소 특 집

상용 실시간 운영체제에서의 프로세스 스케줄링에 대한 고찰

은 성 배*, 진 성 기**

*한남대학교 정보통신 멀티미디어공학부, **삼성전자 네트워크 사업부 초고속 통신 사업팀

요 약

실시간 시스템은 응용 프로그램의 수행에 있어서 프로세서의 동작이나 자료의 흐름에 대해서 시한성이 매우 엄격한 시스템이다. 따라서 실시간 운영체제는 이러한 응용 프로그램의 요구에 대처하여 시스템의 자원을 적절히 배분하여 그 시한성을 엄격히 만족시켜 줄 수 있어야 한다. 자원의 배분에 있어서 특히 중요한 고려 사항은 태스크들의 스케줄링과 관계가 있다. 본 논문에서는 상용 실시간 운영체제에서 구현되어 서비스하고 있는 널리 알려진 실시간 스케줄링 기법에 대해서 연구하고, 또한 최근 실시간 스케줄링에서 고려되고 있는 사항들에 대한 현황을 파악한다.

I. 서 론

실시간 시스템이란 정해진 마감시간 내에 응답을 생성하는 컴퓨터 시스템을 말한다. 예를 들어 원자력 발전소의 원자로에서 핵반응 속도를 제어하는 컴퓨터가 마감시간을 넘겨서 제어명령을 처리한다면 원자로가 폭발하는 위험에 빠질 수도 있다. 마찬가지로 항공기의 제어를 담당하는 컴퓨터는 자세 제어를 위한 태스크가 정해진 마감시간 안에 수행되는 것을 보장하여야 한다[1].

이를 위하여 다양한 실시간 스케줄링 기법들이 연구되었는데 단일 프로세서인지 멀티 프로세서인지에 따라, 선점을 허용하는지 아닌지에 따라, 주기 태스크인지 아닌지에 따라, 자원요구 여부에 따라 매우 다양한 연구들이 수행되었다. 이러한 연구들 중에는 실제 시스템에서 구현되어 운영되는 스케줄링 기법들도 있지만 많은 기법들이 이론적인 관심과 흥미에서 연구됐다고 할 수 있다. 특히 최근까지 상용화된 상용 운영체제들은 대부분이 고정우선순위 기반의 선점 스케줄링 기법을 주로 사용하고 있을 뿐이다.

본 논문에서는 이러한 다양한 실시간 스케줄링 연구들 중에서 실제 시스템에서 채택되어 사용되는 스케줄링 기법들에 초점을 맞추어 조사하고 기술하였다. 이를 위하여 상용 실시간 운영체제의 특징에 대하여 조사하였고, 대부분의 상용 실시간 운영체제에서 채택하고 있는 고정우선순위 기반의 선점 스케줄링 기법의 이론적 모태가 되고 있는 주기단조(RM : Rate-Monotonic) 기법과 이를 확장한 스케줄링 기법들, 또한 성능면에서 최적이라고 알려진 최단마감시간우선(EDF: Earliest Deadline First) 기법을 조사, 분석하였다. 또한, 실제 운영환경에서 임계영역을 잘못 처리했을 때 발생하는 우선순위 역전 현상에 대하여 설명하고 이를 해결하는 기법들에 대하여 조사하였다. 끝으로, 최근 들어 큰 관심을 끌고 있는 실시간 특성과 고장 허용 특성을 모두 고려하는 스케줄링 기법들을 조사하고 요약하였다.

본 논문의 내용은 다음과 같다. 먼저 제2장에서는 상용 실시간 운영체제에 대해서 살펴보고 여기서 사용되는 실시간 스케줄링 문제를 정의한 후, 실시간 스케줄링 알고리즘을 태스크 모델에 따라서 분류한다. 제3장에서는 실시간 스케줄링 알고리즘으로 주기단조 알고리즘과 산발(Spora-

dic) 태스크 스케줄링 알고리즘, 그리고 태스크의 마감시간이 주기와 동일하지 않은 경우의 스케줄링 알고리즘에 대해서 설명한다. 제4장에서는 스케줄링을 수행하는데 있어서 임계 영역을 처리하는 문제와 해결책에 대해서 설명하고 제5장에서는 실시간 스케줄링 알고리즘에 고장 허용성을 추가하는 연구에 대해서 기술한다. 마지막으로 제6장에서는 본 논문의 내용을 요약하고 끝을 맺는다.

II. 배 경

1. 상용 실시간 운영체제

상용 실시간 운영체제는 구조적인 측면에서 크게 두 가지로 분류할 수 있다. 하나는 멀티 쓰레드(Multi Thread) 모델이며, 다른 하나는 멀티 프로세스(Multi Process) 모델이다[3]. 멀티 쓰레드 모델은 운영체제의 커널과 응용 프로그램이 합쳐져서 서로의 구분이 없는 하나의 큰 프로그램이 되어 작동하는 구조로서, 커널과 응용 프로그램이 공통의 작업 영역안 메모리를 자유롭게 접근할 수 있다. 운영체제의 크기가 작고, 비교적 작은 크기의 시스템에서 구현이 쉽고 빠르다는 장점이 있지만, 커널과 응용 프로그램이 하나의 프로그램 모듈로 동작하기 때문에 사소한 버그가 시스템 전체를 파괴할 수 있다는 단점이 있다[3].

업계에서 주로 사용되고 있는 내장형 시스템(Embedded System)의 개발 플랫폼이 되는 상용 실시간 운영체제의 예로는 WindRriver사의 pSoS와 VxWorks, Enea OSE Systems사의 OSE, LinuxWorks사의 LynxOS, Finite State Machine Labs사의 RTLinux, MS사의 Windows CE, 그리고 국내의 미지 리눅스에서 개발한 GUI를 가지는 내장형 운영체제인 리누엣(Linu@)등 여러가지가 있으며 이들을 몇 가지 특성에 의해서 분류해 놓은 내용이 〈표 1〉에 잘 나타나 있다[3]. 참고로 국내 서울대학교의 실시간 운영체제 연구실에서 개발한 실시간 운영체제인 Arx/mArx(micro Arx)[4]와 한국전자통신연구원과 다산인터네트가 공동 개발한 실시간 운영체제인 Qplus(Q+)[5]는 해당 소스까지 공개하고 있다.

〈표 1〉 상용 실시간 운영체제들과 특징[3]

운영체제	제조회사	국내대리점	로열티 정책	구 조
VxWorks	WindRiver	WindRiver Korea	○	Multi Thread
OSE	Enea OSE Systems	트라이콤텍	△	Multi Thread
VRTX	Mentor Graphic	다산인터네트	○	Multi Thread
PSoS	WindRiver	WindRiver Korea	○	Multi Thread
Nucleus Plus	Accelerated Technology	ATI Korea	×	Multi Thread
Super Task	US Software	아라전자	×	Multi Thread
μC/OS II	Micrium	디오이즈	×	Multi Thread
QNX	QNX Software Systems	다산인터네트	○	Multi Process
OS-9	Microware	Microware Korea	○	Multi Process
LynxOS	LinuxWorks	•	△	Multi Process
RTLinux	Finite State Machine Labs	•	△	Multi Process
Windows CE	MicroSoft	MicroSoft	○	Multi Process

(〈표 1〉에서 로열티 정책의 ○는 로열티를 받는다는 표시이고 △는 명확하지 않다는 표시이며 ×는 로열티를 받지 않는다는 표시이다.)

