Journal of Korea Multimedia Society Vol. 10, No. 7, July 2007(pp. 882-892)

# 결함허용이 가능한 임베디드 실시간 태스크 관리 메커니즘

정경훈[†], 탁성우[††], 김창수[†††]


## 요 약

본 논문에서는 임베디드 실시간 운영체제 수준에서 주기적 및 비주기적 태스크 스케줄링뿐만 아니라 태스크들의 일시적인 결함을 복구할 수 있는 결함허용이 가능한 임베디드 실시간 태스크 관리 메커니즘을 제안한다. 기존의 임베디드 운영체제들은 주기적 및 비주기적 태스크들을 동시에 고려한 스케줄링 메커니즘을 지원하지 않는다. 그리고 태스크 결함 복구 메커니즘의 미지원으로 인해 결함 태스크로 인한 시스템 고장을 야기할 수도 있다. 제안된 결함허용 실시간 태스크 관리 메커니즘은 운영체제 수준에서 주기적 태스크들의 마감시한과 비주기적 태스크의 실행완료를 보장할 뿐만 아니라 일시적인 결함이 발생한 태스크를 복구함으로써 태스크 결함으로 인한 시스템 고장을 방지할 수 있다.


# Mechanism for Managing Fault-Tolerant Embedded Real-Time Tasks

Kyunghoon Jung[†], Sungwoo Tak[††], Changsoo Kim[†††]


## ABSTRACT

In this paper, we propose a mechanism for both scheduling the hybrid-task set which consists of periodic and aperiodic tasks and recovering tasks with transient faults on the level of the operating system. Existing embedded operating systems would not provide the scheduling of both periodic and aperiodic tasks. Also because of not supporting the recovery of task failures, they can not prevent system failure from transient task faults. Proposed method, on the level of operating system, is able to not only meet the deadlines of all periodic tasks but also complete the execution of aperiodic tasks. In addition, it is able to prevent the system failure from transient task faults by recovering the task faults.


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

## 1. 서 론

산업용 로봇, 공장 자동화와 같은 산업용 제어 시스템이나 자동차 엔진제어, 항공기 운행 제어와 같은 교통제어 시스템을 위한 임베디드 운영체제는 범용 운영체제와 달리 제한된 컴퓨팅 자원을 효율적으로 관리하고 외부 반응에 즉각적으로 응답하는 실시간성을 지원해야 한다. 일반적으로 기존의 상용 및 공개용 임베디드 운영체제들은 실시간 태스크들을 스케줄하기 위해 POSIX 1003.1.B 표준[1]에서 제안한 태스크 스케줄링 방법들을 지원한다. POSIX 1003.1B는 실시간 요구사항을 만족하는 애플리케이션을 위

※ 교신저자(Corresponding Author) : 김창수, 주소: 부산광역시 남구 대연 3동(608-737), 전화 : 051)620-6394, FAX : 051)620-6394, E-mail : cskim@pknu.ac.kr
접수일 : 2007년 2월 12일, 완료일 : 2007년 6월 14일
[†] 정회원, 부산대학교 U-Port 정보기술산학공동사업단 (E-mail: jungkh@pknu.ac.kr)
[††] 종신회원, 부산대학교 정보컴퓨터공학부/컴퓨터 및 정보통신 연구소 (E-mail : swtak@pusan.ac.kr)
[†††] 종신회원, 부경대학교 전자컴퓨터정보통신공학부

해 Unix 인터페이스를 확장한 API(Application Programming Interface)이며, FIFO(First-In First-Out), 선점형 우선순위 기반 스케줄링과 라운드 로빈(Round-Robin) 스케줄링을 실시간 태스크 스케줄링을 위한 기본 요구사항으로 제안하고 있다[1]. 그러나 이러한 태스크 스케줄링 기법들은 임베디드 운영체제뿐만 아니라 범용 운영체제에서도 사용되는 기법들이므로 임베디드 실시간 운영체제가 결정성(deterministic)과 신뢰성(reliability)과 같은 실시간 요구사항을 만족하기 위해서는 실시간 태스크들을 위한 스케줄링 메커니즘을 지원할 필요가 있다. 실시간 태스크는 실행시간의 속성에 따라 실행시간 간격이 일정한 주기적 태스크와 그렇지 않은 비주기적 태스크로 구분[2]될 수 있다. 주기적 태스크들은 마감시한(deadline)내에 실행완료가 보장되어야 하며 그렇지 않을 경우 태스크 오동작으로 인한 시스템 고장(failure)을 유발할 수 있다. 또한 주기적 태스크는 우선순위가 높은 비주기적 태스크로 인해 마감시한을 초과(miss)할 수도 있다. 따라서 주기적 및 비주기적 태스크가 동시에 존재하는 임베디드 시스템에서 태스크들의 실시간성 지원을 위해서 주기적 태스크들의 마감시한과 비주기적 태스크들의 실행완료를 보장하는 실시간 태스크 관리 메커니즘이 필요하다.

의료장비나 자동차 엔진제어, 항공기 운행 제어와 같이 안전에 민감(safety-critical)한 임베디드 실시간 시스템에서 발생하는 결함(fault)은 인간의 생명에 직접적인 영향을 미치기 때문에 일부 작업의 결함으로 인한 시스템 고장(failure)은 치명적인 결과를 야기할 수 있다. 센서와 마이크로프로세서, 액츄에이터(actuator)로 구성된 자동차의 SBW(Steering-by-Wire) 시스템에서 액츄에이터 결함은 자동차가 운전자의 의지와 상관없이 진행방향을 바꾸어 나가게 할 수도 있다[3]. 따라서 이러한 safety-critical한 임베디드 실시간 시스템을 위한 운영체제는 반드시 작업 결함으로 인한 시스템 고장을 방지하는 메커니즘을 제공해야 한다. 결함은 결함 지속시간에 따라 영구적 결함(permanent fault), 간헐적 결함(intermittent fault), 일시적 결함(transient fault)으로 분류될 수 있다. 결함에 대한 기존의 결함허용 연구들은 주로 여분(redundancy) 기법들을 사용하고 있으며, 이에는 크게 여분의 하드웨어나 소프트웨어 컴포넌트를 이용하는 공간여분과 결함 발생 시 가장 최근의 작업 성공시점부터 작업을 재시작(rollback)하는 시간 여분 기법으로 분류될 수 있다[4].

