실시간 커널 μC/OS의 최대 허용 태스크 개수의 확장

도유환, 박명진, 오삼권
호서대학교 컴퓨터학부
avenue1@shinbiro.com, mjpark@osk.hoseo.ac.kr, ohsk@dogsuri.hoseo.ac.kr

Modifying The Real-Time Kernel μC/OS for Expanding the Maximum Allowed Number of Tasks

Yuhwan Do, Myungjin Park, Sam Kweon Oh
School of Computing, Hoseo University

요 약

본 논문에서 연구하는 μC/OS는 마이크로프로세서를 위한 선점형(Preemptive) 실시간(Real-Time) 멀티태스킹(Multitasking) 커널(kernel)로서, 연성실시간(SOFT Real-Time) 시스템 개발에 적합하다. μC/OS는 대부분의 소스코드가 C 언어로 작성되어 있으므로 실시간 커널의 연구에 용이하다.
본 논문에서는 μC/OS에서 처리할 수 있는 태스크(task)의 개수 확장을 위한 커널 모듈(mudule)의 설계 및 구현 방법을 기술한다. μC/OS는 최대 64개의 사용자 태스크를 관리할 수 있도록 구성되어 있다. 따라서 64개 이상의 태스크를 필요로 하는 시스템을 위한 커널로서는 적절하지 못함 문제점이 있다. 이 문제의 해결을 위해 본 논문은 태스크의 개수를 256개로 확장하고 이에 따른 커널 모듈 구조의 확장, 변환 방법을 설명한다.

1. 서 론

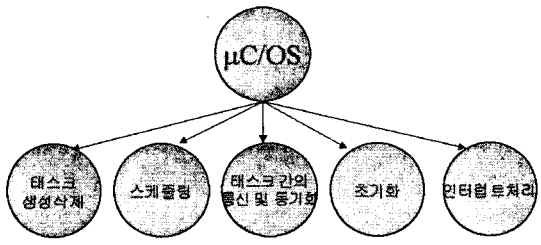
μC/OS, Nucleus RTX, Byte-BOS, VAX ELM, DEC ELX, RTUX, OSE, PDOS, VMEPROM, GX/Kernel, HP 1000 RTE-A, AMX, LynxOS, RTKernel, QNX, IXOS, SPOX, VENIX 등 많은 실시간 커널들이 현재 사용되고 있다. [1]

이중에서 μC/OS는 많은 다른 실시간 OS 처럼 대부분이 C 언어로 작성되어 있고 어셈블리 언어(Assembly Language)로 작성된 부분은 최소화되어 있으므로 실시간 커널의 포팅에 용이하다.

μC/OS는 마이크로 프로세서를 위한 선점형 실시간 멀티태스킹 커널이며, ROM에 이식이 가능하다.[1]

μC/OS는 연성 실시간 시스템이며 최대 64개의 태스크를 관리할 수 있도록 구성되어 있다. 따라서 64개 이상의 태스크를 필요로 하는 시스템을 위한 커널로서는 적절하지 못한 문제점이 있다. 이 문제의 해결을 위해 본 논문은 태스크의 개수를 256개로 확장한다. 여기서 태스크 스케줄링 알고리즘은 최대 허용 태스크의 개수가 64개일 때와 동일한 알고리즘을 사용한다. 이에 따라 가장 크게 그 구조가 확장, 변환되는 커널 모듈은 태스크 스케줄링 모듈이다. 이 모듈들이 확장 되야 하는 이유와 자세한 확장, 변환 방법은 본문에서 설명한다.

본 논문의 구성은 다음과 같다. 2장에서는 μC/OS의 전체 구조를 살펴보고 3장에서는 스케줄링 모듈을 설명한다. 4장에서는 3장에서 설명한 스케줄링 모듈의 확장 전과 확장 후의 상태를 비교 분석하고 5장에서 결론을 맺는다.



[그림 1] μC/OS 모듈의 기능에 따른 분류

2. μC/OS의 구조

μC/OS는 [그림 1]과 같이 각 모듈을 그 기능에 따라 5개의 범주로 나눌 수 있다. 각 범주 별 모듈의 기능은 다음과 같다.

- 태스크 생성, 삭제
  - 태스크 생성 모듈
  - 태스크 삭제 모듈
- 스케줄링
  - 태스크 스케줄링 모듈
  - 태스크 스케줄링을 일시 정시시키는 모듈
  - 태스크 스케줄링을 제기시키는 모듈
- 태스크간의 통신 및 동기화
  - 메일박스(Mailbox), 큐(Queue)를 이용하여 태스크간의 통신을 담당하는 모듈
  - 세마포(Semaphore)를 이용하여 태스크간의 동기화를 담당하는 모듈
- 초기화
  - μC/OS의 헤더 파일에 정의되어 있는 변수들과 자료구조들을 멀티태스킹이 시작하기 전에 초기화하는 모듈
- 인터럽트 처리
  - 인터럽트를 처리하는 모듈

3. 스케줄링 모듈

최대 허용 태스크의 개수를 확장한 결과 그 구조가 가장 크게 확장, 변환되는 것은 태스크 스케줄링 모듈이다. 스케줄링 모듈에서는 최상위 우선순위 태스크를 선택하기 위해 [그림 2]와 [그림 3]에 있는 준비리스트(Ready List)와 맵핑테이블(Mapping Table)을 사용한다.