이러한 상용 운영체제에서는 고정우선순위 기반의 선점 스케줄링 기법을 사용하고 있는데 이는 대부분의 중요 태스크들이 주기적이고 선점적이며 단일 프로세서 상에서 수행된다는 가정에 바탕을 둔다. 이때 주기단조 알고리즘이나 그의 변형이 사용하기 편하며 스케줄 가능성을 예측하기 쉽다는 연구 결과들이 발표되었다. 최근 들어 스케줄링 효율이 더 좋은 최소 마감시간 우선 기법(EDF : Earliest Deadline First)을 상용 운영체제에 채용하려는 노력이 계속되고 있다.

2. 실시간 스케줄링

태스크 스케줄링이란 실행해야 할 임의의 태스크 집합에 대해서 이용 가능한 컴퓨터 자원(프로세서, 메모리, 등등)을 이용하여 태스크가 가지는 요구 조건을 만족시키면서 실행할 수 있는 순서를 정하는 일련의 과정이라고 할 수 있다. 각각의 태스크가 가지는 요구 조건은 여러 가지 항목이 있으나, 특히 실시간 시스템에서는 그것들이 실행을 종료되어야 하는 시점이 매우 중요하다. 이 종료 시점을 태스크의 마감시간이라고 한다. 태스크 집합이 적절한 스케줄링 알고리즘에 의해서 마감시간을 준수하면서 실행할 수 있으면 이 태스크 집합은 수행 가능(feasible)하다고 말한다.

실시간 시스템에서 태스크가 요구하는 마감시간을 만족시키기 위한 많은 알고리즘들이 기존에 제시되었다. 본 절에서는 이러한 알고리즘들을 설명하기에 앞서, 이 알고리즘들을 설명하는데 필요한 기본적인 용어를 정리한다. 아래에 태스크가 가지는 성질들과 그 기호를 표시하였다.

- n : 태스크 집합에 포함되어 있는 태스크의 개수
- e_i : 태스크 T_i의 실행시간
- P_i : 만일 태스크 T_i가 주기적이라면 T_i의 주기
- d_i : 태스크 T_i의 상대적 마감시간
- D_i : 태스크 T_i의 절대적 마감시간
- r_i : 태스크 T_i의 release time

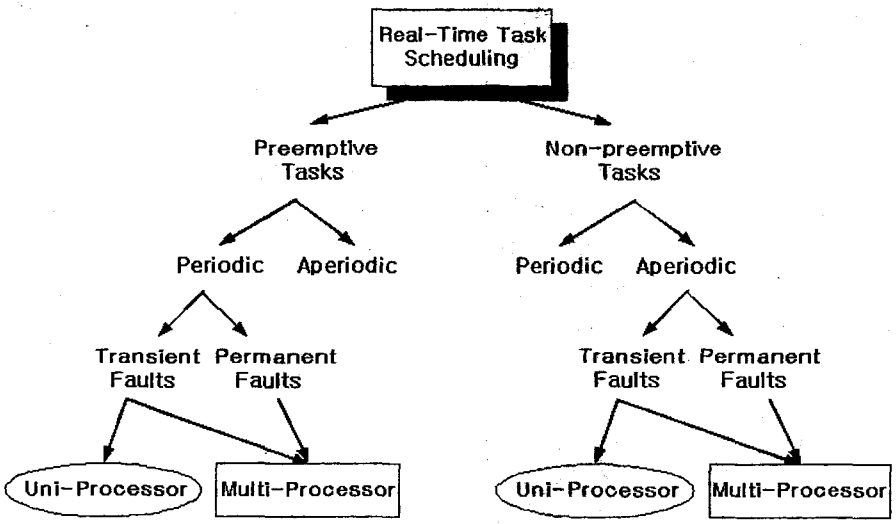
3. 실시간 스케줄링 알고리즘의 분류

실시간 시스템의 설계에 있어서 가장 중요한 문제는 태스크 스케줄링이다. 이는 태스크들을 어떻게 스케줄링하여 실행하느냐에 따라서 해당 실시간 시스템의 성능이 달라질 수 있기 때문이다. 또한 스케줄링에 있어서의 가장 중요한 문제는 태스크 모델을 어떻게 선정하느냐 하는 점이다. 실제로 실시간 시스템을 구성하는 태스크 모델의 종류에 따라서 각각 적절한 스케줄링 알고리즘을 적용할 수 있다. 〈그림 1〉은 실시간 태스크 스케줄링을 태스크의 모델과 태스크를 실행하는 프로세서의 수에 따라서 분류해 놓은 그림이다.

〈그림 1〉에서 보는 바와 같이 실시간 태스크 스케줄링 알고리즘은 크게 단일 프로세서(Uni-Processor) 상에서의 스케줄링과 다중 프로세서(Multi-Processor) 상에서의 스케줄링으로 크게 나눌 수 있다. 프로세서의 개수에 따라서 분류된 스케줄링 알고리즘은 다시 선점형(preemptive) 태스크와 비선점형(non-preemptive) 태스크 혹은 주기적 태스크냐 비주기적 태스크냐에 따라서 나누어질 수 있다. 태스크가 선점형이라는 의미는 더 높은 우선 순위를 가지는 태스크가 낮은 우선 순위를 가지는 태스크의 자원을 강제로 획득하여 실행될 수 있는 방식을 말한다. 그리고 태스크가 비선점형이라는 의미는 프로세서를 먼저 점유한 태스크가 그 실행을 종료하기 전까지 다른 태스크에게 자원을 넘겨주지 않는 방식을 말한다.

〈그림 1〉에서 각각의 태스크 모델에 따라 수개에서 수십 개의 연구들이 수행되었다. 또한 태스크 모델들이 조합된 경우에는 또 다른 스케줄링 문제가 만들어지며 이에 대한 연구도 수행되었다. 하지만 상용 실시간 운영체제에서는 이들 중 극히 제한된 일부만 사용하고 있는데 그 이유는 실제 태스크가 수행되는 환경은 연구들에서 채택하고 있는 가정들보다 훨씬 복잡하기 때문에 대부분은 이론적인 면에서만 의미가 있을 뿐이다.

따라서 본 논문에서는 최근 상용 실시간 운영체제에서 주로 채택되고 있는 단일 프로세서 상



〈그림 1〉 실시간 태스크 스케줄링 알고리즘의 분류

에서의 선점이 가능한 주기적 태스크 모델을 가정한 스케줄링 알고리즘만을 고려하기로 한다.

Ⅲ. 선점형이며 주기적인 태스크 스케줄링 알고리즘

1. 주기단조 스케줄링 알고리즘

주기단조(RM : Rate-Monotonic) 스케줄링 알고리즘은 상용 실시간 운영체제에서 가장 보편적으로 사용되고 있다. RM 알고리즘은 1) 하나의 프로세서(Uni-Processor) 상에서 2) 고정된 우선권을 가지는 프로세스들에 대한 3) 선점 방식의 스케줄링 알고리즘이다. 이 알고리즘은 Liu와 Layland에 의해서 처음 제안되었으며 태스크의 주기가 짧을수록 높은 우선 순위를 할당하는 스케줄링 방식이다[13]. RM 알고리즘의 동작을 분석하기 위해서는 다음의 A1~A5까지의 가정 사항을 필요로 한다[1,13].

A1. 어떠한 태스크들이라도 비선점적인 동작을 요구하지 않고 선점적 동작을 실행하기 위해서 필요한 비용은 무시할만하다.

A2. 하나의 태스크들을 실행하는 데에는 프로세서의 처리능력만이 중요한 요소가 되고, 메모리, 입/출력, 그리고 다른 어떠한 자원에 대한 요구는 무시할만하다.