본 논문은 주기적 및 비주기적 태스크들이 동시에 실행되는 임베디드 실시간 시스템에서 주기적 태스크들의 마감시한과 비주기적 태스크들의 실행완료를 보장함과 동시에 태스크의 일시적인 결함을 복구할 수 있는 결함허용이 가능한 실시간 태스크 관리 메커니즘을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 기술하고, 3장에서는 본 논문에서 제안하는 결함허용 임베디드 실시간 태스크 관리 메커니즘에 대해 기술한다. 4장에서는 구현 결과 및 분석을 기술하며, 마지막 5장에서는 결론을 기술한다.

## 2. 관련 연구

Liu와 Layland[5]에 의해 처음 제안된 RM(Rate Monotonic)은 태스크 집합내의 개별 태스크 스케줄 가능성을 검사하여 전체 태스크 집합의 스케줄 가능성을 판단하며, 고정 우선순위 기반의 선점형 스케줄링 방식하에서 최적의 스케줄링 기법임이 증명되었다. RM[5]에서 사용하는 전체 처리기 이용률 $U=\sum_{i=1}^{N}\frac{C_i}{T_i}\leq N\left(2^{\frac{1}{N}}-1\right)$은 N이 증가함에 따라 0.69에 수렴하며, 이는 U의 값이 0.69보다 작은 태스크 집합은 RM 스케줄링으로 항상 스케줄 가능함을 의미한다. 그러나 Liu의 알고리즘[5]에서 전체 처리기 이용률 U를 만족하지 않는 태스크 집합이라 할지라도 RM 스케줄링 기법에 의해 타당한 스케줄이 생성 가능함을 Lehoczky[6]가 보여주었다. Lehoczky[6]은 [5]의 스케줄 가능성 분석 방법의 부정확성을 보완하여 스케줄 가능성 검사를 위한 필요충분조건을 제시하였다. Joseph 등[7]과 Audsley 등[8]은 태스크의 최악 반응시간을 계산하여 태스크 스케줄 가능성을 분석하는 기법을 제안하였다. Lehoczky와 Ramos-Thuel[9]이 제안한 slack stealing 알고리즘은 비주기적 태스크 처리를 위한 주기적 태스크를 생성하지 않고, 모든 주기적 태스크들의 마감시간을 만족함과 동시에 비주기적 태스크가 사용가능한 CPU 시간을 계산하는 작업을 수행하는 slack stealer라는 passive 태스크를 생성한다. 이렇게 함으로써 주기적 태

스크 뿐만 아니라 비주기적 태스크의 응답시간 예측과 실행종료를 보장하였다. 그러나 비주기적 태스크의 실행가능 시간을 계산하기 위해 모든 주기적 태스크들의 스케줄 가능성을 검사하기 때문에 시간 알고리즘의 계산복잡도로 인한 시간 오버헤드가 크다는 단점이 있다. Lencevicius[10]는 개인휴대통신장비에서의 고정 우선순위 태스크 스케줄링 성능향상을 위한 실행시간 아키텍처(Runtime Architecture)를 제안하였다. 이 아키텍처에서 Raimondas는 시간적으로 중첩(overlap)이 가능한 use cases들의 집합인 featureset 개념을 사용하였으며, featureset이 항상 스케줄가능하면 태스크 집합이 스케줄가능함을 보였다. 임보섭[11]은 uC/OS를 기반으로 한 다중레벨 태스크 스케줄링을 제안하였다. 이 방법에서 태스크의 최종 우선순위는 각 태스크의 우선순위와 마감시한에 따라 3단계로 구분되어 결정된다. 태스크의 최종 우선순위는 1단계에서 태스크 우선순위 인자 값에 의해 결정되며 2단계에서 태스크의 마감시한과 우선순위의 조합에 의해 결정된다. 그리고 3단계에서 마감시한에 의해서 태스크 우선순위가 결정된다. 이렇게 함으로써 중요한 태스크는 1단계에서 관리되고, 나머지 태스크들은 마감시한에 의해 유연하게 스케줄링 될 수 있도록 하였다. 그러나 임보섭[11]은 비주기적 태스크 스케줄링을 고려하지 않았으며, 주기적 태스크의 마감시한만 고려할 뿐 태스크 주기를 고려하지 않았기 때문에 마감시한이 주기보다 짧은 경우에 대한 주기적 태스크들을 스케줄할 수 없다. 남상엽 등[12]은 동일 우선순위를 가지는 태스크들에 대해 우선순위 기반 라운드 로빈 스케줄링이 가능한 운영체제를 제안하였으나, 태스크의 주기와 마감시한이 고려되지 않았을 뿐만 아니라 비주기적 태스크 스케줄링 알고리즘은 제시하지 않았다.

