다중작업 분할처리를 위한 적응형 스케쥴링 기법

고정환* · 김영길**

The technique of an adaptive scheduling for a multi-tasking separation

Jeong-hwan Go* · Young-kil Kim**

요 약

프로그램의 복잡화와 대규모프로그램의 등장으로 다중작업을 분할하여 소규모 단위의 타스크(Task)로 나누고 각각의 타스크를 우선순위에 따라 스케쥴링을 수행해야하는 요구가 점점 확대되고 있다. 또한, 프로그램 개발환경의 다양화로 인하여 프로그램을 구현하다 보면 다양한 환경 조건에 맞추어 개발하게 된다. 예를 들어 임베디드(Embedded) 환경인지 윈도우즈(Windows) 환경인지에 따라 다르고 운용체제의 사용에 따라서도 제약사항을 가져오는 경우가 많다. 이에 개발환경과 운용체제에 의존적이지 않도록 다중작업 분할처리를 수행할 수 있는 적응형 스케쥴링 기법을 소개한다. 본 논문에서는 적응형 스케쥴링 기법에 적용된 알고리즘에 대한 설명과 구현 후 적용한 사례를 기반으로 한 내용을 다룬다.

ABSTRACT

As the substantial increment in program complexity and appearance of mega program, the programs need to be divided to small tasks with multiple partitions and performed with a priority based scheduling. And also, a program development has to be progressed according to diversify of development environment. For instance, there are some restrictions upon O/S environment such as embedded O/S or windows. Therefore, the adaptive scheduling technique which performs multiple task partitioning process, regardless environment or O/S, is suggested. In this study, In this study, the adaptive scheduling technique algorithm and its applied examples are described.

키워드

스케쥴링, 미들웨어, 타스크, 운영체제, 큐

Key word

Scheduling, Middleware, Task, OS, Queue

* LIG넥스원(주) 책임연구원 (주저자, kojh2010@lignex1.com)
** 아주대학교 전자공학과 교수 (교신저자)

접수일자 : 2009. 06. 08
심사완료일자 : 2010. 07. 25

Ⅰ. 서 론

프로그램의 복잡화와 각종 소프트웨어 언어 및 도구 등의 발전으로 인하여 대규모의 프로그램이 점점 많아지고 있다. 이러한 이유로 실시간처리가 중요한 임베디드(Embedded) 시스템에 주로 적용되었던 스케쥴링 알고리즘이 대규모 프로그램의 효율성과 모듈화된 처리를 위하여 윈도우즈 기반의 비실시간 프로그램등에도 많이 쓰여지고 있는 추세이다.[1]

현재 여러가지 플랫폼에서 다양한 스케쥴링을 지원하는 운영체제(OS) 및 도구가 많지만 다양한 하드웨어의 특성과 사용되는 운영체제의 종류에 따라 호환성이 부족하기 때문에 하드웨어 또는 운영체제와 어플리케이션(Application) 간의 사이를 매끄럽게 연결하기 위하여 여러 형태의 소프트웨어적인 계층을 두기도 한다.

이러한 소프트웨어의 복잡한 계층구조 속에서 효과적으로 작업을 분할하고 스케쥴링하며 운영체제 환경 및 여러가지 플랫폼에 영향을 받지 않고 커널레벨이 아닌 프로그램의 계층인 미들웨어(Middleware) 또는 어플리케이션의 어디에서나 다중 타스크(Task)를 처리하기 위해 스케쥴링을 수행 수 있는 적응형 스케쥴링 기법을 소개한다.

적응형 스케쥴링 기법은 효과적인 타스크의 처리를 위해 우선순위 기반의 스케쥴링 알고리즘을 적용하며 본 논문에서는 실제 모듈화하여 적용하고 검증된 사례를 바탕으로 그 효과를 입증한다.

Ⅱ. 적응형 스케쥴링 기법의 개념 및 구조

2.1. 기존의 스케쥴링 방식

프로그램을 구현할 때 실시간운영체제(RTOS)를 사용하는 경우에는 [그림1] 에서와 같이 운영체제의 커널에서 제공하는 스케쥴러에 의해 사용자가 만든 타스크(또는 프로세스, 쓰레드)가 실행되며 운영체제와 스케쥴링 방식에 따라 또 다른 시스템에 동일한 어플리케이션을 적용하기 힘들뿐만 아니라 동일한 실행 동작을 예측하기 힘들다.