A3. 모든 태스크들은 서로 독립적으로 실행된다. 즉, 어떤 태스크의 실행 결과가 다른 태스크의 실행에 전혀 영향을 미치지 않는다(precedence constraints).

A4. 태스크 집합에 포함되어 있는 모든 태스크들을 주기적이다.

A5. 태스크의 상대적 마감시간은 주기내의 시간에서 같은 값을 가진다.

이상과 같은 가정 중에서 특히, A5는 RM의 분석을 단순화시키는데 큰 도움을 주게 된다. RM 알고리즘에 대해서는 매우 단순한 스케줄 가능성(schedulability)에 대한 확인 방법이 Liu와 Layland에 의하여 제안되었으며 그 내용은 다음과 같다[1,6,13].

만일 스케줄 되어질 태스크들의 총 이용률(U: utilization)이 $n(2^{\frac{1}{n}}-1)$보다 크지 않으면 이 태스크들은 RM 알고리즘으로 각 태스크들의 마감시간을 준수하면서 스케줄 될 수 있다. 이때 n은 스케줄 되어질 태스크들의 개수이다. 여기서 $n\to\infty$이면, $n(2^{\frac{1}{n}}-1)$의 값은 0.693으로 수렴

한다.

$$U=\frac{e_1}{P_1}+\frac{e_2}{P_2}+\frac{e_3}{P_3}+\cdots+\frac{e_n}{P_n}+$$
$$=\sum_{i=1}^{n}\left(\frac{e_i}{P_i}\right)\leq n(2^{\frac{1}{n}}-1)$$

RM 알고리즘은 결국, 총 이용률이 0.693 이하일 때 가장 작은 주기를 가지는 태스크들에 가장 높은 우선권을 할당되면 모든 태스크들의 마감시간을 준수하면서 스케줄 가능하다는 것을 보장한다. 이때, 총 이용률이 0.693 이상인 경우에는 스케줄이 안 만들어질 수도 있는데 이는 RM 알고리즘이 최적은 아니라는 것을 말한다. 그러나 RM 알고리즘은 그 단순함과 예측 가능성 때문에 1973년 처음 제안된 이후로 계속 사용되어 왔다[13].

2. 산발 태스크 스케줄링 알고리즘

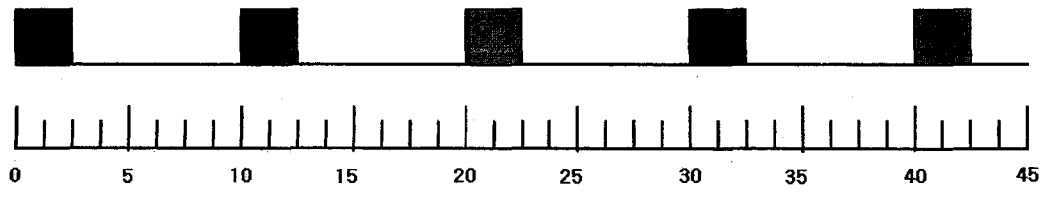
RM 스케줄링의 경우에는 주기적인 특성을 가지는 태스크들을 고려했다. 하지만 산발(sporadic) 태스크들은 주기적이지 않으며, 그것들의 준비상태에 이르는 시간이 불규칙적이다. 예들들어, 운영체제가 특정 이벤트를 받아들인 후, 입력 이벤트에 따라서 특정 태스크가 호출된다면 이 태스크는 산발태스크라고 볼 수 있다. 이 태스크들을 특징짓는 요소는 태스크들이 준비상태에 이르는 최대 비율(maximum rate)이다.

산발 태스크들을 다루는 몇 가지 방법이 존재하는데, 그 첫번째 방법은 이 태스크들을 주기적인 태스크들로 취급하는 것이며, 이때 주기는 태스크들이 release되는 최소 interarrival time으로 정한다. 이 방법은 산발 태스크들을 다루는 가장 단순한 방법이다. 산발 태스크들을 다루는 두 번째 방법[1]은 최고 우선권과 특정 값으로 선택된 실행 시간을 가지는 가상의 주기적인 태스크를 정의하는 것이다. 그리고 이 태스크가 프로세서에 의해서 실행되기 위해서 스케줄되는 동안 프로세스는 서비스 받기를 기다리는 산발 태스크들을 실행할 수 있다. 이외의 시간동안 프로세스는 주기적인 태스크들만을 서비스한다.

〈그림 2〉를 살펴보면, 이 경우의 예를 볼 수 있다. 이 경우, 가상의 최고 우선권을 가지는 태스크의 주기가 10이고 실행 시간이 2.5이다. 이 태스크는 그림의 어둡게 표시된 부분에서만 실행된다. 즉, 어둡게 표시된 부분에서만 pending되어 있는 산발 태스크를 실행할 수 있다. 만일 이 시간동안 pending되어 있는 산발 태스크가 존재하지 않으면 프로세서는 쉬게 된다. 프로세서는 어둡게 표시된 시간 이외에는 산발 태스크를 실행할 수 없으며 이 시간 동안에는 주기적인 태스크들을 스케줄하여 실행한다.

산발 태스크들을 스케줄하는 세 번째 방법은 DS(Deferred Server[1]를 이용하는 것인데, 이 방법은 두 번째 방법에 비해서 프로세스의 효율을 조금 더 높일 수 있다. DS 방법은 프로세서가 산발 태스크를 스케줄하여 실행하고자 하고, 마침 이때 서비스 받기를 기다리는 산발 태스크가 없다면 프로세서는 다른 주기적인 태스크들을 우선 순위에 따라서 실행시킨다. 하지만 주기적인 태스크를 실행하는 도중에 산발 태스크가 도착하면, 실행중인 주기적인 태스크를 밀쳐내고(preemption) 도착한 산발 태스크를 실행시키게 된다.

DS 알고리즘의 스케줄 가능성에 대한 확인은



〈그림 2〉 산발 태스크를 다루는 방법 I

RM 알고리즘과 유사하다. U_s를 산발 태스크들을 위해서 할당된 프로세서의 총 이용율이라 정의하고 이 태스크들의 relative deadline이 주기와 동일하다고 가정하자. 이러한 환경에서 만일 sporadic 태스크들을 포함한 모든 태스크들의 이용율(U)이 다음 식을 만족하면 모든 주기적인 태스크들은 DS 알고리즘으로 스케줄링 가능하다.

$$U \leq \begin{cases} 1-U_s, & \text{if } U_s \leq 0.5 \\ U_s, & \text{if } U_s \geq 0.5 \end{cases}$$

하나의 예를 들어보자, 만일 $P_s=6$이 DS의 주기이고 어떤 주기적인 태스크 T_1의 주기 $P_1=6$이라고 가정하자. 이때 sporadic 태스크의 예약된 실행 시간이 $e_s=3$이면 $U_s=\frac{3}{6}=0.5$가 된다. 만일 프로세서가 이 산발 태스크의 실행을 위해서 각 주기의 후반 3의 시간만큼 할당한다면, 주기 P_1에서 태스크 T_1을 실행하기 위해 남겨지는 시간이 부족하여 주기적인 태스크를 스케줄할 수 없게 된다.

3. 마감시간과 주기가 일치하지 않을 때

3.1절과 3.2절에서 소개된 실시간 시스템을 위한 스케줄링 알고리즘들은 태스크들의 상대적 마감시간이 해당 태스크들의 주기와 동일하다고 가정한 상태에서 동작하는 알고리즘들이다. 특히, RM 알고리즘은 고정 우선권의 주기적인 태스크들에 대해서 최적의 스케줄을 수행하는 알고리즘이었다. 하지만 마감시간과 주기가 동일하다는 가정 사항을 완화시키면 RM 스케줄링 알고리즘은 더 이상 적용될 수 없다.

