제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

임베디드 운영체제 정의와 MicroC/OS-II에서의 확장된 스케줄링 구현

박현선*, 김인국**
단국대학교 전자계산학과
e-mail : (sunys, igkim)@cs.dankook.ac.kr

Definition of Embedded Operation System and an Implementation of Expended Scheduling for MicroC/OS-II

Hyun-Sun Park*, In-Kook Kim**
Dept. of Computer Science, Dankook University

요 약

90년대 중반 이후부터 컴퓨터 산업의 빠른 발전과 급격한 인터넷 발달로 컴퓨터 대중화가 빠르게 이루어져 왔다. 더불어, 근래에는 임베디드 시스템 분야가 빠르게 성장하고 있는데, 예전에는 단순한 작업만을 요구하던 임베디드 시스템이, 사용자들의 요구가 다양해지면서 운영체제의 필요성이 높아졌다.

이 논문에서는 임베디드 시스템과 임베디드 운영체제를 정의하고, 상용되고 있는 임베디드 운영체제에 대해 간단히 살펴보도록 하고, 그 중 MicroC/OS-II의 기존의 스케줄링 방식을 수정, 확장시켜보도록 하겠다.

1. 서론

90년대 중반 이후 급격하게 발전하게 된 컴퓨터 시장은, 90년대 후반, 2000년대 초반에 이르러서는 기능은 물론, 규모를 작게 하여 휴대하기 편리한 컴퓨터를 개발하기 시작하였다. 이에 노트북은 물론 많은 종류의 PDA들과 같은 임베디드 시스템 제품들이 출시되었고, 그 기능은 사용자들의 요구에 따라 빠르게 발전하고 있다.

기존의 임베디드 시스템은 단순한 공장화 작업과 같은 단순한 처리만을 하였기 때문에 특별히 운영체제를 필요로 하지 않았다. 그러나, 근래에는 사용자들의 편리성을 만족시키기 위해 많은 기능들을 필요로 하게 되었고, 그러면서 임베디드 시스템이 많은 기능을 제어하고 처리하기 위해 운영체제를 필요로 하게 되었다. 이에, 이 논문에서는 임베디드 시스템을 정의하고, 몇 가지 대표적인 임베디드 운영체제를 알아보도록 하며, μC/OS-II의 스케줄링을 RMS를 적용시켜 기존의 최우선순위 스케줄링과 비교해보고, 좀더 효과적인 스케줄링을 제안해보독 한다.

2. 임베디드 시스템과 임베디드 운영체제 개요

과거, 임베디드 시스템은 미리 정해진 특정한 작업만 하는 순차적 작업을 실행하는 시스템이었다. 그러나, 근래에는 너무나 많은 분야에서 과거와는 달리 복잡한 작업을 수행하는 수준에 이르고 있다. 다음 <표 1>은 임베디드 시스템의 적용 분야를 보이고 있다.

분야	사용 예
공장 자동화(FA)	제품 제조 공장의 생산라인 시스템 제어, 감시등을 모두 가능하게 함. 자동차 회사 등 공장의 「무인화(unmanned factory)」를 가능하게 함.
가정 자동화(HA)	컴퓨터 통신망을 이용한 생활정보, 문화정보, 홈뱅킹, 홈쇼핑, 학습정보, 진료를 비롯, 조명, 수도 가스, 난방, 가전제품까지 실내·외에서 제어가능
PDA	기본적인 수첩기능 외 필요 소프트웨어 사용 가능, 인터넷도 사용 가능하여 e-mail이나 주식 거래가 가능함
기타	세탁기, 냉장고, 전자레인지, 가스레인지, 전기 밥솥, 청소기, 에어컨 등 거의 모든 전자제품

* 학생 회원

** 정회원

<표 1> 임베디드 시스템 적용 분야

이렇게 다양한 분야에서 사용되는 임베디드 시스템은 수행하는 서비스들이 많아지고, 또 사용자가 요구하는 멀티미디어 서비스, 네트워크 서비스들을 수행하기 위해 운영체제를 필요로 하게 되었다. 그러나, 근래의 임베디드 시스템은 기본적으로 기기가 수행하는 서비스들이 많아지고, 또 사용자가 요구하는 멀티미디어 서비스, 네트워크 서비스를 수행하기 위해 운영체제를 필요로 하게 되었다.