운영체제를 적용하지 않을 경우는 하나의 타스크로 모든 처리를 수행하는 형태로 프로그램하거나 별도의 스케쥴러를 개발하여 사용하는 경우가 많다.

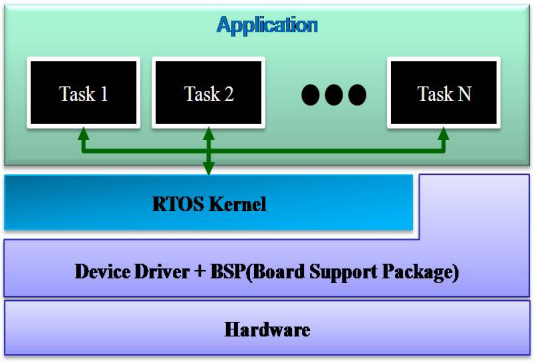

그림 1. RTOS를 사용한 소프트웨어 계층구조
Fig. 1 S/W Structure of used RTOS

2.2. 적응형 스케쥴링 기법의 개념

적응형 스케쥴링 기법은 기존에 운영체제를 사용하지 않는 단일 타스크 프로그램에서 운영체제 커널의 스케쥴러의 역할을 대신하며 운영체제를 사용하는 멀티타스킹(Multi-Tasking) 프로그램에서도 운영체제 레벨에서의 스케쥴링 이외에 어플리케이션 레벨에서도 타스크를 분할하여 별도의 스케쥴링을 수행할 수 있는 기법이다.

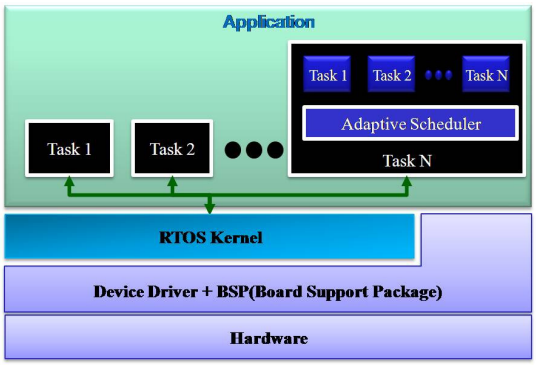

그림 2. 적응형 스케쥴러가 적용된 구조
Fig. 2 Structure of the applied adoptive scheduler

적응형 스케쥴링 기법은 [그림2]와 같이 운영체제가 제공하는 타스크 내에서 또 다른 작은 단위의 타스크를 분할하여 스케쥴링하고 싶은 경우에 타스크 내부에 적응형 스케쥴링 기법을 적용하면 타스크를 여러개의 더 작은 단위의 타스크로 쪼개어 스케쥴링 할 수 있다.

2.3. 적응형 스케쥴러의 구성

적응형 스케쥴러의 구성은 [그림3]에서 보는바와 같이 타스크의 스케쥴링을 위한 3가지 형태의 스케쥴러와 Interface Controller, Message Handler, Queue Controller로 나뉜다.

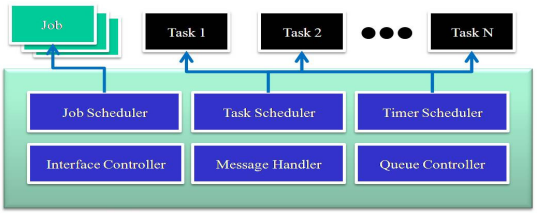

그림 3. 스케쥴러의 구성 및 타스크 실행 개념
Fig. 3 Composition of scheduler and concept of task execution

Task Scheduler는 비선점형 방식으로 스케쥴링을 수행하며 3가지 형태의 스케쥴링 방식을 지원한다. 3가지의 스케쥴링 방식은 Round Robin, Fixed Priority, Changable Priority 방식으로 적응형 스케쥴링 기법에 특화된 스케쥴링을 수행한다. 이 스케쥴링 방식은 사용자의 설정에 의하여 선택적으로 사용할 수 있다.