만일 상대적 마감시간(d_i)이 주기보다 작은 경우를 고려해보면 태스크 T_i가 RM 스케줄 가능하기 위한 필요 충분 조건은 다음과 같다.

$$W_i(t)=t, \text{ for some } t \in [0,\ d_i]$$

하지만, $d_i > P_i$인 경우는 보다 복잡하다. 이전의 경우와 마찬가지로 RM 스케줄링 알고리즘에서 T_i가 T_j보다 높은 우선권을 가짐은 $P_i < P_j$와 필요 충분 관계에 있다. 따라서 태스크 T_i가 다음과 같은 식을 만족하면 $d_i > P_i$인 경우에도 RM 스케줄 가능하다[1].

$$t(k,\ i) < (k-1)P_i + d_i,\ \forall k \leq l_i$$
$$\text{wher } l_i = \min\{m \mid mP_i > t(m,\ i)\}$$

4. 최단마감시간우선(EDF : Earliest Deadline First) 알고리즘

EDF 알고리즘[1]은 마감시간이 가까운 태스크를 우선 스케줄링 한다는 직관을 알고리즘화한 것이다. EDF는 현재 준비상태에 있는 태스크들 중에서 마감시간이 가장 짧은 태스크를 선택하여 수행한다. 태스크들의 우선순위가 시간이 흐름에 따라 동적으로 변화하므로 동적 우선순위 기반 알고리즘이라고도 불린다. RM 알고리즘의 경우 우선순위가 한번 부여되면 더 이상 변하지 않는 고정 우선순위 방식이므로 이에 비교된다.

EDF의 가장 큰 장점은 이론적으로 총 이용률이 1 이하이기만 하면 언제나 가능 스케줄을 만든다는 것이다. 즉, EDF 알고리즘은 상기한 태스크 모델에서 최적이며 당연히 RM 알고리즘보다 더 좋은 성능을 낸다. 또한 알고리즘도 RM에 비하여 크게 복잡하지도 않다. 그러나 대부분의 상용 운영체제들은 RM 알고리즘과 같이 정적 우선순위기반의 스케줄링 기법을 채용하고 있다. 그 이유는 이론의 가정이 되는 태스크 모델과 실제 수행환경에서의 태스크들과 차이가 크다는 점이다. 즉, 태스크들의 수행시간과 마감시간, 주기 등을 정확히 예측하여야만 EDF등을 이용하여 총 이용률을 더 향상시킬 수 있는데, 실제로는 태스크 모델을 이론에서와 같이 정확히 가정할 수가 없다. 또한, RM을 사용하더라도 많은 경우 0.9 이상의 이용률에서 스케줄 가능하기도 하고 실제 시스템에서는 안전을 위하여 총 이용률을 0.5 이하로 두고 시스템을 구축하는 일도 많다.

Ⅳ. 임계 영역의 처리 (Handling Critical Sections)

3절에서 언급된 모든 실시간 스케줄링 알고리즘에서는 모든 태스크들이 그들의 수행 도중에 선점형 특성으로 인해 스왑(Swap)될 때 어떠한 영역에서라도 가능하다고 가정했다. 하지만 어떤 태스크들은 공유될 수 없는 시스템 자원의 접근을 필요로 한다. 태스크가 메모리 블록에 대해서 쓰기 연산을 수행하고 있는 중인 경우가 그 예가 될 수 있다. 이 경우, 쓰기 연산이 종료될 때까지 어떠한 다른 태스크들도 해당 메모리 블록에 접근할 수 없어야 한다. 현재 공유 불가능한 시스템 자원에 대해서 연산하고 있는 하나의 태스크는 해당 자원과 관계된 임계 영역(Critical Section)에 있다고 말한다[1].

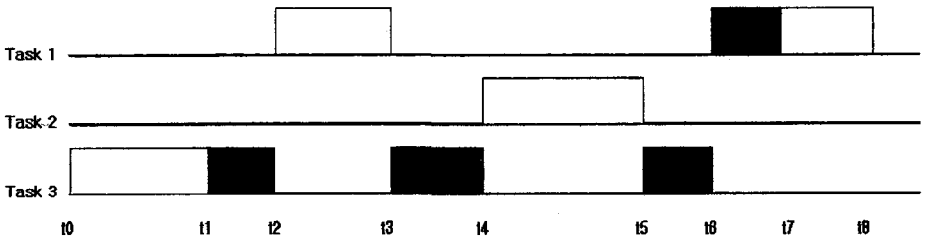
공유 자원에 대한 배타적인 접근(exclusive access)을 보장하는 한 가지 방법은 임계 영역을 이진 세마포어(binary semaphore)로 보호하는 것이다. 만일 세마포어가 잠겨 있으면(예를 들어 1로 세트되어 있음), 이는 어떤 태스크가 해당 자원을 사용하고 있으므로 다른 태스크는 이전 작업이 종료될 때까지 해당 자원을 사용할 수 없다는 것을 말한다. 따라서 태스크는 항상 임계 영역과 관련된 작업을 수행할 때에는 세마포어가 잠겨있는지 열려있는지를 확인해야 할 것이다. 또한 임계 영역 내에서 작업을 실행하는 태스크는 작업을 시작하기 이전에 해당 세마포어를 잠구어야 하고, 작업이 종료되면 잠구었던 세마포어를 열어야 한다.

1. 우선권 역전

임계 영역을 고려할 때, 하나의 태스크가 실행을 시작하면 실행 이후에 올 수 있는 동작은 크게 세 가지로 나눌 수 있다. 그 첫번째는 해당 실행을 종료하는 것이고 두번째는 높은 우선권을 가지는 태스크에 의해서 선점되는 것이고 마지막 세번째는 낮은 우선권을 가지는 태스크가 자신이 필요로 하는 임계 영역을 사용하여 잠구고 있을 때 블록되는 것이다. 물론 입/출력 작업이나 페이지 폴트와 같은 현상을 만날 수도 있겠지만 본 논문에서는 앞서 언급된 세 가지의 태스크 상태만을 고려하기로 한다.

낮은 우선권을 가지는 태스크 T_L은 특정 임계 영역을 사용하고 있을 때, 동일한 임계 영역을 필요로 하고 높은 우선 우선권을 가지는 태스크 T_H를 블록할 수 있다. 이와 같이 낮은 우선권을 가지는 태스크가 높은 우선권을 가지는 태스크를 블록시키는 동작은 "*우선권 역전(Priority Inversion)*"과 같은 좋지 않은 부작용을 초래할 수 있다[1]. 〈그림 3〉의 예를 들어보자

〈그림 3〉에 나타난 것처럼 우선권에 따라 정렬된 세 개의 태스크 T_1, T_2, T_3를 고려하자. 그리고 T_1, T_3가 공유하는 임계 영역 S가 있다. 〈그림 3〉에서 T_3는 시간 t_0에서 실행을 시작한다. 시간 t_1에 T_3는 임계 영역 S에 진입한다. T_1은 t_2에 시작하여 T_3를 선점한 후 임계 영역 S에 진입하기 전 t_3까지 수행된다. 그러나 S는 T_3에 의해서 잠구어져 있기 때문에 비록 T_1이 높은 우선권을 가지지만 블록된 채로 기다리게 된다. 시간 t_4에서 태스크 T_2가 시작되는데, 이 태스크는 T_3보다 높은 우선권을 가지므로 T_3를 선점한다.



