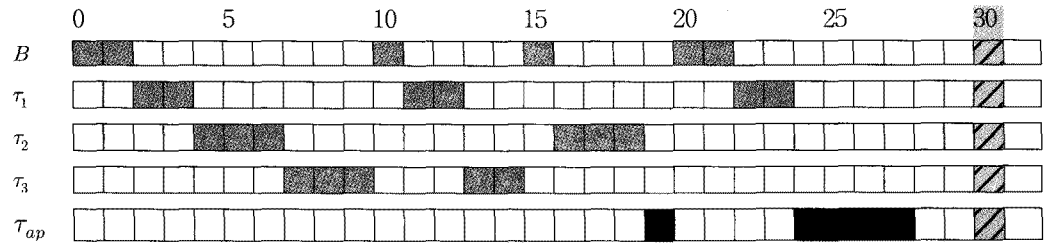

그림 5. Slack Stealing 기법에 의한 태스크 할당 테이블

를 부여하지는 않는다. 즉, 결함이 발생한 태스크의 우선순위 보다 높은 태스크의 실행 요청이 있을 경우 마감시간을 비교하여 우선순위가 높은 태스크의 마감시간이 클 경우 결함이 발생한 태스크의 마감시간을 보장하기 위해서 선점되지 않고 결함이 발생한 태스크의 실행이 종료된 후에 우선순위가 높은 태스크가 수행한다.

4. 결 론

최근 임베디드 시스템은 하드웨어의 사양이 향상되고 있으며 이를 이용하는 사용자들의 요구 또한 다양하게 증가하고 있다. 그러므로 이러한 변화하는 환경에 적용할 수 있는 임베디드 시스템의 운영체제는 태스크 응답의 실시간성을 제공할 수 있어야 하며 일시적인 태스크의 결함에도 지속적인 시스템 운영이 가능하여야 한다.

따라서 본 논문에서는 결함 허용을 고려한 임베디드 실시간 태스크 스케줄러를 설계하였다. 태스크 응답의 실시간성을 고려하기 위해 태스크를 주기적 태스크와 비주기적 태스크로 분류하여 스케줄링한다. 주기적 태스크의 실행을 위해 최적의 알고리즘으로 알려진 RMS(Rate Monotonic Scheduling)을 적용하고 주기적 태스크의 실행 시간 외의 시간인 잉여시간을 비주기적 태스크에게 할당한다. 또한 결함이 발생한 태스크의 결함 허용을 위해서 백업시간을 관리한다.

스케줄러에 의한 주기적 태스크 집합을 실행하기 전에 실행 가능성을 검사함으로써 잘못 설계된 태스크 집합의 실행으로 인한 오버헤드를 줄일 수 있으며 주기적 태스크 집합의 실행 시간을 분석함으로써 결정할 수 있는 잉여 시간을 이용하여 비주기적 태스크의 실행과 완료를 보장할 수 있다. 또한 태스크 결함으로 인해 발생할 수 있는 시스템 장애를 극복하기 위하여 운영 체제 레벨에서 결함이 발생한 태스크의 결함을 허용할 수 있는 방법을 제공한다. 이를 위해 백업 시간을 확보한 후, 결함이 발생한 태스크를 재실행함으로써 복구를 마감시간 내에 완료할 수 있으며 복구 완료에 대한 시간 예측이 가능하다는 특징이 있다.

본 논문은 임베디드 시스템에 적용할 수 있는 스케줄러를 설계하는데 의의를 두고 있으며 향후 본 논문에서 제시한 알고리즘과 기존 연구에서 제시한 알고리즘과의 성능 비교를 위하여 공통 요소들을 분석 및 추출하여 상호 비교할 수 있는 추가적인 연구가 필요할 것으로 사료된다.

참 고 문 헌

[1] J.T. Baldwin, "Predicting and Estimating Real-Time Performance. Embedded Systems Programming," 8(2), Feb. 1995.

