MicroC/OS - II 기반에서 Multi-Level 스케줄링의 설계 및 구현

임보섭° 이재윤 김광 허신
한양대학교 컴퓨터 공학과
{bslim°, jylee, kkim, shinheu}@cse.hanyang.ac.kr

Design and Implementation of Multi-Level scheduling on MicroC/OS-II

Bosub Lim° Jaeyoon Lee Kwang Kim Sin Heu
Department of Computer Science and Engineering, Hanyang University

요 약

임베디드 시스템은 범용 컴퓨팅 시스템과 달리 자신을 포함하고 있는 기기에 부과된 특정 목적의 컴퓨팅 작업만을 수행한다. 이 시스템을 제어하기 위해서 운영체제가 필요로 하며, 임베디드 환경에서는 신뢰성과 정확성을 요하는 부분이 많기 때문에 실시간 운영체제를 필요로 한다. Real-Time kernel을 기반으로 하는 MicroC/OS-II는 수많은 용도로 사용되고 있지만 task 사용에 한계가 있다. 이 논문에서 제안하는 스케줄링은 task의 생성 수를 늘려주지만, 이 경우 task간의 우선순위 설정이 어려워진다. 이 문제 해결을 위해서 task들의 우선순위 결정은 deadline을 이용하여 3레벨로 나눈다. 3레벨로 나누어지면 task의 수가 증가해도 개발자는 task들을 레벨에 맞게 설정하면 task 관리로 인하여 생기는 문제를 줄일 수 있으며, 효율적인 스케줄링을 가능하게 한다.

1. 서 론

임베디드 시스템이란 "다른 시스템의 일부로 내장된 마이크로프로세서 기반 디지털 시스템"을 의미한다. 주로 특정 기기에 내장된 컴퓨팅 시스템을 일컫는 말로, 임베디드 시스템은 범용 시스템과 달리 자신을 포함하고 있는 기기에 부과된 특정 목적의 컴퓨팅 작업만 수행한다.

이런 임베디드 시스템은 1950년대 통신 장비를 제어하기 위해여 컴퓨팅 시스템이 내장되면서 등장하였다. 이후 1990년대 초까지 군사용, 산업용 기기들을 제어하기 위한 목적으로 사용되었으며, 1990년대 중반부터 3C로 대변되는 기술들의 융합이 이루어지며 영역이 크게 확대되었다. 임베디드 시스템은 첨단 산업으로 재등장 하였으며, 우리의 일상에서 꼭 필요한 기기가 되었다.

임베디드 운영체제란 임베디드 시시템에서 사용되는 운영체제를 말한다. 아주 간단한 제어기능을 수행하기 위한 임베디드 시스템에는 운영체제를 사용하지 않는다. 하지만 임베디드 시스템의 성능이 커지면 기존에 간단한 펌웨어 형식으로 구현되었던 임베디드 프로그램이 점차 해야할 일들이 많아지고 복잡하게 되며, 순차적인 프로그램의 구현이 어려워진다. 이런 문제점을 해결하기 위해서 임베디드 시스템에 임베디드 운영체제를 적용해야 한다.

임베디드 운영체제를 사용하는 목적은 임베디드 시스템이 개인용 컴퓨터와는 다르기 때문에 프로세서의 성능이나 메모리 용량이 많아 시스템의 성능이 우수할수록 좋은 평가를 받는 것이 아니다. 단지 특정한 용도에 필요한 최소의 기능만 적절히 수행하면 된다. 하지만 인공위성이나 미사일 제어 등과 같이 오류에 견고한 시스템들은 신뢰성과 정확성을 요구하기 때문에 범용적인 운영체제로는 안정적이지 않다. 이런 경우에는 다양한 목적의 범용 운영체제보다는 RTOS와 같은 임베디드 운영체제를 사용해야 한다. [5]

Real-Time Kernel 기반의 MicroC/OS-II는 단순하면서 강력한 기능을 가지고 있기 때문에 현재 여러 분야에서 사용되고 있다. 하지만 MicroC/OS-II는 task 생성의 한계를 가지고 있으며, task 수가 늘어날수록 개발자에게는 task 관리의 어려움을 준다. 이 논문에서는 task 수가 확장이 되더라도 개발자가 task관리를 효율적으로 할 수 있게 한다.