임베디드 시스템은 작은 제한된 시스템이기 때문에, 자원에 대한 제약이 심하나, 안전성과 신뢰성을 만족해야하며, 이동성, 실시간성, 이식성 그리고 멀티미디어 서비스 제공이 필수적이 되고 있다. 그러면서 자연히 운영체제를 필요로하게 되었고, 이렇게, 제한된 임베디드 시스템에 사용되는 운영체제가 바로 임베디드 운영체제이다. 많은 자료들 중, 상당수에서 임베디드 운영체제와 실시간 운영체제(RTOS)를 같다고 표현하고 있는 것으로 보았는데, 이는 좀 문제가 있다고 하겠다. 임베디드 운영체제는 RTOS를 포괄하는 폭넓은 분야이기 때문이다. 임베디드 운영체제를 정의하자면, 제한된 환경의 임베디드 시스템에서 다양한 기능을 수행하기 위해 프로그램을 관리하는 운영체제라 할 수 있겠다.

근래에는 사용자의 요구에 따라 임베디드 운영체제가 데스크탑 운영체제의 모습을 갖춰가고 있는게 현실이다. 기존 RTOS 업계는 물론, 범용 운영체제 업계, 팜 OS 등이 가장 두드러지게 발전하고 있는데, 우리 나라에서도 임베디드 운영체제를 연구, 개발하는 업체가 100여개 가량 되고 있다.

2.1 임베디드 운영체제 소개

현재 꽤 많은 수의 임베디드 OS가 개발되고 있고, 또 대부분의 RTOS가 임베디드 시스템에서 사용가능하기 때문에, 이 논문에서는 몇 가지만 소개하도록 한다.

2.1.1 VxWorks

VxWorks는 Wind River System의 제품으로 화성착륙선 "패스파인더"에서 사용된 멀티 쓰레드형 운영체제로 세계 시장에서의 점유율이 가장 높다고 알려져 있는 실시간 운영체제인데, 마이크로 커널 기반의 클라이언트/서버 구조로, wind 마이크로 커널이라 부르며, 다음 <표 2>와 같은 기능을 지원한다.

Task 관리	멀티태스킹 지원
	선점형 스케줄링 라운드로빈 스케줄링
	Context Switching 지원
	256 Level 우선순위 지원
Task간 통신	Binary, Counting, Mutual Exclusion 세마포어 지원
	Message Queue 지원
	POSIX 파이프와 시그널 지원
	공유 Memory 지원
Interrupt 처리	
부동 소수점 지원	
Memory 관리	
System Clock 기능과 Timing 기능	

<표 2> Wind Kernel의 특징

이밖에도 VxWorks는 업계 표준의 망 기능을 통합한 실시간 운영체제로, TCP, UDP, IP, Socket, NFS, RPC, FTP, rlogin, telnet, bootp등을 지원하고 있다.

2.1.2 OS-9

Microware의 OS-9는 Multi Process형 임베디드 운영체제로, 1979년 모토로라가 개발한 8bit CPU6809용으로 개발된 것이 최초였다. 당시 8bit이면서도 실시간, 멀티 태스크, 멀티 유저를 지원하는 획기적인 OS로 주목받았었다. 이후, OS-9은 CPU가 6800계, 인텔 X86계로 진화되고, 현재는 PowerPC, SH, ARM, StrongARM 등의 RISC 프로세서로 그 수가 증가되었다. 그러면서 RTOS로서의 성능과 신뢰성, 유연한 구조가 지원되면서, 의학, 항공우주관련 전자공학, 로봇 공학, 통신, 산업용 프로세스 제어 등 다양한 분야에 걸쳐 전 세계적으로 사용되고 있다.

OS-9은 다음과 같은 기능을 제공한다.

- 컴퓨터와 사용자간의 인터페이스 제공
- 시스템의 I/O 동작 관리
- 프로그램 로딩(Loading)/실행 제공
- 디렉토리와 파일 시스템을 생성/관리
- 타임쉐어링(Timesharing)과 멀티 태스킹 관리
- 다양한 목적의 메모리 할당
- IPC(Interprocess Communication) 서비스 할당/관리

OS-9은 어떤 프로세스를 얼마나 실행해야 하는지를 실행될 프로세스들의 우선순위를 기반으로 그 순서를 결정하는데, 현재 Round Robin, Priority와 함께 Aging 스케줄링 방식을 사용하여, 우선순위가 낮은 프로세스들도 실행이 가능하도록 제공하고 있다.

이 외에도 pSOS나 MS사에서 개발한 Windows CE, RT-Linux, 그리고, 국내에서 개발된 e-Linux,

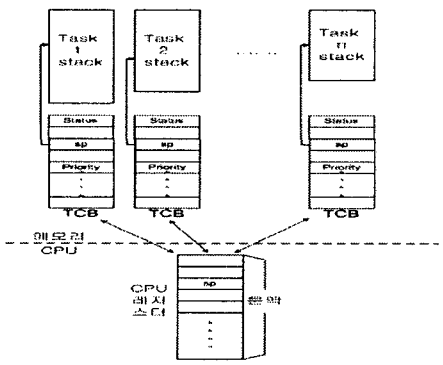
정보가전용으로 개발된 Qplus 등 많은 운영체제가 개발 중이다.