Job Scheduler는 기본적인 타스크의 동작 중에 인터럽트에 의하여 급하게 수행되어야 할 이벤트 발생시 인터럽트 루틴상의 빠른 처리를 위해 수행되는 스케쥴러로 스케쥴링을 위한 우선순위 플래그(Priority Flag)가 존재하고 각각의 플래그 마다 우선순위를 가지며 각각 하나의 긴급작업(Job)과 연결된다. 일반적으로 다른 스케줄러보다 더 높은 우선순위로 동작한다.

Timer Scheduler는 일정시간 후에 실행해야 할 타스크가 있을 때 프로그램에서 계속 기다리지 않고 원하는 시간 뒤에 메시지를 전달하여 기능을 수행할 수 있는 스케쥴러이다. 더 높은 우선순위의 타스크 수행 때문에 낮은 우선순위의 타스크가 CPU로부터 자원할당을 받지 못하는 CPU 기근현상(CPU starvation)을 해결 할 수 있는 방법으로 시간이 지난 후에도 원하는 상태가 되지 않았을 때 다시 타이머를 등록하면 된다.

Interface Controller는 적응형 스케쥴러와 외부모듈간의 인터페이스를 담당한다. 운영체제를 적용하지 않는 임베디드 시스템에서는 사용자가 인터페이스를 정의하여 미들웨어로도 활용할 수 있으며 어플리케이션 및 타스크가 실제 하드웨어 또는 디바이스 드라이버와 인터페이스를 수행하도록 제어 할 수 있고, 다른 미들웨어 및 프로그램 모듈과도 연동할 수 있다. Interface Controller는 사용자가 실행환경에 따라 인터페이스를 정의하거나 새로이 프로그래밍 할 수 있다.

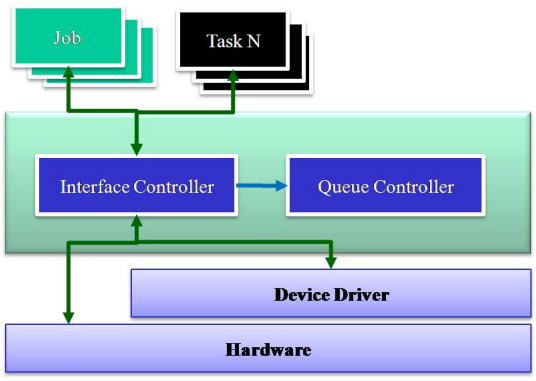

그림 4. Interface Controller의 동작 개념도
Fig. 4 Operational concept of the Interface Controller

Message Handler는 외부로 부터의 메시지에 대하여 어느 타스크에서 처리되는지를 설정하면 해당 타스크로 메시지를 전달해 주는 역할을 수행하고 타스크간의 메시지를 주고받을 때 원할 한 동작을 지원한다.

Queue Controller는 타스크에서 사용되는 수신 큐(Queue)의 동작을 제어한다. 각각의 타스크는 생성시 하나의 수신 큐를 갖게되며 수신 큐는 FIFO(First In First Out) 방식의 회전 큐로 구성된다.

2.4. 스케쥴링 알고리즘

적응형 스케쥴링 기법의 스케쥴링 알고리즘은 시스템에 종속되지 않고 어플리케이션 레벨의 원활한 다중작업 분할처리를 위해 비선점형(Non-Preemptive) 스케쥴링 방식을 적용한다. 이때 스케쥴링의 대상이 되는 타

스크는 이벤트나 메시지에 의해 수행되는 기능적으로 유사한 모듈들의 집합이며 타스크 생성시 하나의 수신 큐를 갖게 된다. 스케쥴러는 타스크의 수신 큐에 메시지가 수신되면 스케쥴링 알고리즘에 의해 타스크를 실행시킨다.

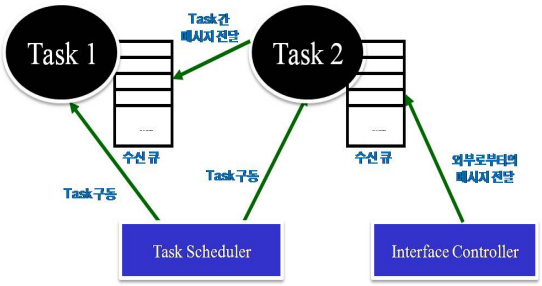

그림 5. 타스크 동작 개념도
Fig. 5 Operational concept of the Task

타스크의 스케쥴링은 Task Scheduler에서 수행하며 Round Robin, Fixed Priority, Changable Priority의 방식을 지원한다. 이는 일반적인 비선점형 스케쥴링 방식의 FIFO(First In First Out) 형태의 스케쥴링 기반하에 우선순위 스케쥴링을 접목한 방식이다.