2. 관련 연구

2.1 스케줄링

임베디드 운영체제는 일반적으로 task라 불리는 프로그램 수행 단위를 가진다. 임베디드 시스템은 여러 task가 동시에 수행되는 multi-tasking 환경을 가지며, 이때 운영체제 내부의 스케줄러에 의해서 다음번에 수행되어야 할 태스크를 결정한다.

Task의 도착 형태에 따라서 주기적, 비주기적 태스크로 분류될 수 있다. 주기적인 task는 일정시간마다 활성화되어 자기 자신의 종료시한 내에 작업을 수행하여야 한다. 그에 반해서 비주기적인 task는 외부로부터 발생되는 비동기적인 사건의 처리를 위해 사용된다.

Preemptive kernel은 어떤 태스크가 수행되고 있을 경우 커널이 강제로 그 task의 수행을 중지시키고 다른 task의 기능을 수행시킬 수 있으며 Non-Preemptive kernel에 비해 interrupt latency가 조금 긴 단점이 있다. 대부분의 임베디드 운영체제에서는 우선순위가 높은 task의 수행을 우선시하기 때문에 Preemptive kernel을 채택한다. 이 방법은 커널의 안정성을 높게 유지시킨다.

임베디드 운영체제에서는 우선순위 기반을 둔 스케줄링 알고리즘을 많이 사용하는데 FIFO, 라운드로빈, RM, EDF등을 사용한다. 이외 많은 알고리즘이 존재하지만 구현과 성능의 문제로 위 알고리즘을 가장 널리 사용하고 있다. [6]

2.2 MicroC/OS - II

MicroC/OS-II는 가장 기본적인 Kernel의 기능을 가지고 있는 운영체제라서 RTOS(Real Time Operating System)를 배우는 사람들에게는 교과서 같은 임베디드 운영체제이다. MicroC/OS-II는 다음과 같은 특징을 가진다.

* 공개된 오픈소스
* 8, 16, 32, 64 bit 시스템으로 높은 이식성
* 프로젝트에 따른 소스 코드의 절약
* 멀티태스킹
* Preemptive 방식
* 높은 신뢰성과 안정성
* 각종 커널 서비스

다양한 프로세서나 컨트롤러, DSP 등에 사용하기 위해서 범용적으로 설계되었으며, 소스가 모듈별로 나뉘어져 있어서 불필요한 기능일 경우 프로그램에 제외항으로 어플리케이션의 사이즈를 줄일 수 있다. [그림1]에서 보면 MicroC/OS-II는 크게 세 부분으로 나누어진다.

* 프로세서에 의존적인 부분
* 프로세서 독립적인 커널 파일
* 어플리케이션 부분

2.3 MicroC/OS-II 스케줄링

MicroC/OS-II는 우선순위 방식을 사용한다. 총 64개의 우선순위를 가지며 이중에 몇 개는 MicroC/OS-II에 의해 제한되어 있다. 우선순위 방식을 구현할 때 발생되는 문제 중의 하나는 만일 현재 프로세서를 점유하고 있는 task 보다 높은 우선순위를 가진 task가 프로세서를 요구한다면 프로세서를 점유 할 수 없다는 것이다. 높은 우선순위를 가진 task가 낮은 우선순위의 task의 작업을 강제로 중지시켜 자신의 프로세서를 점유하는 것이다.