Gosh 등[13]은 같은 처리기 상에서 결함 태스크의 재실행을 통한 일시적 결함을 허용하는 RM 기법을 제안하였다. [13]은 다음에 이어질 두 개의 요청들 사이의 최소한의 계산에 필요한 slack time을 백업 태스크로 취급하여 결함 태스크의 실행에 사용하였다. Pandya와 Malek[14]은 RM 스케줄링에 대한 최소한의 이용 가능한 이용률을 도출하였다. 이 방법은 결함 발생 이후의 태스크들 중 완료하지 않은 모든 태스크들을 재수행함으로써 복구작업을 수행하며, 처리기 이용률이 0.5보다 작거나 같으면 하나의 일시적 결함발생에 대해 어떠한 태스크도 마감시한을 넘기지 않도록 하였다. Gosh 등[15]는 primary/backup 방법을 사용하여 비선점이며 독립적인 비주기적 실시간 태스크들의 결함 허용 스케줄링을 연구하였다. [15]는 태스크가 시스템에 도착하면 primary는 가능한 빨리 스케줄되며 backup은 primary 스케줄링 시간보다 늦고 태스크의 마감시한 보다 빠른 시간에 스케줄된다. Primary가 성공적으로 수행 종료하면 그에 대응하는 스케줄된 backup의 할당이 해제된다. Hong[16]은 primary/backup 기반의 분산 실시간 시스템에서 처리기 이용률을 높이면서 주기적 태스크들의 결함허용 스케줄링 방법을 제안하였다. 이 방법에서 각 태스크는 처리기 이용률이 0.5보다 작으면 TL, 크거나 같으면 TH로 분류된다. TH로 분류된 태스크의 경우 primary 태스크와 backup 태스크는 동일한 처리기에 할당되며, backup 태스크가 동일한 처리기에 스케줄 불가능하면 TL 태스크 리스트에 포함된다. TL에 포함된 태스크들의 경우 primary 태스크와 backup 태스크는 서로 다른 처리기에 할당된다. 그리고 TL에 속하는 TH의 backup 태스크는 TL의 backup 태스크 할당 후에 처리기에 할당된다. 이렇게 함으로써 TH의 primary 태스크에서 일시적인 결함이 발생하면 그에 대응하는 backup 태스크가 동일한 처리기에 할당되어 있으므로 결함복구를 위한 시간오버헤드를 줄일 수 있다. 그러나 Hong[16]의 방법은 분산 다중 실시간 처리기 시스템을 위한 결함허용 방법이기 때문에 이를 단일 처리기를 가지는 임베디드 실시간 시스템에 적용하기에는 한계가 있다.

## 3. 결함허용 임베디드 실시간 태스크 스케줄러

### 3.1 가정 및 시스템 구성

본 논문에서 제안하는 태스크 스케줄러를 위해 다음의 가정을 한다.

- 주기적 태스크를 $\tau_i$라고 하고, $\tau_i$의 실행시간과 주기를 각각 $C_i$와 $T_i$라고 하면 n개의 태스크를 가지는 태스크 집합 $S=\{\tau_1,\tau_2,\cdots,\tau_n\}$에 대해 $\tau_i$는 $\tau_{i-1}$보다 우선순위가 낮다.
- 시스템은 하나의 처리기를 가진다.
- 선점의 비용과 스케줄링 오버헤드는 무시할 수 있다.

- 태스크들은 모두 독립적이며, 주기적 태스크는 스스로 수행을 중단하지 않는다.
- 태스크 집합은 RM 알고리즘에 의해 스케줄된다.
- 주기적 태스크의 마감시한과 주기는 동일하다.
- 비주기적 태스크들은 FIFO(Firt-In First-Out) 순서로 스케줄된다.
- 비주기적 태스크의 도착 시간은 알려지지 않으며, 수행시간은 도착 시에 알려진다.
- 주기적 태스크 집합은 초기 시작 시간이 동일한 동기적(synchronous) 태스크 집합이다.

그림 1은 본 논문이 제안하는 결함허용 임베디드 실시간 태스크 관리 메커니즘(FT-ERTS: Fault Tolerant - Embedded Real-time Task Scheduler)으로서 실시간 태스크 스케줄러(PATS: Periodic and Aperiodic Task Scheduler)와 결함허용 메커니즘(FTM: Fault Tolerance Mechanism), 주기적 태스크 집합, 비주기적 태스크 집합으로 구성된다. PATS는 스케줄 가능성 검사자(SC: Schedulability Checker)와 여유시간 매니저(STMAT: Slack Time Manager for Aperiodic Task), 비주기적 태스크 스케줄러(ATS: Aperiodic Task Scheduler), 주기적 태스크 스케줄러(PTS: Periodic Task Scheduler)로 구성된다. SC는 주기적 태스크 집합의 스케줄 가능 여부를 자동 검사한다. STMAT는 처리기 여유시간(slack time)을 평가하며 여유시간 계산자(STCAT: Slack Time Calculator)와 여유시간 할당자(STAAT: Slack Time Allocator)로 구성된다. STCAT는 여유시간을 계산하여 여유시간 테이블(STT: Slack Time Table)에 저장하며, STAAT는 ATS에 의해 선정된 비주기적 태스크에게 여유시간을 할당한다. ATS는 STAAT에 의해 할당받은 비주기적 태스크를 FIFO 방식으로 스케줄한다. PTS는 주기적 및 비주기적 태스크들을 우선순위에 의해 스케줄하며 주기적 및 비주기적 태스크들 중에서 가장 높은 우선순위의 태스크를 실행한다. 본 논문은 주기적 및 비주기적 태스크들을 스케줄하기 위해 RM(Rate Monotonic) 알고리즘[6]과 Slack Stealing 알고리즘[5]을 사용한다. FTM은 결함 태스크 관리자(FM: Fault Manager)와 PATS의 구성요소들간의 상호 유기적인 관계를 통해 태스크 결함을 복구한다. FM은 결함 태스크를 식별하여 ATS에게 태스크 스케줄과 재실행을 요구한다.

### 3.2 임베디드 실시간 태스크 스케줄러

#### 가. 스케줄 가능성 검사자

SC는 태스크 집합내 개별 태스크의 처리기 이용률 계산을 통해 전체 처리기 이용률을 계산한다. 처리기 이용률 계산을 위해 3.1절의 가정을 따르며 기본 용어들은 표 1과 같다. 그리고 표 1에 사용된 시간 t는 (1)의 $S_i$에 의해 계산된다.