다음은 준비 리스트와 맵핑테이블의 설명이다.

(1) 준비 리스트(Ready List)

- OSRdyTbl[]
  - 태스크의 우선순위 번호를 갖고 있는 테이블
  - 테이블의 전체 비트 수가 커널에서 허용 가능한 최대 태스크의 수를 나타낸다.
  - 따라서 태스크의 개수가 증가하면 가장 먼저 확장되어야한다.
- OSRdyGrp
  - OSRdyTbl[]의 태스크 우선순위가 그룹화 되어있다.
  - 각각의 비트는 태스크 그룹의 어떤 태스크라도 실행 준비 상태라면 1로 세트된다.

(2) 맵핑 테이블(Mapping Table)

- OSMapTbl[]
  - 0부터 7까지 8개의 값을 이진 값으로 비트 마스크(bit mask) 하기 위한 테이블
- OSUnMapTbl[]
  - 최상위 우선순위 태스크의 OSRdyTbl[]에서의 위치를 찾는데 사용되는 배열
  - OSRdyTbl[] 하나의 변수가 나타낼 수 있는 10진수 숫자만큼의 상수가 필요하다.
    예를 들면 8비트의 OSRdyTbl[]을 사용한다면 2^8(256)개의 상수를 갖는 OSUnMapTbl[]이 필요하다.

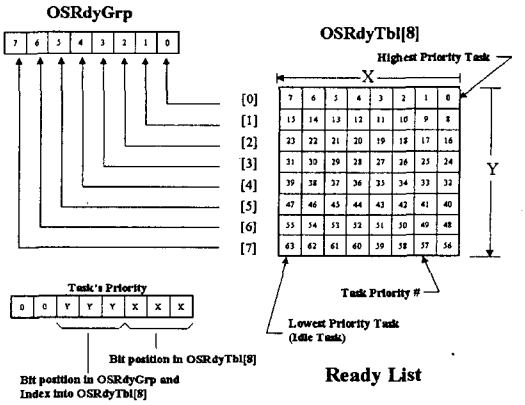
태스크 스케줄링 모듈에서 최상위 우선순위 태스크를 선택하는 과정은 다음과 같다.

```
y = OSUnMapTbl[OSRdyGrp];          --- ①
x = OSUnMapTbl[OSRdyTbl[y]];       --- ②
p = (y<<3) + x;                    --- ③
```

①은 [그림 2]의 OSRdyGrp의 값을 OSUnMapTbl[]에 적용하여 최상위 우선순위 태스크가 OSRdyTbl[]의 몇 번째 변수에 있는지를 알아낸다.

②은 ①에서 얻은 값을 이용하여 최상위 우선순위 태스크가 있는 OSRdyTbl[]의 값을 구한다. 다시 이 값을 OSUnMapTbl[]에 적용하여 OSRdyTbl[]에서 최상위 우선순위 태스크의 위치를 알아낸다.

③은 ①에서 구한 값에 8을 곱하고 여기에 ②에서 구한 값을 더하여 최상위 우선순위 태스크의 우선순위 번호를 구한다.



[그림 2] 64개의 태스크를 수용 가능한 준비리스트

```
UBYTE const OSMapTbl[]   =
{0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

UBYTE const OSUnMapTbl[] = {
0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0 };
```

[그림 3] 태스크 개수 확장 이전의 OSMapTbl[]와 OSUnMapTbl[]

4. Task 개수 확장을 위한 스케줄링 모듈의 설계 및 구현

4.1 Task 개수 확장 이전의 스케줄링 모듈의 구조

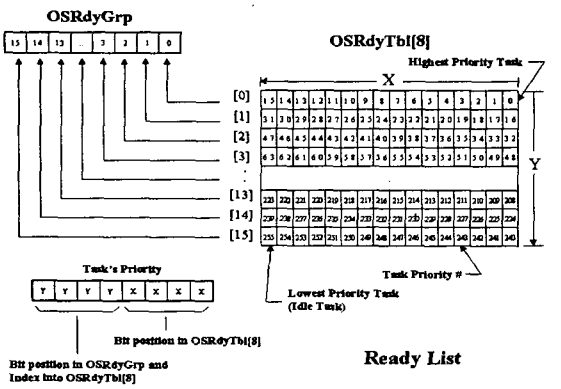
[그림 2]은 64개의 태스크를 수용할 수 있는 준비리스트의 구성을 나타낸다. OSRdyTbl[8]은 8비트의 변수 8개로 구성되어 있으므로 테이블 전체의 비트 수는 64개가 되고, 각 비트는 태스크의 우선순위 번호를 나타낸다.