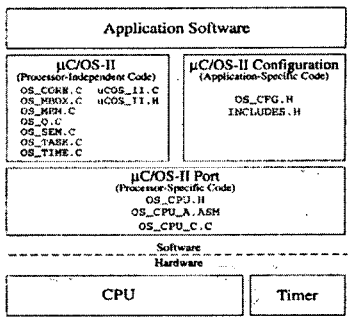

그림 1 MicroC/OS-II Architecture

3. 다단계 스케줄링 설계 및 구현

3.1 task 생성

각 태스크들은 커널의 스케줄링에 의하여 [그림2]와 같이 크게 WAIT, READY 그리고 RUNNING의 세 가지 상태를 가진다. [그림3]은 이 세 가지 상태를 확장한 그림이다. [그림2]의 상태를 스케줄링 관점에서 보게 되면 READY와 RUNNING 상태에 있는 task만이 scheduling의 대상이 되며, WAIT 상태에 있는 task가 scheduling 대상이 되려면 반드시 READY 상태로 와야 한다.

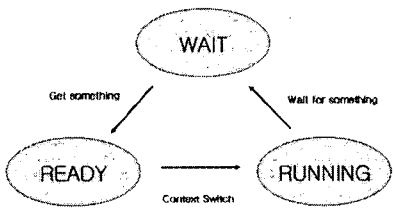

그림 2 Task의 상태

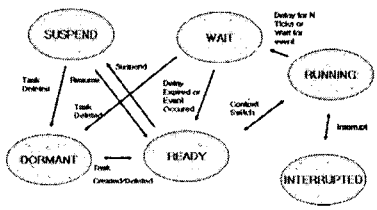

그림 3 확장된 task의 상태

위 그림과 같은 상태를 가진 task를 생성하기 위해서는 [그림4]의 함수를 사용한다.

이 논문에서는 task에 데드라인을 추가하기 때문에 생성 함수를 [그림5]와 같이 변경이 된다. 파라미터 'deadline'의 시간단위는 ms가 되며, 'level'은 task의 레벨을 가리킨다. 이 파라미터들은 task의 상태정보를 갖는 OS_TCB 구조체에 추가된다.

```
INT8U OSTaskCreate (void (*task)(void *pd), void *pdata,
                    OS_STX *ptos, INT8U prio
                    INT16U id, OS_STX *pbos,
                    INT32U stk_size, void *pext, INT16U opt )

task - Task로 만들기 위한 함수의 시작주소
pdata - Task에게 넘길 인자
ptos - Task가 사용할 스택의 시작주소
prio - Task의 우선순위
id - Task 구별 식별자
pbos - Task 스택의 마지막 번지
stk_size - Task의 스택 크기
pext - 사용자 정의 task control block
opt - task 생성 옵션을 지정
```

그림 4 task 생성 함수

기존에 'prio'는 task 식별자로도 사용했지만, 이 시스템에서 'prio'는 단순히 입력 우선순위만을 나타내고, 'id'가 task의 식별자로 사용하게 된다.

```
INT8U OSTaskCreate (void (*task)(void *pd), void *pdata,
                    OS_STX *ptos, INT8U prio
                    INT16U id, OS_STX *pbos,
                    INT32U stk_size, void *pext, INT16U opt
                    INT32U deadline, INT8U level)

deadline - task의 종료 시한
level - task의 레벨 설정
```

그림 5 새로운 task 생성 함수

3.2 priority 설정 부분

MicroC/OS-II는 task를 등록할 때 [그림6]과 같이 OSRdyGrp와 OSRdyTbl[]의 각 비트에 작업의 우선순위를 등록한다. 우선순위가 '11'인 task가 있으며, 이 task의 16진수는 '0B'이며, 이 값을 2진수로 변경하면 '00001011'이 된다. 최상위 2bit는 버리고, 중간의 3bit와 하위 3bit로 분리하면 '00-001-011'로 분리된다. 이 값의 상위 2bit는 버리고 중간의 3bit는 OSRdyGrp에 '0x2'로 변경되어 설정되며, 아래 3bit인 '011'은 '0x8'로 변경되어 OSRdyTbl[]에 설정된다.