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

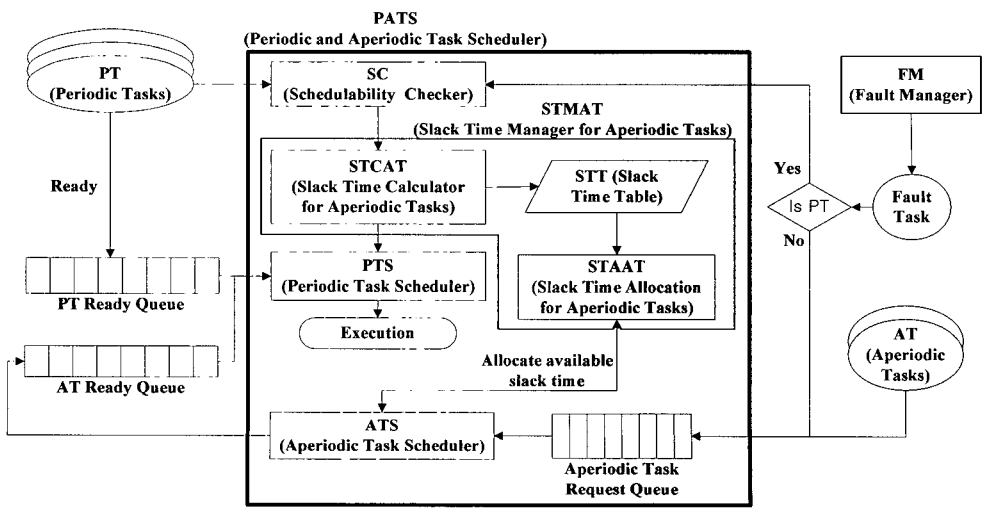



그림 1. FT-ERTS 구성도

표 1. 처리기 이용률에 사용된 기본 용어

| 용어 | 정의 | 의미 |
|---|---|---|
| $W_i(t)$ | $\sum_{j=1}^{i} C_j \cdot \lceil t/T_j \rceil$ | 태스크 작업량 |
| $L_i(t)$ | $W_i(t)/t$ | 처리기 이용률 |
| $L_i$ | $\min_{\{0<t\le T_i\}} L_i(t)$ | 개별 태스크의 처리기 이용률 |
| $L$ | $\max_{\{1\le i\le n\}} L_i$ | 태스크 전체 집합의 처리기 이용률 |

개별 태스크의 처리기 이용률은 (1)에서 구한 $S_i$의 요소 값들에 대해 계산한 $L_i(t)$의 값들 중에서 최소값인 $L_i$가 된다. $L_i$의 값들 중 최대값인 L은 태스크 전체 집합의 처리기 이용률로서 $L \le 1$이면 태스크 집합 내의 모든 주기적 태스크들이 스케줄 가능하다.

주기적 태스크 집합 $\tau=(T,C)=\{(5,2),(10,3),(40,4)\}$가 주어질 때, 표 2는 $\tau$에 대한 모든 가능한 처리기 이용률을 나타낸다. 표 2에서 $L_1$=0.40, $L_2$=min(1.00, 0.70)=0.70, $L_3$=min(1.80, 1.10, 1.06, 0.90, 0.92, 0.83, 0.85, 0.80)=0.80이다. 따라서 $\tau$의 전체 처리기 이용률 $L$=max(0.40, 0.70, 0.80)=0.80<1이므로 조건 L≤1을 만족하여 주기적 태스크 집합 $\tau$는 스케줄가능하다.

표 2. $\tau_2$의 처리기 이용률

| $i$ | $\tau_i(T_i,C_i)$ | $t$ | $W_i(t)$ | $L_i(t)$ | $L_i$ | $L$ |
|---|---|---|---|---|---|---|
| 1 | $\tau_1(5, 2)$ | 5 | $C_1$=2 | 0.4 | *0.4* | |
| 2 | $\tau_2(10,3)$ | 5 | $C_1+C_2$=5 | 1.0 | | |
| | | 10 | $2C_1+C_2$=7 | 0.7 | *0.7* | |
| 3 | $\tau_3(40,4)$ | 5 | $C_1+C_2+C_3$=9 | 1.8 | | |
| | | 10 | $2C_1+C_2+C_3$=11 | 1.1 | | |
| | | 15 | $3C_1+2C_2+C_3$=16 | 1.0 | | |
| | | 20 | $4C_1+2C_2+C_3$=18 | 0.9 | | |
| | | 25 | $5C_1+3C_2+C_3$=23 | 0.9 | | |
| | | 30 | $6C_1+3C_2+C_3$=25 | 0.8 | | |
| | | 35 | $7C_1+4C_2+C_3$=30 | 0.8 | | |
| | | 40 | $8C_1+4C_2+C_3$=32 | 0.8 | *0.8* | ***0.8*** |

### 나. 여유시간 매니저

여유시간 매니저(STMAT)는 처리기 여유시간 계산을 위해 다음의 용어를 정의한다. $\tau_i$의 j번째 요구를 $\tau_{ij}$라고 하면, $\tau_{ij}$의 도착시간(release time)은 (j-1)$T_i$이고, 마감시한은 $D_{ij}$, 실행시간은 $c_{ij}$가 된다. $a_i(t)$를 시간 [0, $t$], $0 \le t \le D_{ij}$에서 우선순위가 $i$보다 크거나 같은 비주기적 태스크들의 전체 실행시간이라 하고, $P_i(t)$를 [0, $t$]에서 우선순위 $i$인 주기적 태스크의 준비시간이라고 하자. 그리고 $I_i(t)$를 [0, $t$]에서 우선순위 $i$보다 낮은 태스크들이 수행되거나 처리기가 유휴상태인 시간이라고 하면, 태스크들의 전체 처리기 이용시간 $W_i(t)$는 (2)에 의해 구할 수 있다.

$$W_i(t) = a_i(t) + P_i(t) + I_i(t) \tag{2}$$

$A_{ij}$를 시간 $[0, C_{ij}]$에서 우선순위가 $i$보다 크거나 같은 비주기 태스크들을 처리하기 위한 최대 여유시간이라고 하면 (2)는 (3)의 조건식을 얻을 수 있다. 이때, $C_{ij}$는 주기적 태스크 $\tau_{ij}$의 종료시간이다.

$$\min_{\{0 \le t \le D_{ij}\}}\{(A_{ij}+P_i(t))/t\} = 1 \tag{3}$$

$A_i(t) = A_{ij}, \quad C_{ij-1} \le t < C_{ij}, j \ge 1$ 이라고 하고, $A^*(t)$를 최대 여유시간이라고 하면,

$$A^*(t) = \min_{\{1 \le i \le n\}} A_i(t), \quad C_{ij-1} \le t < C_{ij} \tag{4}$$