3. MicroC/OS-II에서 확장된 스케줄링 구현

이 논문은 MicroC/OS-II의 기존의 Priority - Scheduling, 즉, 단순 우선순위 기반 스케줄링으로는 여러개의 태스크를 효율적으로 관리하기 곤란한 부분이 있게 되는데 바로, 낮은 우선순위의 태스크들의 실행 보장을 받을 수 없게 되는 점이다. 이를 본 연구에서는 Rate-Monotonic 스케줄링으로 변경하여 좀더 나은 스케줄링 결과를 보이려 한다.

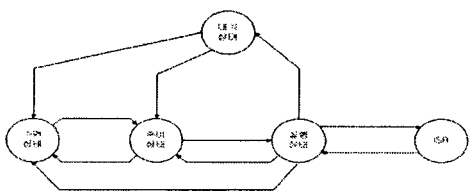
우선, MicroC/OS-II는 선점형 real-time 커널로, 멀티태스크를 지원, 최대 64개의 태스크를 지원한다.(시스템에서 사용중인 우선순위가 2개, 예약된 태스크가 6개 이므로, 사용자가 사용할 수 있는 우선순위는 56개 이다.) 또, 모든 태스크는 각각 중복되지 않은 고유 우선순위를 갖기 때문에, 선점형 우선순위 스케줄링만을 지원하고 있다. Context Switching, 세마포어, 이벤트 플래그, 메시지 메일박스, 메시지 큐 등도 지원하고 있다.

MicroC/OS-II는 멀티 쓰레드형 운영체제로서 여러개의 태스크가 존재하며, 각 태스크는 각각 TCB(Task Control Block)를 가지고 있으나, 실행되는 태스크는 1개이다.



<그림 2> MicroC/OS-II의 태스크

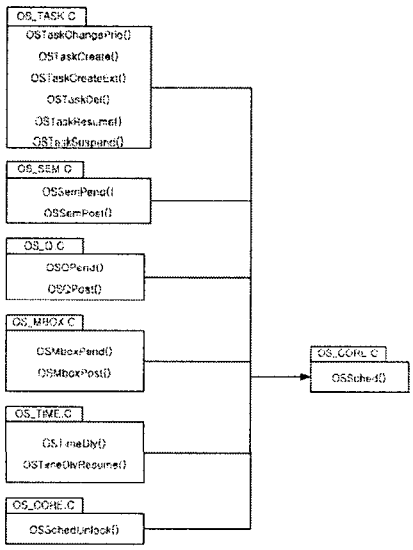
MicroC/OS-II에서 가능한 태스크들의 상태 다이어 그램이다.



<그림 3> MicroC/OS-II의 태스크 상태

이상에서 언급했듯이, MicroC/OS-II는 최우선순위 스케줄링 기법만을 사용하고 있다. 그러나, 이 스케줄링 방식은 최하위 우선순위를 가진 태스크의 경우 계속적인 대기상태에 있을 수 있다. 이러한 문제점의 해결 방법으로 Rate-Monotonic Scheduling(이하 RMS)을 이용하는 방법이다. 이미 RMS는 실제로 많은 논문에서 설명하고 증명되었기 때문에 이 논문에서는 생략한다.

MicroC/OS-II는 항상 실행 가능한 가장 높은 우선순위의 태스크를 실행하는데, 이 우선순위를 가지고 있는지를 결정하는 것이 바로 스케줄러가 담당을 하게 된다. 그 과정에서 여러 부분의 함수에서 호출하게 되는데, 다음 <그림 4>는 MicroC/OS-II에서 스케줄러 호출 구조이다.



<그림 4> MicroC/OS-II의 OSSched() 호출 과정

반면, RMS는 실행 주기가 가장 짧은 태스크에 가장 높은 우선순위를 주는 스케줄링이다. 원래 MicroC/OS-II는 우선순위가 고유했기 때문에 우선순위를 ID로도 사용했고, 태스크의 주기나 실행 시간을 필요로 하지 않았기 때문에, 구현을 위해 이들을 선언할 필요가 있었다. 다음은 MicroC/OS-II의 스케줄러 알고리즘이다.

```
void OSSched (void)
{
    크리티컬 섹션 시작;
    스케줄링이 Lock되었거나, ISR이 비어있을 경우{
      실행준비 상태인 최상위 우선순위 태스크 포인터 얻음
      현재 태스크가 최상위 우선순위인지 아닌지 판단
        최상위이면 문맥전환 없음
        최상위가 아니라면 문맥 전환 카운터 증가

        문맥 전환 수행
      }
    }
    크리티컬 섹션 종료;
}
```