[2] L. Doyle and J. Elzey, "Successful Use of Rate Monotonic Theory on a Formidable Real Time System," In 11th IEEE Workshop on Real-Time Operating Systems and Software, pp. 74-78. IEEE, May 1994.

[3] H Kopetz, "Automotive Electronics-Present State and Future Prospects," *In FTCS 25,* 1995.

[4] K. W. Tindell, "Fixed Priority Scheduling of Hard Real-Time Systems," PhD thesis, Univ of York, UK, 1994.

[5] C.L Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," JACM, 20(1), pp. 46-61, 1973.

[6] John Lehoczky, Lui Sha and Ye Ding, "The

Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior," RTSS pp. 166-171, 1989.

[7] J.P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," Proc. 13th Real-Time Systems Symp., pp. 110-123, 1992.

[8] T.S. Tia, "Utilizing Slack Time for Aperiodic and Sporadic Requests Scheduling in Real-Time Systems," Technical Report No. UIUCDCS-R-95-1906, University of Illinois

[9] 정경훈, 김병훈, 이동건, 김창수, 탁성우, "확장성 및 실시간성을 고려한 실시간 센서 노드 플랫폼의 설계 및 구현," 한국통신학회논문지 제32권 제8호, pp. 509-520, 2007.

[10] 김병훈, 정경훈, 탁성우, "주기 및 비주기 태스크의 효율적인 관리를 위한 실시간 센서 노드 플랫폼의 설계," 정보처리학회지 제14-C권 제4호 통권 제114호, pp. 371-382, 2007.

[11] 김희헌, 박학봉, 박문주, 박민규, 조유근, 조성재, "잉여 여유시간을 이용한 연성 비주기 태스크들의 효율적인 스케줄링," 정보과학회논문지, 시스템 및 이론 제36권 제1호, 2009.

[12] S. Gosh, R. Melhem, D. Moss, and J. Sensarma, "Fault-tolerant rate-monotonic scheduling," *Real-Time Systems*, Vol.15, No. 2, pp. 149-181, 1998.

[13] M. Pandya and M. Malek, "Minimum achievalble utilzation for fault-tolerant processing of periodic tasks," *IEEE Trans. on Comput.*, Vol.47, pp. 1102-1113, 1998.

[14] S. Gosh, R. Melhem, and D. Moss, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel and Distributed Systems*, Vol.8, No.3, pp. 272-284, 1997.

[15] Y. S. Hong and H. W. Goo, "A fault-tolerant technique for scheduling periodic tasks in real-time systems," Proc. of the Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, 2004.

[16] Ching-Chih Han, Kang G. Shin, and Jian Wu, "A Fault-Tolerant Scheduling Algorithm for Real- Time Periodic Tasks with Possible Software Faults," *IEEE Transactions on Computers*, Vol.52, No.3, pp. 362-372, Mar. 2003.

[17] 정경훈, 탁성우, 김창수, "결함허용이 가능한 임베디드 실시간 태스크 관리 메커니즘," 멀티미디어학회논문지 제10권 제7호, pp. 882-892, 2007.

[18] S. Ghosh, D. Mosse, and R. Melhem, "Implementation and Analysis of a Fault-Tolerant Scheduling Algorithm," IEEE Transactions on Parallel and Distributed Systems, Mar 1997.

전 태 건

1997년 부경대학교 전자계산학과 (학사)
1999년 부경대학교 전자계산학과 (석사)
2001년 부경대학교 전자계산학과 (박사수료)

관심분야 : 임베디드 운영체제, 센서네트워크, 실시간 스케줄링, 분산병렬처리

김 창 수

1991년 중앙대학교 컴퓨터공학과 박사
2006년~현재 유비쿼터스 부산 도시협회 방재분과위원장
2006년~현재 (사)그레고리장학회 이사

2009년~현재 한국멀티미디어학회 정책자문위원
1992년~현재 부경대학교 IT융합응용공학과 교수
관심분야 : 방재 IT, UIS/GIS, 운영체제, 시멘틱 웹, 재난관리, 공간검색, 도시방재 등