가 된다. 주기적 태스크 집합 $\tau=(T,C)=\{(5,2),(10,3),(40,4)\}$가 주어지고 모든 $\tau_{i\ 1\le i\le 3}$에 대한 최소공배수인 hyperperiod를 계산하면 40이 된다. 앞 절로부터 $\tau$는 스케줄 가능이므로 (2)-(4)로부터 그림 2의 스케줄 할당 테이블을 얻을 수 있다. 그림 2에서

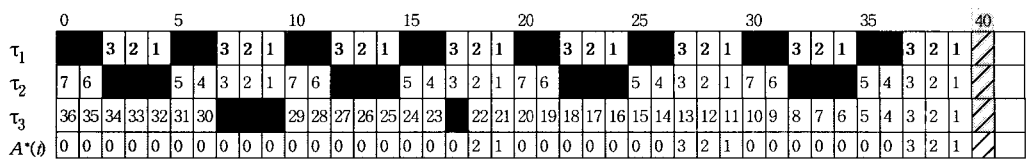



그림 2. 여유시간 매니저에 의해 생성된 여유시간 테이블(STT)

명암부분은 태스크 실행으로 인해 처리기가 busy인 상태를 의미한다. 표 2로부터 $\tau$의 전체 처리기 이용률은 80%이고 hyperperiod는 40이므로 비주기적 태스크 수행을 위해 할당 가능한 처리기 여유시간은 8(20%)임을 알 수 있다. 그림 2에서 비주기적 태스크 AT가 t=18에 도착하였다면 STMAT는 최대 2의 여유시간을 할당할 수 있다.

다. 비주기적 태스크 스케줄러

비주기적 태스크 스케줄러(ATS)는 비주기적 태스크 요구 큐(ATRQ: Aperiodic Task Request Queue)에 도착한 비주기적 태스크에 대해 STMAT로부터 STT의 값을 할당받은 비주기적 태스크를 FIFO 방식으로 스케줄한다. 주기적 태스크 집합 $\tau=(T,C)=\{(5,2), (10,3),(40,4)\}$와 실행시간이 9인 비주기적 태스크 AT가 주어지고, AT의 우선순위는 $\tau_2$보다는 작고 $\tau_3$보다는 크다고 하자. $\tau$로부터 hyperperiod는 40이다. 그림 3은 $\tau$와 AT를 선점형 우선순위 기반의 태스크 스케줄링 시의 태스크 할당 테이블로서 AT는 실행을 완료하지만 $\tau_3$는 AT로 인해 41에 실행을 완료하여 마감시한을 초과한다. 그림 4는 STMAT로부터 여유시간을 할당받아 AT를 스케줄한 경우의 태스크 할당 테이블이다. 그림 4에서 AT는 STMAT로부터 여유시간을 할당받아야만 스케줄되기 때문에 비록 $\tau_3$보다 우선순위가 높지만 $\tau_3$의 실행완료 후에 스케줄된다. 왜냐하면 그림 2의 STT에서 할당 가능한 가장 빠른 여유시간은 시각 18이기 때문에 AT는 18에서 최초로 스케줄될 수 있기 때문이다. 그림 4에서 모든 주기적 태스크들은 hyperperiod내에 자신의 실행을 완료하는 반면 AT는 hyperperiod내에 실행완료 할 수 없으나 다음 hyperperiod에서 실행을 완료할 수 있다.

라. 주기적 태스크 스케줄러

주기적 태스크 스케줄러(PTS)는 주기적 태스크와 비주기적 태스크에 대해 선점 가능한 우선순위 기반의 스케줄링을 수행한다. 주기적 태스크들의 우선순위는 RM[6]에 의해 부여되며 비주기적 태스크는 설계단계에서 태스크의 중요도에 따라 우선순위가 정해진다. PTS는 태스크를 스케줄하기 위해 주기적 및 비주기적 태스크 우선순위 테이블을 관리하며, 두 개의 우선순위 테이블에서 가장 높은 우선순위를 가지는 태스크를 선정하여 스케줄한다.

### 3.3 결함허용 메커니즘

그림 5는 본 논문에서 제안하는 결함허용 메커니즘(FTM: Fault Tolerance Mechanism)으로서, 결함 태스크 관리자(FM: Fault Manager)와 PATS와의 상호 연관성을 가지면서 태스크 결함을 복구한다. 임베디드 시스템의 낮은 처리기 성능으로 인해 태스크 결함은 태스크 재시작(task reexecution) 방법을 통해 복구된다. 태스크 실행 중에 결함이 발생하면, FM은 결함 태스크가 비주기적 태스크인지 주기적 태스크인지를 판별한다. 만약 비주기적 태스크 결함이면 비주기적 태스크 우선순위 테이블에서 해당 태스크를 삭제한다. 그런 다음 FM은 ATS에게 결함 태스크 재실행을 요청하고, ATS는 STAAT로부터 STT의 값을 할당받은 결함 태스크를 스케줄한다. 비주기적