<그림 5> MicroC/OS-II의 스케줄러

이상의 스케줄러를 RMS로 바꿔 구현하기 위해서 몇 가지를 수정해 줘야 했다. 기존의 스케줄러는 우선순위만을 필요로했기 때문에, 태스크가 생성될 때 실행 시간이나 주기가 필요 없었으나, RMS는 주기가 가장 짧은 태스크를 가장 높은 우선순위로 주기 때문에 주기와 실행시간을 받을 수 있도록 했다. 그리고, 가장 짧은 주기를 갖은 태스크를 선택하기 위해, 현재 시간을 기준으로 가장 남은 주기가 짧은 태스크를 찾을 수 있도록 했다. 다음 <그림 6>은 RMS 알고리즘을 적용한 새로운 스케줄러 이다.

```
OS_TCB* RMS_Find_Optimized_Task(int iCurTime)
{
  /* TCB 리스트에서 처음을 가리키는 포인터 */
  /* TCB를 가리키는 포인터에 남은 주기를 넣어주고, 만약
  남은 주기가 0보다 작을때 원래의 주기와 실행 시간 받도록
  하고, 포인터를 다음 리스트로 포인터를 옮겨는 방식으로,
  모든 태스크를 OSTCBList에 넣어주도록 한다. */

  /* 모든 태스크에 대하여 반영이 끝났다면 , 현재시간 기준
  으로 남은 기간에 비하여 남은 주기가 가장 짧은 것을 선택
  하도록 한다. */
  /* 주기가 가장 짧다는 것은 가장 할일이 많은 것, 즉 남은
  주기-남은 실행시간이 제일 작은 것을 선택하도록 했다.*/
}
```

<그림 6> RMS로 적합한 Task 찾는 알고리즘

다음 <그림 7>은 변형된 스케줄러 알고리즘이다.

```
void OSSched (void)
{
    크리티컬 섹션 시작;
    스케줄링이 Lock되었거나, ISR이 비어있을 경우{
     실행준비 상태인 최상위 우선순위 태스크를
      RMS_Find_Optimiz ed_Task()로 얻음
     리스트의 최상위 자리가 null이 아니라면, 0으로 하고 문
      맥전환시킴
      }
    }
    크리티컬 섹션 종료;
}
```

<그림 7> 변형된 스케줄러 알고리즘

4. 결과 및 결론

기존의 우선순위 기반 스케줄러를 사용했을 경우, 낮은 우선순위의 태스크에 대한 실행을 보장할 수 없었다. 그러나, RMS 알고리즘으로 수정된 알고리즘은 낮은 우선순위의 태스크라도 빈번히 실행되어야 하는 경우 그 실행을 보장받을 수 있다. 위 구현으로, 낮은 우선순위의 태스크에 대한 실행이 보장되는 스케줄링은 가능해 졌으나, 문제점은 남아있다.

RMS는 주기를 우선으로 하기 때문에 주기가 짧은, 즉 자주 실행되는 태스크가 가장 우선순위가 높게 된다. 그러나, 정작 중요한 태스크가 우선순위가 가장 높게 될 수 없을 수도 있는게 문제이다. 앞으로, RMS를 기반으로 스케줄링 되면서, 응용 프로그램의 내용에 따라 스케줄링 될 수 있도록 하는 스케줄러가 연구되어야 한다.

참고문헌

[1] 이정배, 이두원, "임베디드 시스템 연구 동향", 한국정보처리학회 학회지 제 9 권 제 1 호, pp 13-18, 2002년 1월.

[2] 김인국, "흐름 공정 모델의 효율적인 실시간 스케줄링", 박사학위 논문, 아주대학교, 1995

[3] 장현준, "RT-Linux를 위한 확장된 실시간 스케줄러 구현", 석사학위 논문, 단국대학교, 1998

[4] 차성덕, "RTLinux SMP를 위한 Mode Change 가능한 실시간 스케줄러의 구현", 석사학위 논문, 단국대학교, 2000

[5] Jean J. Labrosse, "MicroC/OS-II The Real-Time Kernel", R & D Technical Books, 1998

[6] William Stalling, "Operation System", Prentice-Hall, 1998

[7] Jane W. S. Liu, "Real-Time Systems", Prentice-Hall, 2000

[8] 이병관, "운영체제 일반", 도서출판 대림, 1998

[9] http://www.microware.co.kr

[10] http://www.windriver.com/

[11] http://www.microware.co.kr/

[12] http://www.dpc.or.kr/dbworld/document/