〈그림 3〉 우선권 역전 현상

T_2는 임계 영역 S를 필요로 하지 않고, 따라서 시간 t_5에 실행이 종료된다. 이후 T_3는 그 실행을 다시 시작하고 t_6에 임계 영역 S를 빠져 나온다. 이제 T_1은 T_3를 선점하고 임계 영역 내에서 실행될 수 있다.

이상의 경우를 잘 살펴보면 임계 영역으로 인해서 T_1이 T_2보다 높은 우선권을 가지지만 종료되는 시점은 T_2가 앞서게 되는데 이러한 현상을 우선권 역전이라고 한다. 우선권 역전 현상은 임계 영역에서의 실행이 태스크의 우선순위와 관계없이 보호되기 때문에 발생하는 부작용이라고 할 수 있다.

2. 우선권 상속

"*우선권 상속(Priority Inheritance)*" 기법[1]의 사용은 임계 영역으로 인한 우선권 역전 현상을 피할 수 있게 해 준다. 이 기법에서는 임계 영역의 사용으로 인해서 높은 우선권을 가지는 태스크 T_H가 낮은 우선권을 가지는 태스크 T_L에 의해서 블록될 때, 낮은 우선권을 가지는 태스크가 일시적으로 높은 우선권을 가지는 태스크의 우선권 자체를 상속받게 된다. 그리고 블록킹 현상이 끝나고 나면 다시 원래의 우선권을 되찾게 된다. 우선권 상속 기법을 〈그림 3〉의 예에 적용하면, 시간 t_4에 태스크 T_2가 태스크 T_3를 선점하고자 할 때, 태스크 T_3는 이미 태스크 T_1의 우선권을 상속받았기 때문에 T_2에 의해서 선점당하지 않고 작업을 계속 실행할 수 있으므로 T_1과 T_2 사이에 우선권 역전 현상이 발생하지 않게 된다.

우선권 상속 기법은 불행히도 여러 개의 임계 영역에 대해서 데드락을 발생시킬 수 있다. 하나의 예를 들어보자. 두개의 태스크 T_1과 T_2가 두 개의 임계 영역 S_1과 S_2를 이용한다. 그리고 이 태스크들은 다음의 순서로 임계 영역을 필요로 한다.

- T_1 : Lock S_1 → Lock S_2 → Unlock S_2 → Unlock S_1
- T_2 : Lock S_2 → Lock S_1 → Unlock S_1 → Unlock S_2

태스크 T_2가 태스크 T_1보다 먼저 수행이 되어야 한다고 하자. 태스크 T_2는 시간 t_0에 실행을 시작하고 시간 t_1에 S_2를 잠군다. 시간 t_2에 태스크 T_1이 시작되고 이때 T_1은 높은 우선권으로 T_2를 선점한다. 시간 t_3에 T_1은 S_1을 잠군다. 그리고 시간 t_4에 T_1은 S_2를 잠구고자 한다. 하지만 이때 T_2는 작업을 종료하지 않았기 때문에 T_1은 블록되고 우선권 상속 기법에 의해서 T_2는 T_1의 우선권을 상속받는다. 그러나 시간 t_5에 T_2가 S_1을 잠구고자 할 때, T_1이 이미 S_1을 잠구고 있기 때문에 그럴 수 없다. 이때 데드락이 발생하게 된다.

3. 우선권 상한

우선권 상속 기법은 우선권 역전 현상을 회피할 수 있게 했으나 다수의 임계 영역이 포함된 실행에 있어서는 태스크들간 데드락을 발생시키는 문제점을 가진다. 우선권 상한 프로토콜[1]은 우선권 상속 기법의 데드락 문제를 해결해 준다. 세마포어의 우선권 상한이란 해당 세마포어를 잠굴 수 있는 태스크의 가장 높은 우선권을 말한다. $P(T)$는 태스크 T의 우선권을 표시하고 $P(S)$는 임계 영역 S의 세마포어의 우선권 상한 값을 나타낸다고 하자. 세개의 태스크와 네 개의 임계 영역이 주어지고 우선권이 태스크의 인덱스와 같다고 가정할 때 〈표 2〉는 한 가지 우선권 상한 값을 보여주는 예이다.

우선권 상한 프로토콜은 우선권 상속 기법과 한 가지 면을 제외하고는 동일하게 동작한다. 즉

〈표 2〉 우선권 상한 예

임계영역	접근(잠구기 가능) 태스크	Priority Ceiling
S_1	T_1, T_2	$P(T_1)$
S_2	T_1, T_2, T_3	$P(T_1)$
S_3	T_3	$P(T_3)$
S_4	T_2, T_3	$P(T_2)$

다른 점은, 만일 현재 다른 태스크(이 태스크의 우선권 상한 값이 해당 태스크의 우선권보다 같거나 크다)에 의해서 잠구어진 어떠한 세마포어가 있으면 태스크는 임계 영역에 진입하는데 있어서 블록 되어질 수 있다. 〈표 2〉에 나타난 예를 들어보자. 만일 T_2가 현재 S_2를 잠구고 있고 태스크 T_1이 시작되었다. 태스크 T_1은 S_1에 진입하는데 있어서 블록될 것이다. 왜냐하면 그 우선권이 S_2의 우선권 상한 값보다 크지 않기 때문이다. 이러한 조건으로 인해서 우선권 상한 프로토콜은 우선권 상속 기법에서 발생하는 데드락 문제를 해결한다.

V. 고장 허용성의 처리

1. 고장 허용 특성

앞서 언급했듯이, RM 스케줄링 알고리즘은 단일 프로세서 상에서 주기적인 태스크에 대해서 프로세서에 의해서 수행될 태스크의 우선 순위를 해당 태스크의 주기만을 가지고 할당하기 때문에 구현이 간단하고 쉽다. 또한 이러한 태스크 모델에 대해서는 최적의 알고리즘으로 동작한다. 따라서 RM 스케줄링 알고리즘이 실시간 운영체제의 스케줄링 알고리즘으로 가장 많이 사용되고 있다고 알려져 있다. 하지만 프로세서 내부의 동작에 있어서 고장(fault)이 발생할 경우, 시간 중복을 통한 고장 복구에 대해서는 더 이상 RM 스케줄링 알고리즘은 태스크들의 마감시간을 보장하지 못하는 경우가 발생할 수 있다. 즉, 고장을 고려할 경우에는 RM 스케줄링 알고리즘은 최적의 스케줄링 알고리즘이 될 수 없다. 실시간 스케줄링 알고리즘에서 고장을 허용할 경우, 고장의 모델을 어떻게 선정하느냐 하는 문제에 대해서도 많은 연구가 진행되고 있지만 본 논문에서는 다음과 같이 단순한 고장 모델을 가정하기로 한다.

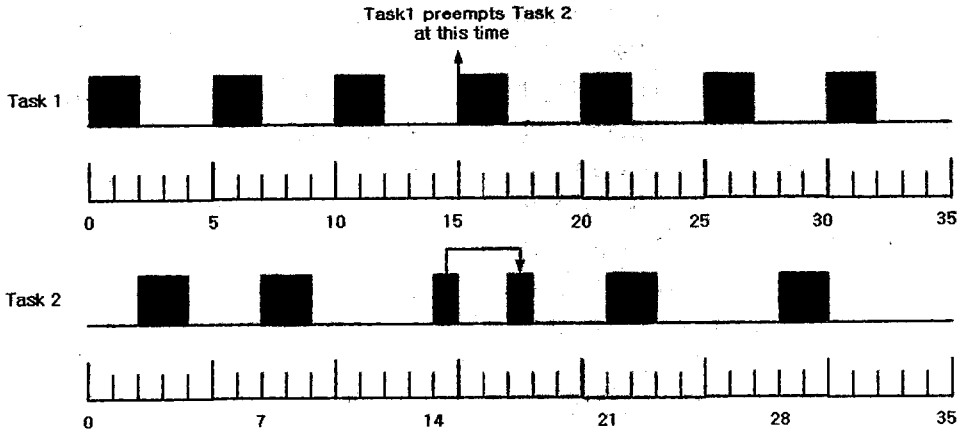
- 고장은 태스크의 실행이 끝난 직후, 다음 태스크의 시작 직전에 감지(detection)된다.
- 고장이 감지된 시점에 곧바로 복구 기능(recover action)이 실행된다.
- 프로세서상의 고장은 일시적인 고장이다.
- 고장의 복구 방법은 재수행(re-execution)이다.