Round Robin 스케쥴링은 우선순위가 없이 순서대로 타스크가 수행되는 방식으로 특별히 순위가 높은 타스크가 없는 경우에 주로 사용된다.

Fixed Priority 스케쥴링 방식은 타스크의 생성시 정의된 Priority 순위에 따라 스케쥴링이 일어나는 방식이다. Fixed Priority 스케쥴링 방식에서는 스케쥴링을 관리할 수 있는 우선순위 플래그를 두며 Queue Controller에 의해 큐에 메시지가 저장 될 때 플래그는 활성화된다. 또한 타스크가 수신 큐에 저장된 모든 메시지 처리가 완료되면 플래그는 비활성화된다. 이때 Task Scheduler는 타스크에 하나의 메시지 처리가 끝날 때마다 새로운 더 높은 우선순위의 타스크를 실행시킨다.

Changeable Priority 방식은 타스크의 생성시 정의된 Priority 순위를 기본으로하여 스케쥴링이 프로그램 실행도중에 우선순위를 바꿀 수 있는 스케쥴링 방식이다. 큐에 메시지가 저장될 때 우선순위를 변경을 Queue Controller가 감지하면 Task Scheduler는 다시 스케쥴링을 수행하고 그 결과로 최상위 우선순위 타스크를 선정하여 실행시키는 방식이다.

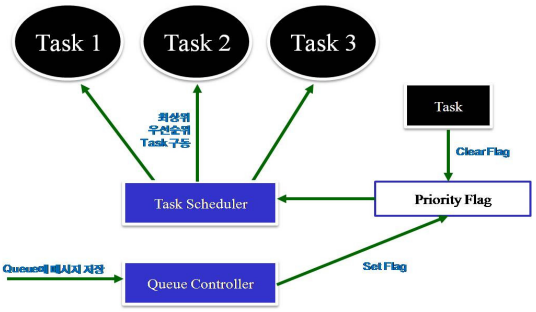

그림 6. Fixed-Priority 스케쥴링 개념도
Fig. 6 Concept of the Fixed-Priority scheduling

Ⅲ. 구현 및 검증

구현된 적응형 스케쥴링 알고리즘 동작의 검증을 위하여 [표1]과 같은 타스크와 우선순위를 등록하고 이벤트 발생에 맞추어 이벤트가 발생하도록 테스트 프로그램을 구성하였으며 실제 타스크 실행 상태와 각 타스크별 수신 큐의 상태를 구현한 GUI화면에서 확인한다.

표 1. 타스크 우선순위 및 설정값
Table. 1 Priority of task and setting value

Task	Priority	Arrival Time	ServiceTime
Task4(1)	4	0ms	90ms
Task1(1)	1	50ms	60ms
Task3(1)	3	80ms	30ms
Task5(1)	5	130ms	90ms
Task7(1)	7	150ms	60ms
Task8(1)	8	210ms	30ms
Task2(1)	2	250ms	120ms
Task6(1)	6	300ms	60ms
Task7(2)	7	320ms	30ms
Task4(2)	4	370ms	90ms
Task2(2)	2	400ms	30ms
Task3(2)	3	400ms	90ms
Task5(2)	5	450ms	120ms
Task6(2)	6	500ms	90ms
Task8(2)	8	550ms	60ms
Task1(2)	9	590ms	60ms

Round Robin 스케줄링 방식으로 [표1]에 구성한 설정값을 적용하여 실험한 실행결과는 [그림7]과 같다. 발생된 타스크의 이벤트에 따라 Task1부터 계단형태로 실행되는 것을 알 수 있다.