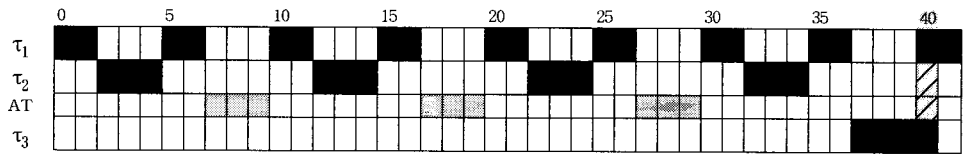



그림 3. 우선순위 기반에 의한 태스크 할당 테이블

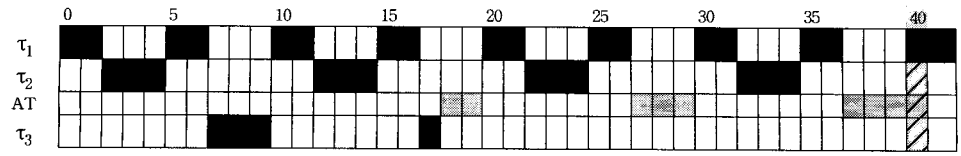



그림 4. STMAT를 이용한 태스크 할당 테이블

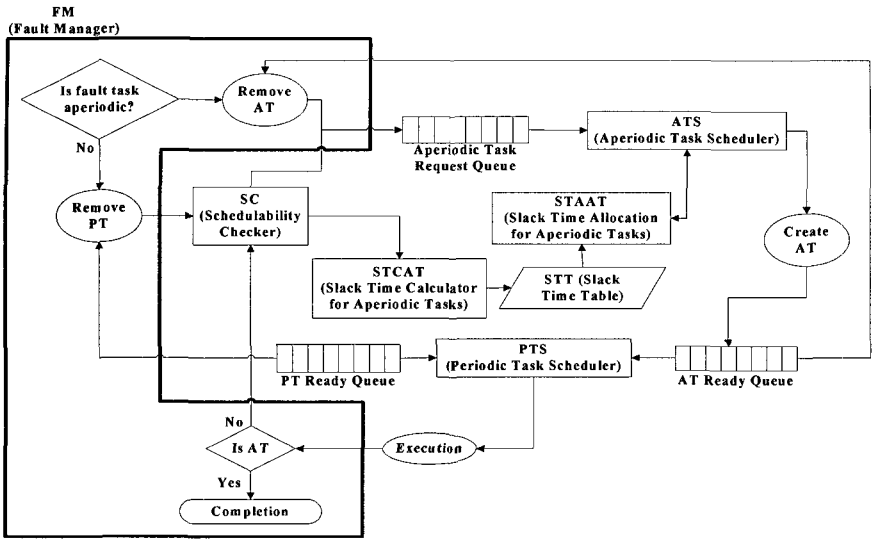



그림 5. 결함허용 메커니즘

태스크로 스케줄된 결함 태스크는 PTS에 의해 우선순위에 따라 재실행된다. 결함 태스크의 복구가 완료되면 FM은 복구된 태스크를 비주기적 태스크 우선순위 테이블에 등록한 후에 복구 작업을 완료한다. 결함 태스크가 주기적 태스크이면, FM은 주기적 태스크 우선순위 테이블에서 해당 태스크를 삭제하고 스케줄 가능성 검사를 재요청한다. SC는 결함 태스크를 제외한 주기적 태스크들에 대해 스케줄 가능성 검사를 수행하고, STCAT는 STT를 갱신한다. 그런 다음 비주기적 태스크 결함복구와 동일한 작업을 수행한다. 주기적 태스크 복구 작업의 경우, 복구가 완료되면 FM은 스케줄 가능성 검사와 STT값을 한 번 더 갱신하여 주기적 태스크 집합을 이전의 상태로 복구한다.

## 4. 구현 및 분석

본 논문에서 제안한 FT-ERTS의 구현을 위해 공개용 임베디드 실시간 운영체제인 uC/OS ver. 2.76을 소프트웨어 환경으로 사용하며, 인텔 Pentium IV 기반의 Windows XP환경에서 시뮬레이션을 수행한다. 본 절에서는 SC, STMAT, 그리고 실시간 태스크들에 대한 평균 응답시간 계산, 결함 태스크의 복구 과정을 통해 본 논문의 제안 방법을 분석한다. 표 3은

표 3. AOCS의 실시간 태스크들 (단위: ms)

| 우선순위 | 태스크 이름 | 주 기 | 최악경우의 수행시간 |
|---|---|---|---|
| 5 | Real_Time_Clock | 50 | 0.28 |
| 4 | Read_Bus_IP | 10 | 1.76 |
| 9 | Command_Actuators | 200 | 2.13 |
| 10 | Request_DSS_Data | 200 | 1.43 |
| 11 | Request_Wheel_Speeds | 200 | 1.43 |
| 7 | Request_IRES_data | 100 | 1.43 |
| 6 | Process_IRES_data | 100 | 8.21 |
| 8 | Control_Law | 200 | 52.84 |
| 13 | Process_DSS_data | 1000 | 5.16 |
| 12 | Calibrate_Gyro | 1000 | 6.91 |

표 4. STMAT 예제 태스크 집합(시간단위: ms)

| 태스크 우선순위 | 주 기 | 최악 경우의 수행시간 |
|---|---|---|
| 4 | 10 | 2 |
| 5 | 40 | 4 |
| 6 | 50 | 5 |
| 7 | 50 | 6 |
| 8 | 100 | 13 |
| 9 | 100 | 9 |
| 10 | 200 | 15 |
| 11 | 200 | 5 |
| 12 | 200 | 10 |
| 13 | 200 | 9 |

Olympus 위성에서 실행된 고도 및 궤도 제어 시스템 (AOCS: Attitude and Orbital Control System) [17]의 태스크들로서 SC를 검증하기 위한 태스크 집합이다. 표 4는 STMAT의 수행을 검증하기 위한 태스크 집합으로서, 10개의 주기적 태스크들로 이루어져 있으며 태스크들의 우선순위는 RM에 의해 할당된다.

그림 6은 표 3의 AOCS의 태스크들에 대한 스케줄 가능성 검사를 수행한 결과로서 표 3의 태스크 집합에 대한 최대 처리기 이용률 $L$은 0.5792(<1)이므로 스케줄 가능함을 알 수 있다. 그림 7은 표 4의 태스크 집합에 대한 STT로서 표 4의 태스크 집합에 대한 hyper period는 200이므로 STT의 slack_time[]의 크기는 200이 된다. STT의 값이 0보다 크면 그 시점에서의 비주기적 태스크 실행을 위한 할당 가능 처리기 여유시간이 존재함을 의미한다. 그림 7에서 t=185 일 때 비주기적 태스크 요청이 존재하면 slack_time[185]의 값이 13이므로 최대 13의 처리기 여유시간을 비주기적 태스크에게 할당가능하다.