[그림 3]은 태스크의 개수가 64개일 경우에 필요한 OSMapTbl[]과 OSUnMapTbl[]을 나타낸다. OSMapTbl[] 에는 0부터 7까지 8개의 값을 이진 값으로 비트 마스킹(Masking)할 때 필요한 8개의 16진수 값이 필요하다. OSUnMapTbl[]에는 2^8에 해당하는 256개의 상수가 필요하다.

4.2 Task 개수 확장 이후의 스케줄링 모듈의 구조

[그림 4]는 256개의 태스크를 수용할 수 있도록 확장한 후의 준비리스트를 나타낸다. OSRdyTbl[16]은 16비트의 변수 16개로 구성되어 있으므로 테이블 전체의 비트 수는 256개가 되고, 각 비트는 태스크의 우선순위 번호를 나타낸다.

[그림 5]은 태스크의 개수를 256개로 확장했을 때 필요한 OSMapTbl[]과 OSUnMapTbl[]2, OSUnMapTbl() 함수이다. 3장에서 사용한 스케줄링 알고리즘을 그대로 적용한다면, OSMapTbl[]에는 0부터 15까지 16개의 값을 이진 값으로 비트 마스킹할 때 필요한 16개의 16진수 값이 필요하다. OSUnMapTbl[]에는 2^{16}에 해당하는 65536개의 상수가 필요하다. 이 때 발생하는 65536개의 상수는 너무 많은 소스 코드의 양을 필요로 한다는 문제점이 있다. 이를 해결하기 위해서 본 논문은 OSUnMapTbl[]의 기능을 대신하면서 소스 코드를 간단하게 줄일 수 있는 OSUnMapTbl() 함수를 구현하였다. OSUnMapTbl() 함수는 OSUnMapTbl[]에 나열되는 상수들의 나열 규칙을 이용한 것이다. 이 함수는 256개 이상의 태스크를 사용가능 하도록 준비리스트와 맵핑테이블을 확장하여도, 스케줄링 알고리즘의 변화 없이 함수를 사용할 수 있도록 설계하였다. OSUnMapTbl()은 소스코드의 양을 줄일 수 있는 장점을 얻을 수 있는 반면, 태스크의 스케줄링 시에 OSUnMapTbl[]을 사용하는 것보다 처리 속도가 증가하는 단점이 있다.



[그림 4] 256개의 태스크를 수용 가능한 준비리스트

```
UBYTE const OSMapTbl[] =
{0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020,
0x0040, 0x0080, 0x0100, 0x0200, 0x0400, 0x0800,
0x1000, 0x2000, 0x4000, 0x8000};

UBYTE const OSUnMapTbl2[] =
{0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0};

UBYTE const OSUnMapTbl(UWORD p)
{
        UBYTE i;
        UBYTE j = 1;
        UWORD pRemain,pMok;
        pRemain = p%16;
        pMok = p/16;
        if (pRemain != 0)
          return (OSUnMapTbl2[pRemain]);
        else {
                for ( i=4; i<16; i++) {
                if (((pMok/j)%2) == 1) {
                        break;}
                j=j*2;
                }
        }
        return(i);
}
```

[그림 5] 태스크 개수 확장 후의 OSMapTbl[]과 OSUnMapTbl[]2, OSUnMapTbl()

5. 결론

본 논문에서는 64개로 한정되어 있는 μC/OS의 최대 허용 태스크의 개수를 256개로 확장하였다. 그리고 이에 따른 커널 모듈 구조의 확장, 변화를 설계 및 구현하였다. 특히 태스크의 개수가 256개로 증가함에 따라 OSUnMapTbl[] 상수들을 엄청나게 증가하는 문제점을 해결하기 위해 간단한 OSUnMapTbl() 함수를 구현하였다. 하지만 함수형태로 구현한 OSUnMapTbl()은 OSUnMapTbl[]에 비해 처리 속도가 증가하는 단점이 있다.

참고문헌

[1] Jean J.Labrosse, "μC/OS The Real-Time Kernel", R&D Publications, 1992

[2] C.M.Krishna, Kang G.Shin, "Real-Time Systems", The McGraw-Hill Companies, Inc. 1997,

[3] Jean J.Labrosse, "A Portable Real-Time Kernel in C", Embedded Systems Programming, May 1992

[4] Jean J.Labrosse, "Implementing a Real-Time Kernel", Embedded Systems Programming, June 1992

[5] Sang H.Son, "Advances in Real-Time Systems", Prentice Hall, 1995