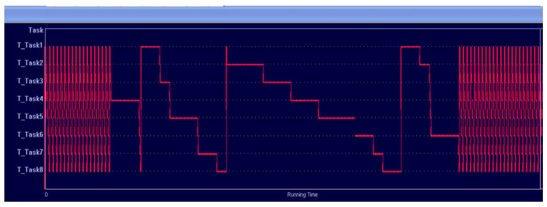

그림 7. Round Robin 스케줄링 실행 결과
Fig. 7 Execution result of Round-Robin scheduling

Fixed Priority 스케줄링 방식으로 [표1]에 구성한 설정값을 적용하여 실험한 실행결과는 [그림8]와 같다. 발생된 타스크의 이벤트에 따라 기본 우선순위가 가장 높은 타스크부터 우선하여 실행되는 것을 알 수 있다.

그림 8. Fixed Priority 스케줄링 실행 결과
Fig. 8 Execution result of Fixed-Priority scheduling

Changeable Priority 스케줄링 방식으로 표 2에 구성한 설정값을 적용하여 실험한 실행결과는 [그림9]와 같다.

그림 9. Changeable Priority 스케줄링 실행 결과
Fig. 9 Execution result of Changeable-Priority scheduling

발생된 타스크의 이벤트에 따라 기본 우선순위가 가장 높은 Task8부터 Fixed Priority 스케줄링 방식과 유사하게 실행되지만 Task1(2)의 이벤트가 최상위 우선순위로 발생하는 순간 Task1(2)가 가장먼저 실행되는 것을 알 수 있다. 타스크의 우선순위별 응답시간 분석결과는 [그림10]과 같다.

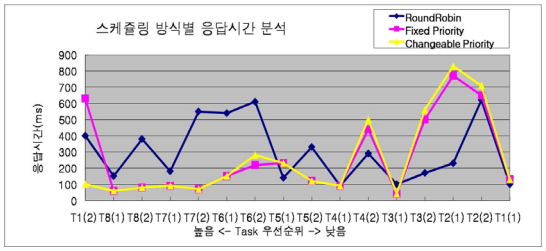

그림 10. 스케줄링 방식별 응답시간 분석 그래프
Fig. 10 Response-time analysis graph according to scheduling methode

[그림10]의 그래프에서 보는 바와 같이 좌측에 우선순위가 높은 타스크일수록 응답시간이 빨라야하는 관점에서 다음과 같이 이벤트에 대한 우선순위 Factor를 p라고하고 높은 것부터 9~1사이 값을 설정하며, 각각의 이벤트에 대한 응답시간을 t, 그리고 그에 따른 스케줄링 효율을 E라고 보면 그 공식은 다음과 같다.

$$E = \frac{p_1}{t_1} + \frac{p_2}{t_2} + \frac{p_3}{t_3} \cdot \cdot \cdot + \frac{p_n}{t_n} = \sum_{k=1}^{n} \frac{p_k}{t_k}$$

[표2]에서 보는바와 같이 우선처리 관점에서의 스케줄링 효율은 Changable Priority 방식이 제일 좋은 것을 알 수 있다.

표 2. 스케줄링 방식별 우선처리 효율
Table. 2 Effect of priority-process according to scheduling methode

스케줄링방식	효율 (E)
Round Robin	0.3481
Fixed Priority	0.7039
Changeable Priority	0.7716

Ⅳ. 적용 사례

적응형 스케쥴링 기법은 방위산업 분야에 적용하여 개발 중에 있으며 그 첫 번째 적용사례는 [그림11]과 같이 대잠전 통제 콘솔의 운용자통제를 위한 영상부 소프트웨어에서 적용된 사례이다. 영상부 소프트웨어는 윈도우즈 운영체제 기반에서 동작하는 프로그램으로 전술화면을 관리하는 부분이외의 나머지 트랙관리에서 발사통제에 이르는 일련의 과정을 적응형 스케쥴링을 사용하여 Fixed Priority 기반의 스케쥴링 한다.

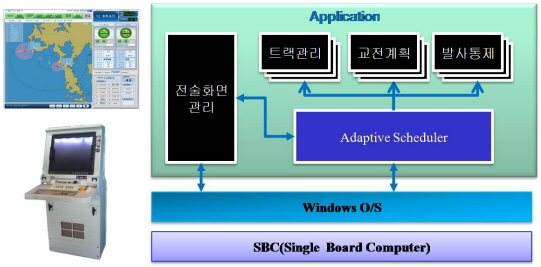

그림 11. 대잠전 통제 콘솔 적용 사례
Fig. 11 Adapted example of anti-submarine control console

이때 Interface Controller는 운영체제와 GUI중간에서 어플리케이션을 연결해주는 역할을 수행한다.