그림 8은 ATS에 의해 생성된 비주기적 태스크의 실행을 나타낸다. 주기적 태스크 집합은 표 4의 태스크들을 사용하였으며, 비주기적 태스크의 실행시간은 13으로 하고 우선순위는 21로 한다. t=185에서 비주기적 태스크 요청이 도착하면 그림 7로부터 slack_time[185] = 13임을 알 수 있으므로 그림 8(a)와 같이 13의 처리기 여유시간을 비주기적 태스크 실행시간으로 할당 받는다. 생성된 비주기적 태스크가 수행을 완료하면 그림 8(b)와 같이 종료 메시지 "Aperiodic Task 21 completed"를 출력하여 태스크 수행을 완료함을 알 수 있다.

그림 9는 주기적 태스크(Periodic Task 6)에서 결함이 발생할 경우의 결함 복구이다. Periodic Task 6에서 결함이 발생하면 그림 9(a)과 같이 Periodic

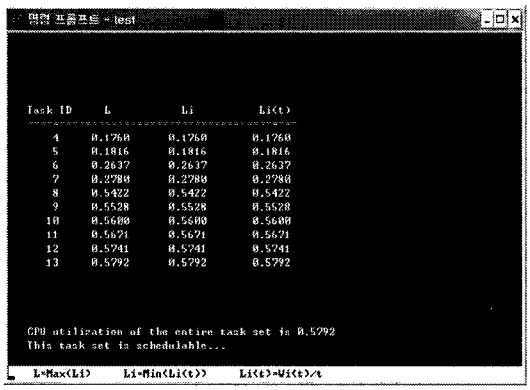



그림 6. 스케줄 가능성 검사 결과

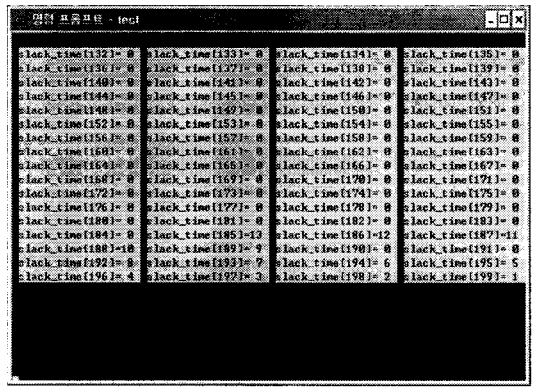
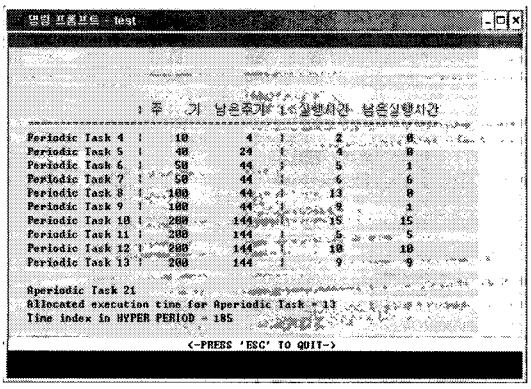



그림 7. 여유시간 관리자 수행 결과




(a) (b)

그림 8. 비주기적 태스크 스케줄

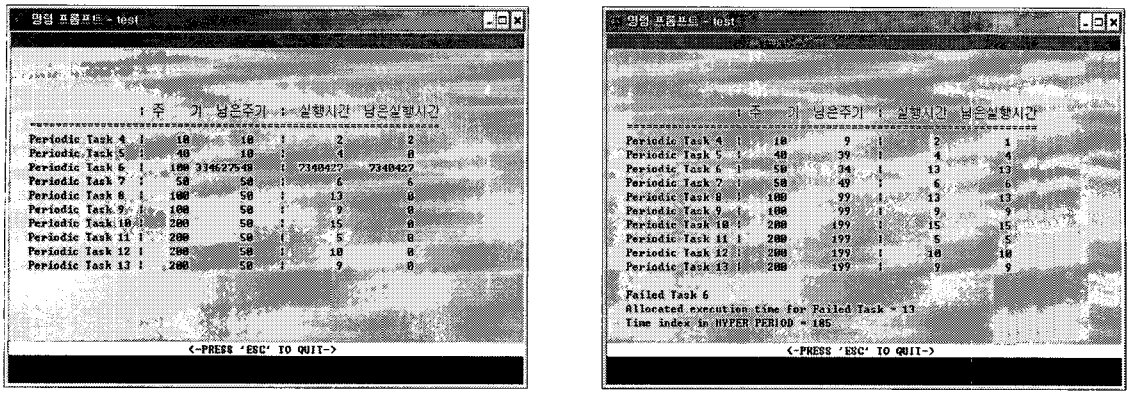



그림 9. 태스크 결함 복구

Task 6은 우선순위 테이블에서 삭제된다. 그런 다음, 그림 9(b)와 같이 결함 허용 메커니즘을 통해 태스크 결함을 복구한다. 그림 9(b)의 경우, 결함 태스크의 복구 작업은 t=185에서 실행되었으며, 이때 할당받은 처리기 여유시간은 13임을 알 수 있다. 복구된 Periodic Task 6은 주기적 태스크 우선순위 테이블에 재등록된 후 정상적으로 작업을 수행함을 알 수 있다.

그림 10은 처리기 이용률이 80%인 주기적 태스크 집합에 대해 비주기적 태스크가 도착하였을 경우, 본 논문에서 제안한 방법을 사용할 경우와 그렇지 않은 경우에 대해 수행하여 비교한 결과이다. 비주기적 태스크의 응답시간 비교를 위해 비주기적 태스크 집합의 처리기 이용률을 각각 5%, 10%, 15%, 20%로 하여 그에 대한 비주기적 태스크의 평균 응답시간을 측정하였다. 그림 10으로부터 본 논문에서 제안한 실시간 태스크 관리 메커니즘이 기존의 주기적 및 비주기적 실시간 속성을 고려하지 않은 방법보다 빠른 응답시간을 가짐을 알 수 있다. 따라서 빠른 응답시간을 필요로 하는 비주기적 태스크가 존재하는 임베디드 실시간 시스템에 본 논문의 제안 방법을 응용할 경우 비주기적 태스크 처리 시간을 단축시킬 수 있을 것이다.