〈그림 4〉는 RM 스케줄링 알고리즘으로 스케줄된 태스크들의 예를 보여주고 있으며, RM 스케줄링 알고리즘을 적용하는 과정에서 고장이 발생하지 않는 경우와 고장이 발생한 경우 스케줄링이 가능한 경우와 그렇지 않은 경우의 예를 잘 보여준다.

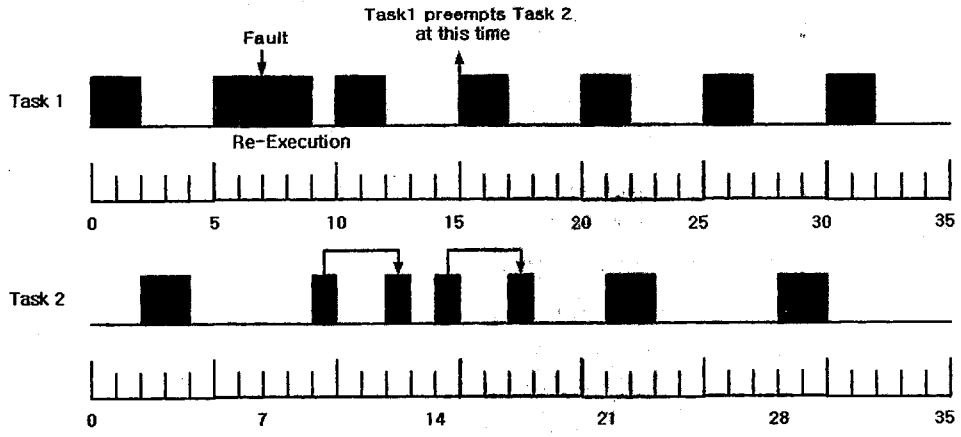
〈그림 4〉에서 태스크 T_1과 T_2의 주기는 각각 5와 7이며 실행 시간은 모두 2로 동일한 경우이다. 우선 순위는 태스크 T_1이 태스크 T_2에 비해 주기가 작으므로 더 높은 우선 순위를 가진다. 먼저 〈그림 4〉의 (a)는 고장이 발생하지 않은 경우 RM 스케줄링 알고리즘에 의해서 데이라인 조건 하에서 스케줄링 되는 것을 볼 수 있다. 즉 위의 태스크 모델에서 전체 이용율 $U=\frac{2}{5}+\frac{2}{7}=\frac{24}{35}\cong 0.686$이므로 모든 임의의 태스크 조합에서 RM 스케줄링 알고리즘이 태스크들을 스케줄할 수 있는 충분 조건인 0.828 ($n=2$)보다 작기 때문에 스케줄링 가능하게 된다.

〈그림 4〉의 (b)는 시간 7에서 고장이 발생하였음에도 불구하고 고장을 복구한 이후에 계속 스케줄링되어 정상적으로 실행되는 예를 보여주고 있다. 하지만 (c)의 경우에는 태스크 T_2가 시간 17에서 고장이 발생하여 두번째 주기에서 마감시간 21을 초과하여 실행됨으로써 실시간 스케줄링에 실패하는 모습을 보여주고 있다.

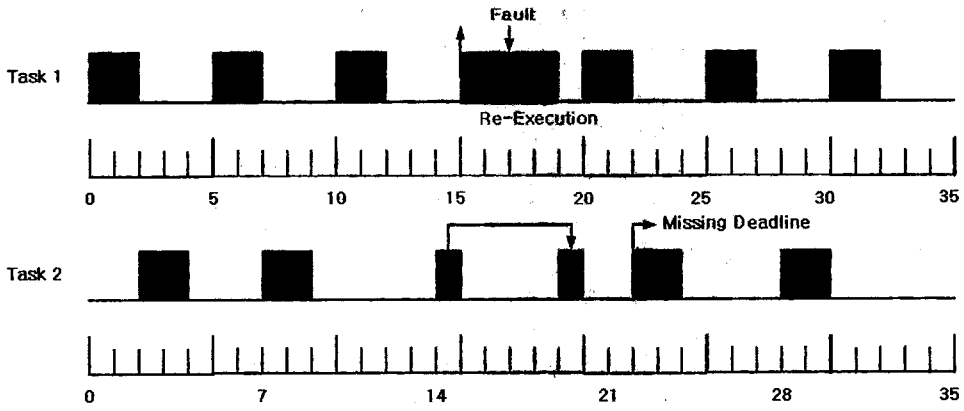
만일 태스크가 매 주기마다 고장이 발생한다고 가정하면 전체 실행 시간이 2배로 소요되기 때문에 RM 스케줄링의 전체 이용율은 $n(2^{\frac{1}{n}}-1)$의 절반으로 떨어지게 된다. 즉 전체 이용율이 $0.693/2=0.3415$ 이하일 경우에 가능한 스케줄을 만들 수 있다. 하지만 이러한 조건은 프로세서의 성능, 나아가서는 전체 실시간 시스템의 성능



(a) 고장이 없는 경우의 RM 스케줄링



(b) 고장이 발생하더라도 RM 스케줄링 가능한 경우



(c) 고장이 발생하여 RM 스케줄링이 불가능한 경우

〈그림 4〉 고장이 발생한 경우 Rate-Monotonic 스케줄링의 예[15]

율 떨어뜨리는 결과를 초래한다. 따라서 최근 이러한 이용율 한계를 확장시키려는 연구들이 수행되고 있다[6,7].

2. 고장 허용형 실시간 스케줄링 알고리즘

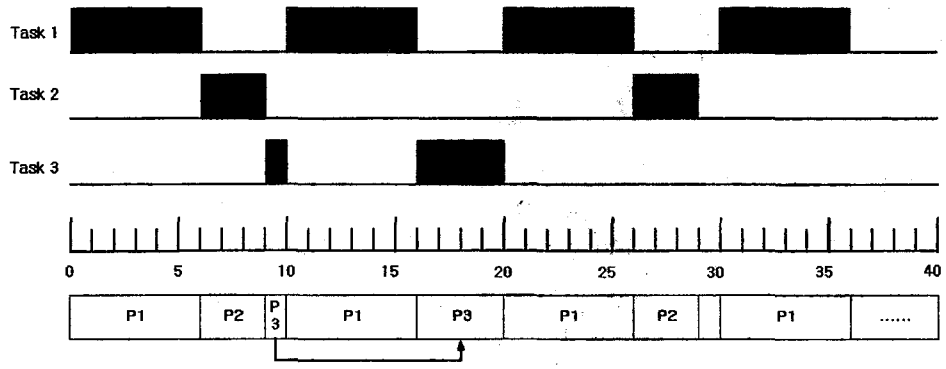
실시간 시스템의 고장 허용성에 대해서는 많은 연구 방향이 있지만 특히, RM 스케줄링 알고리즘에 고장 허용성을 추가해서 이 알고리즘을 확장하는 연구가 주된 방향이다. [6]의 연구에서는 시간 중복을 위해 재 수행을 실행하는 주기적 태스크에 대해 최소 가능한 이용율 바운드를 제시하였다. 또한 [7]의 연구에서는 고장 복구에 필요한 시간 중복을 위해 여유분 시간인 슬랙(slack)을 삽입하는 기법을 제안하였다. 그리고 이러한 시간 중복이 있을 때 주어진 태스크 집합이 모두 시간적 제약을 만족하면서 스케줄링이 가능한 이용율 바운드를 제시하였다. 그러나 두 연구에서는 고장 발생 빈도를 매 인스턴스마다 일어난다고 가정하지 않았다는 한계가 있다.