두 번째 적용사례는 운영체제를 사용하지 않은 시스템에 적용된 사례이다. [그림12]에서 보는바와 같이 항공용 전자전 장비의 OFP(Operational Flight Program)에 적용되었으며 전자전 장비를 통제하고 제어하는 소프트웨어로 개발된 SBC(Single Board Computer)에 탑재되어 있다.

적응형 스케쥴러는 이 시스템에 운영체제 커널의 역할을 대신하여 타스크의 스케쥴링을 수행한다. Interface Controller는 미들웨어로서의 역할도 수행하도록 구현하였으며 어플리케이션이 탑재되어 실행되는 SBC 환경과 디버깅을 위해 일반컴퓨터 환경에도 어플리케이션이 동작 할 수 있게 Interface Controller를 구성하였다.

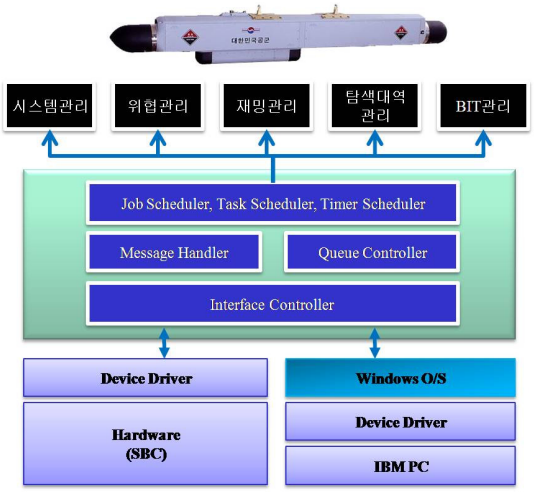

그림 12. 항공용 전자전 장비 적용 사례
Fig. 12 Adapted example of air-electronic-warfare system

Ⅴ. 결 론

소프트웨어는 여러가지 계층구조를 갖고 있으며 그 속에서 각각의 정해진 타스크(또는 프로세스)를 수행하게 된다. 지금까지 타스크의 스케쥴링은 운영체제의 커널에서 수행하는 것이 일반적이지만 적응형 스케쥴링 기법을 사용하면 프로그램내의 어디서나 스케쥴링이 가능한 형태로 구성할 수 있다. 스케쥴링 알고리즘은 비선점형 방식을 이용하여 선점형 방식에 비해서는 실시간성이 부족하지만 Changeable Priority 방식을 이용하여 좋은 성능의 응답시간을 얻을 수 있고 환경에 따라 또 다른 스케쥴링 알고리즘을 구현하여 추가할 수 있다.

적응형 스케쥴링 기법은 실제 적용한 사례에서 보았듯이 임베디드 환경 및 윈도우즈환경에서도 시스템에 구애받지 않고 타스크의 스케쥴링을 구현 할 수 있으며 Interface Controller의 적용에 따라 여러 가지 환경의 변화에 적응할 수 있다. 또한 이중화된 스케쥴링의 구현 및 다양한 형태의 스케쥴링 그룹을 형성할 수 있는 장점도 있다.

참고문헌

[1] 인치호, “실시간 제약 커널 환경하에서의 이중 실시간 스케쥴링 설계”, 전력전자학회 논문집, 2001

[2] WindRiver, "VxWorks Programmer's Guide" , www.windriver.com, 2002.

[3] 정덕영, Windows 구조와 원리 그리고 Codes, 가남사, 470p, 2003.

[3] Jean J. Labrosse, "microc/OS-II The Real-Time Kernel", R&D Books, 498p

저자소개

고정환 (Jeong-hwan Go)

1995년 서울산업대학교 전자공학과 공학학사
2010년 아주대학교 전자공학과 공학석사

1995년~현재 LIG넥스원(주) 책임연구원
※관심분야 : 운영체제, Embedded System, 상호운용성 평가

김영길 (Young-kil Kim)

1978년 고려대학교 전자공학과 공학학사
1980년 한국과학기술원 산업전자공학과 공학석사

1984년 ENST(프랑스) 공학박사
1984년~현재 아주대학교 전자공학과 교수
※관심분야 : 마이크로파 공학, 의료공학, Embedded System