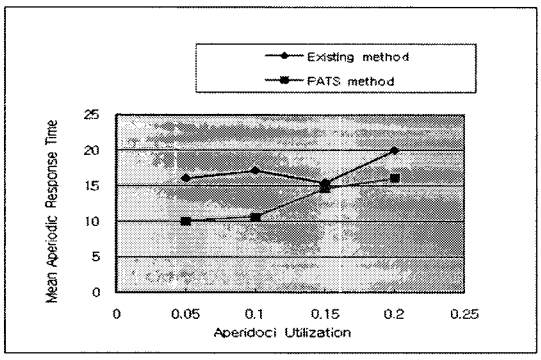



그림 10. 비주기적 태스크의 평균 응답시간 비교

## 5. 결 론

본 논문은 임베디드 실시간 운영체제에서 주기적 및 비주기적 태스크의 실시간성을 보장함과 동시에 일시적인 태스크의 결함을 복구할 수 있는 결함허용 임베디드 실시간 태스크 관리 메커니즘을 제안하고 구현하였다. 기존의 임베디드 실시간 운영체제들은 운영체제 수준에서 실시간성을 지원하지 않거나, 실시간 스케줄링을 지원하더라도 주기적 태스크 또는 비주기적 태스크를 위한 API만 지원하기 때문에 주기적 태스크들의 스케줄 가능성이나 비주기적 태스크의 마감시한을 보장하는 것이 어렵다. 그리고 기존의 임베디드 운영체제들은 의료장비나 항공 운행 제어와 같은 안전에 민감한 분야에서 발생하는 태스크 결함을 복구할 수 있는 메커니즘 연구도 많지 않다. 따라서 제안된 태스크 관리 메커니즘은 운영체제 수준에서 주기적 태스크의 마감시한과 비주기적 태스크들의 응답성을 보장하고 실행 중인 태스크의 일시적인 결함을 복구할 수 있는 기능을 제안하고 있다. 이를 통해 임베디드 실시간 시스템의 신뢰성을 향상

시키고 태스크 결함으로 인한 시스템 고장을 예방할 수 있는 방법을 제시하고 있다.

## 참 고 문 헌


[1] F. Cottet, J. Delacroix, C. Kaiser and Z. Mammeri, *Scheduling in Real-Time Systems*, John Wiley & Sons, Chichester, West Susex, 2002.

[2] 김진석, 김용석, "경성 비주기적 태스크들을 위한 온라인 스케줄링 알고리즘," 정보통신학회 논문지, 제5집, pp. 245-252, 2001.

[3] S. Amberkar, M. Kushion, K. Eschtruth and F. Bolourchi, "Diagnostic Development for an Electric Power Steering Systems," *SAE World Congress, Paper:2000-01-0819*, 2000.

[4] H. Zou and F. Jahanian, "A Real-Time Primary- Backup Replication Service," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 10, No. 6, pp. 533-548, 1999.

[5] C.L Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *JACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[6] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *RTSS*, pp. 166-171, 1989.

[7] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The BCS Computer Journal*, Vol. 29, No. 5, pp. 390-395, 1986.

[8] N.C. Audsley, A. Burns, M. Rihardson, and A. Wellings, "Hard Real-Time Scheduling: The Deadline-Monotonic Approach," *In Proc. of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, pp. 133-137, 1991.

[9] 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.

[10] R. Lencevicius and A. Ran, "Applying Fixed Priority Scheduling in Practice," *Proc. of the 4th international workshop on Software and Performance WOSP '04*, Vol. 29, Issue 1, pp. 156-160, 2004.

[11] 임보섭, 이재윤, 김광, 허신, "MicroC/OS-II 기반에서 Multi-Level 스케줄링의 설계 및 구현", *한국컴퓨터종합학술대회*, Vol. 32, No. 1(A), pp. 832-834, 2005.

[12] 남상엽, 이상원, 박인정, "소규모 임베디드 시스템을 위한 우선순위기반 라운드 로빈 스케줄링 운영체제의 설계 및 구현," 전자공학회논문지, 제40권, 제4호, pp. 42-51, 2003.

[13] S. Gosh et al., "Fault-tolerant Rate-Monotonic Scheduling," *Real-Time Systems*, Vol. 15, No. 2, pp. 149-181, 1998.

[14] M. Pandya and M. Malek, "Minimum Achievable Utilization for Fault-Tolerant Processing of Periodic Tasks," *IEEE Transactions on Computers*, Vol. 47, No. 10, pp. 1102-1113, 1998.

[15] S. Gosh et al., "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.

[16] Y. Hong and H. 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.

[17] A. Burns, A.J. Wellings, C.M. Bailey, and E. Fyfe, "The Olympus Attitude and Orbital Control System: A Case Study in Hard Real-Time System Design and Implementation," *Technical Report YCS-1993-190*, Department of Computer Science, University of York, 1993.

정 경 훈

1996년 부경대학교 전자계산학과 (학사)
1998년 부경대학교 전자계산학과 (석사)
2006년 부경대학교 전자계산학과 (박사)
2006년~현재 부산대학교 U-Port 정보기술산학공동사업단 전임연구원
관심분야 : 센서네트워크, 임베디드 운영체제, 실시간 스케줄링, 분산병렬처리

탁 성 우

1995년 부산대학교 컴퓨터공학과 (학사)
1997년 부산대학교 컴퓨터공학과 (석사)
2003년 미국미주리주립대학교 Computer Science (박사)
2004년~현재 부산대학교 정보컴퓨터공학부 조교수
2004년~현재 부산대학교 컴퓨터 및 정보통신 연구소 겸임 연구원
관심분야 : 유무선 네트워크, SoC 설계, 실시간 시스템, 위치인식, 최적화 기법, 그래프 이론, 큐잉 이론

김 창 수

1984년 울산대학교 전자계산학과 (학사)
1986년 중앙대학교 컴퓨터공학과 (석사)
1991년 중앙대학교 컴퓨터공학과 (박사)
2002년~2003년 UMKC 방문교수
1992년~현재 부경대학교 전자컴퓨터정보통신공학부 교수
관심분야 : USN 방재시스템, LBS/GIS, Semantic GIS 검색, 임베디드 시스템