또 다른 연구에서는 각각의 태스크를 차등의 서비스 품질(QoS : Quality of Service)을 가지는 태스크로 구분하여 시간적 제약에 따른 오류를 줄이는 유연한 방안을 제시하였다. [8]의 연구에서는 모든 태스크가 주 태스크(Primary Task)와 예비 태스크(Alternative/Backup Task)의 두 태스크를 구성하고, 주 태스크는 예비 태스크에 비해 양질의 서비스를 제공하지만 마감시간에 대한 신뢰성을 보장하지는 않는다. 반면에 예비 태스크는 상대적으로 낮은 서비스 품질과 높은 신뢰성을 보장한다. 여기서 서비스 품질이 낮다는 것은 고장이 일어나지 않은 최소한의 태스크 실행을 말하며, 높은 신뢰성은 실행시간이 상대적으로 짧아 마감시간을 만족할 확률이 높다는 것을 의미한다.

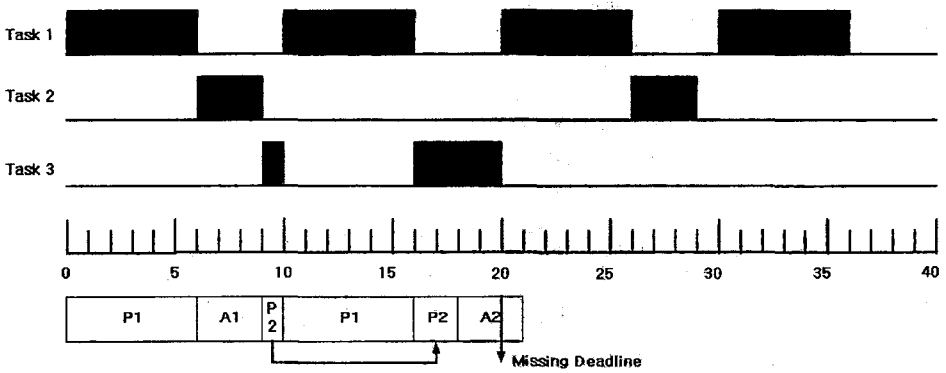
〈그림 5〉와 같은 한 가지 상황을 고려해서 설명하기로 한다. 먼저 〈그림 5〉의 (a)는 고장이 발생하지 않았을 경우, 주 태스크들 $\{(T_1, T_2, T_3)(P_1, P_2, P_3)\}$ 만으로 스케줄이 가능한 경우를 보여준다. (b)는 고장이 발생한 주기에서만 태스크가 주 태스크에서 예비 태스크로 대체되는 "*고정 스킴(Fixed Scheme)*"을 보여준다. 즉, 태스크 T_1이 첫번째 주기에서 고장이 발생할 경우, 예비 태스크 A_1으로 대체되어 마감시간을 넘기지 않았으나 두번째 주기에서 태스크 T_1이 고장이 나지 않았음에도 불구하고 태스크 T_2의 고장 발생으로 인해 태스크 T_2가 예비 태스크 A_2로 대체될 충분한 여유 시간이 부족하여 스케줄링에 실패하는 상황이 발생한다.

〈그림 5〉의 (c)에서는 고정 스킴에서의 문제점을 보완하기 위해서 고장이 발생하지 않더라도 예비 태스크로 대체되는 "*대체 스킴(Replaced Scheme)*"을 보여주고 있다. 즉, 고장이 발생하지 않는 태스크라도 신뢰성이 높은 예비 태스크로 대체하면 마감시간을 만족시킬 수 있는 확률을 높일 수 있는 방안을 나타내고 있는 것이다. 〈그림 5〉의 (c)는 대체 스킴을 적용한 스케줄 가능한 태스크 집합 중에서 최적(그림에서는 주 태스크가 4개 실행됨)의 스케줄링을 보여준다. 이는 가능한 주 태스크 수를 많이 실행시켜 서비스 품질을 높은 것을 의미한다. 그러나 이 연구에서는 예비 태스크가 고장이 발생하지 않는다는 다소 비현실적인 제약을 두고 있는 점이 연구의 부족한 면이 되고 있다.

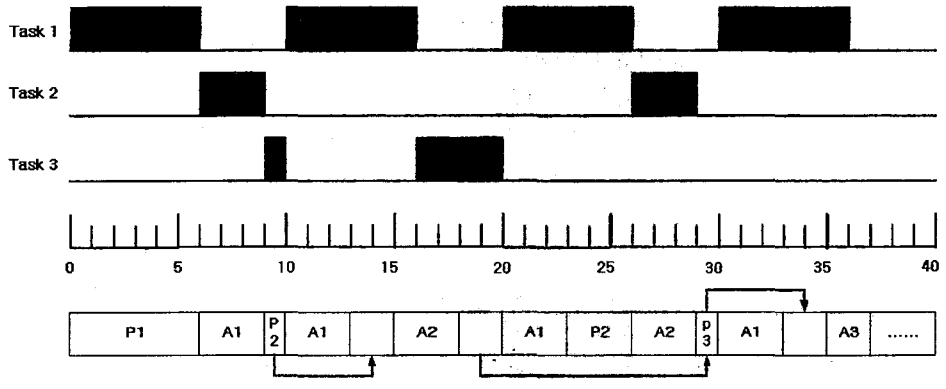
[9]와 [10]의 연구에서는 가능성과 고정 허용성에 대한 연구로 부정확한 계산(imprecise computation) 모델을 제시하였다. 이 모델에서 모든 각각의 태스크는 필수(mandatory) 작업 영역과 선택(optional) 작업 영역으로 나누어지고 스케줄링은 필수 작업 영역이 먼저 실행되고 나머지 선택 작업 영역은 데이라인의 정도에 따라 부정확 계산에 의해 실행된다. 이러한 방안의 장점은 적절한 시간 제약과 서비스 품질간의 타협(tradeoff)을 통해서 이루어지며 점진적인 성능 저하(graceful degradation)가 가능하다는 점이다. 모든 태스크의 필수 작업 영역(이 영역은 매우 작다고 가정)은 반드시 실행시킴으로써 최소한의 성능 수준(performance level)을 보장하고 가능한 많은 수의 선택 작업 영역을 실행시켜 오류를 최소화한다. 그러나 이 방안은 선택 작업 영역 수행이 수행 이전에 결정되기 때문에 응



(a) 고장이 발생하지 않은 경우



(b) 고정 스킴(Fixed Scheme)



(c) 대체 스킴(Replaced Scheme)

〈그림 5〉 세개의 태스크 에 대한 고장 허용 알고리즘들의 적용[15]

답 스케줄링에서는 고장 처리 태스크의 수행이 수행 중에 확률적으로 결정되므로 근본적인 차이점이 있다.