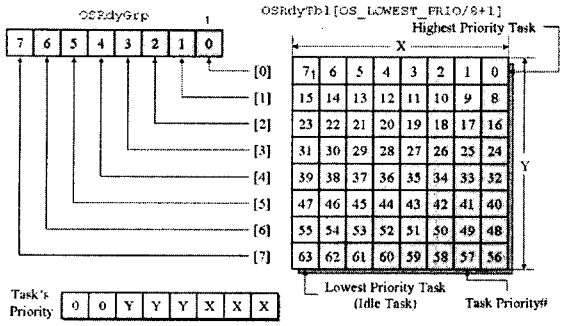

그림 6 MicroC/OS-II 준비 리스트

task가 많아지면 모든 task에 적절한 우선순위 설정하기가 어렵다. 개발자는 주요한 task들의 우선순위만 설정하는 부분(Level 1), 주요한 task는 아니지만 비슷한 우선순위들의 task들을 데드라인에 기반 하여 스케줄링 할 수 있는 부분 (Level 2), 우선순위와 상관없이 데드라인에 기반으로 실행하는 task 부분 (Level 3)으로 나눈다. 이때 task는 데드라인 안에서 꼭 수행이 완료되어야 하는 것은 아니다.

제안된 task 생성 함수는 'level' 파라미터 기반으로 'prio'와 'deadline'을 가지고 새로운 우선순위를 만든다. 우선순위를 결정시 [그림7]과 같이 3가지 레벨로 나눈다. 1레벨은 deadline에 상관없이 'prio'값만 가지고 수행되야 할 task이며, 2레벨은 'deadline' 값과 'prio' 값을 가지고 새로운 우선순위를 만들게 되며, 3레벨은 'prio'에 관계없이 'deadline' 값만 가지고 우선순위를 결정한다.

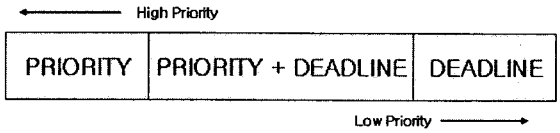

그림 7 레벨별 우선순위

각 레벨에 OSRdyGrp과 OSRdyTbl[]을 두어 스케줄링을 하면 task의 수는 3개로 증가하며, 레벨이 낮을수록 스케줄링이 우선시 되어 task들을 효율적으로 관리할 수 있게 된다. 개발자는 중요 task의 우선순위만을 신중히 설정하며, 나머지 task는 데드라인 값을 가지고 유연하게 스케줄링 가능하다. 이 시스템으로 개발자는 모든 task들을 편하게 생성하며 효율적인 관리를 한다.

4. 결론 및 향후 과제

MicroC/OS-II에서 task는 생성할 수 있는 제약이 있었으며, 만약 task의 수가 늘어나면 task들의 관리에 문제가 생긴다. 이 문제는 개발자를 힘들게 하며 효율적이지 못한 스케줄링의 요소가 된다. 이 논문에서 제안한 3레벨에 따른 스케줄링은 기존 시스템보다 3배 많은 task를 생성하며, 각 레벨에 따른 효율적인 스케줄링은 개발자에게 task 관리를 좀 더 손쉽게 해준다.

향후 과제로는 task의 데드라인과 우선순위를 조합하는 2레벨 단계에서 좀 더 효율적인 알고리즘을 제안하고, 3개의 OSRdyGrp와 OSRdyTbl[]을 단계별로 스케줄링 하는 방법의 연구가 필요하다.

참고문헌

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

[2] J.P. Lehoczky and S. R. Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems", In Proceedings of the 13th Real-Time Systems Symposium, pp.110-123, 1992.

[3] Jean J. Labrosse, "MicroC/OS-II The Real-Time Kernel", R&D Techincal books, 1998

[4] 김대홍, "uC/OS-II 뛰어넘기", Embedded World, 2003

[5] 홍성수 "임베디드 소프트웨어 기술동향", 한국정보산업연합회, 2003

[6] 김성관,하란,"실시간 스케줄링", 정보처리 제5권 제4호, pp.12~14,1998