이와 같이 고장 허용성을 가지는 RM 스케줄링 알고리즘을 제안하는 기존의 연구들은 스케줄링이 가능한 태스크 집합의 조건을 제시하였다. 이 조건은 주어진 태스크 집합이 100% 스케줄링이 가능한지 아닌지를 판단하는 척도가 된다.

VI. 결 론

본 논문에서는 현재 상용운영체제에서 많이 사용되는 스케줄링 기법들에 대하여 살펴보았다. 태스크 모델에 따라 무수히 많은 스케줄링 기법들이 연구되었으나 대부분은 이론적 연구에 그치고 실제 상용 시스템에서 사용되는 기법들은 소수이다. 그 이유는 대부분의 연구들이 태스크의 특성을 여러 가지 모델로 가정하는데 이들이 대부분 현실에 적용하기에는 너무 단순하기 때문이다. 현재 상용 운영체제들은 RM 알고리즘에 기반을 둔 고정우선순위 선점 스케줄링 기법을 기본 스케줄러로 채택하고 있으며 이를 일부 보완하여 사용하는 정도이다.

본 논문에서는 우선 상용 운영체제들의 종류를 나열하고 이들을 멀티 쓰레드와 멀티 프로세스 모델로 분류하여 기술하였다. 또한 실시간 스케줄링의 용어를 소개하고 기존의 실시간 스케줄링 기법들을 분류하여 기술하였다. 기존의 스케줄링 기법들 중에서 대부분의 상용 운영체제에서 채용하고 있는 고정우선순위기반 선점 스케줄링 기법과 RM 알고리즘을 소개하고 좀 더 복잡한 경우에 이를 수정하는 기법들을 소개하였다. RM보다 스케줄링 성능이 더 좋은 것으로 알려진 EDF 알고리즘에 대하여 소개하고 이의 장단점을 분석하였다. EDF 알고리즘은 RM 만큼 단순하고 성능은 최적이지만 마감시간과 같은 태스크들의 특성을 실제로 예측하기 어렵고 그 외의 태스크 특성도 예측하기 어렵다는 점 때문에 그보다 단순한 RM을 선호하는 것으로 분석된다.

4장에서는 운영 환경에서 실제로 발생하고 있는 우선순위역전 문제를 기술하고 이를 해결하기 위한 우선순위 상속기법과 우선순위 상한기법을 소개하였다. 최근의 상용 운영체제들은 우선순위 상속기법을 기본적으로 제공하고 있으며 우선순위 상한기법을 지원하는 경우도 찾을 수 있다. 우선순위 상한기법이 무한대기문제를 해결할 수 있으나 사용자가 태스크가 사용하는 임계 영역들을 미리 제공해야 함으로써 운선순위 상속기법이 더 편리하다고 할 수 있다.

끝으로 최근 들어 연구되고 있는 고장을 허용하면서 실시간으로 스케줄링하는 기법들을 소개하였다. 고장허용 분야와 실시간 분야는 그 동안 독자적으로 연구되는 경향이었으나 최근에 둘을 통합하려는 움직임이 가속화되고 있다. 두 분야는 실시간 시스템의 동전의 양면 같은 것인데 그 동안 연구가 분리돼 있었다. 소개된 고장허용 실시간 스케줄링 기법들에서는 수행 중 발생하는 동적인 고장들을 고려하여 태스크들을 RM 기반이나 EDF 기반으로 스케줄링하며 총 이용률을 높이면서 스케줄 가능성을 높이는 방향으로 연구를 진행하고 있다.

참 고 문 헌

〔1〕 C. M. Krishna, and Kang G. Shin, "Real-Time Systems", McGRAW-HILL International Edition, 1997

〔2〕 Xu, J., and D. L. Parnas, "Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusive Properties", IEEE Trans. Software Engineering 16 : 360~369, 1990

〔3〕 http://www.nexto.co.kr/rtos/kind.htm

〔4〕 http://arx.snu.ac.kr

〔5〕 http://embenix.com/qplus/

〔6〕 M. Pandya and M. Malek, "Minimum Achievable Utilization for Fault-tolerant Processing of Periodic Tasks", Technical Report TR94-07, Univ. of Texas Austin, Dept. of Computer Science, 1994

〔7〕 S. Ghosh, R. Kelhem, D. Mosse, and J. Sansarma, "Fault Tolerant, Rate Monotonic Scheduling", Journal of Real-Time Systems, Vol. 15, No. 2, September 1998.

〔8〕 A. L. Liestman, and R. H. Campbell,

"A Fault-Tolerant Scheduling Problem", IEEE Trans. Software Engineering, Vol. SE-12, No. 11, pp.1089~1095, November, 1986.

[9] J. A. Stankovic, and Fuxing Wang, "The Integration of Scheduling and Fault Tolerance in Real-Time Systems", TR92-49, Univ. of Massachusetts, Amherst. Dept. of Computer Science, 1992.

[10] R. Bettati, N. S. Bowen, J. Y. Chung, "On-Line Scheduling for Checkpointing Imprecise Computation", Proceedings of the 15th Euromicro Workshop on Real-Time Systems, pp.238~243, June, 1993.

[11] Bannister, J. A., and K. S. Trivedi, "Task Allocation in Fault-Tolerant Distributed Systems", Acta Information 20 : 261~281, 1983

[12] Baruah, S. K., A. K. Mok, and L. E. Rosier, "Preemptive Scheduling Hard-Real-Time Sporadic Tasks on One Processor", Proceedings of the IEEE Real-Time Systems Symp., pp. 182~190, IEEE, Los Alamitos, CA, 1990

[13] C. L. Liu, and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of the Association for Computing Machinery, Vol. 20, No. 1, pp.46~61, 1973.

[14] Lehoczky, J. P., "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", Proceedings of the IEEE Real-Time Systems Symp., pp.201~210, IEEE, Los Alamitos, CA, 1990

[15] 강상규, 응답 시스템에서의 Rate-Monotonic 스케줄링, 석사학위논문, 한남대학교, 2000.

[16] 은성배, 송효정, 맹승렬, 조정완, "실시간 단일 처리기에서 비주기 태스크의 시간 제약과 결함 허용을 최적화하는 스케줄링", Journal of KISS, Vol. 24, No. 6, pp. 553~558, June, 1997.

[17] Lehoczky, K. P., L. Sha, and Y. Ding, "The Rate Monotonic Algorithm: Exact Characterization and Average-Case Behavior", Proceedings of the IEEE Real-Time Systems Symp., pp. 166~171, IEEE, Los Alamitos, CA, 1990

저 자 소 개

殷 誠 培

1981년 3월~1985년 2월 서울대학교 컴퓨터공학과 (공학사), 1985년 3월~1987년 2월 한국과학기술원 전자전산학과 전산학전공 (공학석사), 1987년 3월~1990년 2월 한국전자통신연구원 운영체제 개발실 연구원 1990년 3월~1995년 2월 한국과학기술원 전자전산학과 전산학 전공 (공학박사), 1995년 3월~현재 : 한남대학교 정보통신멀티미디어 공학부 부교수, 〈주관심 분야 : 실시간 운영체제, 멀티미디어 시스템〉

陳 成 琇

1990년 3월~1995년 2월 서강대학교 전자계산학과 (공학사), 1995년 3월~1997년 2월 한국과학기술원 전자전산학과 전산학전공 (공학석사), 1997년 3월~2002년 8월 한국과학기술원 전자전산학과 전산학전공 (공학박사), 2002년 7월~현재 : 삼성전자 정보통신 네트워크 사업부 초고속 통신 사업팀 책임연구원, 〈주관심 분야 : 임베디드 시스템, 초고속 IP 라우터, 인터넷 서비스 품질〉