실시간 운영체제를 위한 태스크 스케줄링의 설계 및 구현

박윤미[0], 김용희, 성영락[†], 이철훈
충남대학교 컴퓨터공학과, [†] 국민대학교 전자정보통신공학부
{ympark[0], yonghee, chlee}@ce.cnu.ac.kr, [†] yeong@mail.kookmin.ac.kr

Design and Implementation Task Scheduling for Real-Time Operating Systems

Yun-Mi Park[0], Yong-Hee Kim, Yeong Rak Seong[†], and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
[†] School of Electrical Engineering, Kookmin Univ

요 약

최근의 임베디드 시스템 분야에서 실시간 운영체제를 탑재하여 개발 된 제품들이 점차 늘어 증가하는 추세이다. 실시간 운영체제는 다른 범용 운영체제와는 달리 시간 결정성을 보장하는 운영체제이다. 그리고 실시간 운영체제를 포함하는 임베디드 시스템은 특정 목적을 위해 간결하게 만들어지기 때문에 한정된 자원을 효율적으로 사용해야 한다. 본 논문에서는 실시간 운영체제 iRTOS™에서 태스크의 우선순위를 64 에서 256 으로 확장하면서 발생하는 메모리의 낭비를 줄이는 스케줄링 방법을 설계하고 구현한 내용을 설명한다.

1. 서론

최근의 임베디드 시스템 분야에서 실시간 운영체제를 탑재하여 개발 된 제품들이 점차 늘어 나고 있는 추세이고, 네트워크나 멀티미디어 장비와 같이 시스템에서 처리해 주어야 하는 일의 양이 점점 많아지는 상황에서 실시간 운영체제 탑재는 너무나 당연한 것으로 인식 되고 있다. 실시간 운영체제는 다른 범용 운영체제인 Unix™, Linux™, Windows™등과 같이 멀티태스킹(Multitasking) 및 ITC(InterTask Communication), 인터럽트 서비스 루틴(Interrupt Service Routine), 메모리 할당(Storage Allocation) 등의 기능에 실시간 적인 요소가 추가된 운영체제이다. 여기서 실시간의 의미는 임의의 정보가 시스템에 입력 됐을 때, 주어진 시간 안에 작업이 완료되어 결과가 주어져야 하는 것을 의미한다. 즉, 일반 목적의 시스템에 탑재되는 운영체제의 경우에는 하드웨어 자원(메모리, I/O 디바이스, 하드디스크 등)을 얼마나 효율적으로 사용하고, 사용자들에게 얼마나 공평하게 이 자원을 분배할 것인지에 초점을 맞추고 있는 반면, 실시간 운영체제는 정해진 시간 제약을 해결하는데 초점을 맞추고 있다. 그리고 스케줄러는 범용 운영체제와 실시간 운영체제를 구분하는 주요 요소이다. 대부분의 상용 실시간 운영체제는 우선순위(Priority) 기반의 선점형(Preemptive) 스케줄링을 수행한다. 스케줄러는 수행 준비된 태스크들 중에서 가장 높은 우선순위의 태스크를 찾기 위해 사상(mapping) 테이블을 사용하여 정해진 몇 번의 연산으로 가장 높은 우선순위를 찾아낼 수 있다. 사상 방법은 가장 높은 우선순위를 찾는 데는 매우 효율적이지만, 우선순위의 수가 증가 할 수록 사상 테이블의 크기가 커지면서 메모리 용량을 많이 차지하게 된다. 임베디드 시스템은 특정 목적을 위해 간결하게 만들어지기 때문에 메모리 용량이 한계가 있다. 따라서 본 논문에서는 실시간 운영체제 iRTOS™을 연구대상으로 하여 태스크의 우선순위를 64 개에서 256 개로 확장함으로써 발생하는 메모리 사용의 증가를 줄일 수 있는 태스크 스케줄링 방법을 설계하고 구현한 내용을 설명하고 있다.

본 논문은 2 장에서 관련 연구를, 3 장에서 스케줄링에 대한 설계 및 구현 내용을, 4 장에서는 테스트 환경 및 결과를 기술하고, 마지막 5 장에서는 결론 및 향후 연구과제에 대해서 기술한다.

2. 관련 연구

2.1 실시간 운영체제

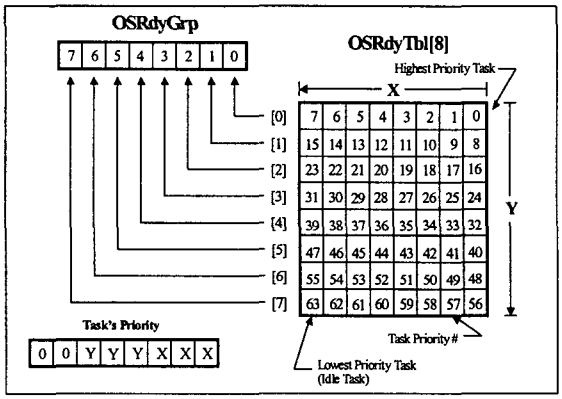
실시간 운영체제는 주어진 작업을 정해진 시간 안에 수행하여 결과를 제공하는 운영체제이다. 실시간 시스템은 크게 경성 실시간 시스템(Hard Real Time System)과 연성 실시간 시스템(Soft Real Time System)의 두 가지로 나뉜다. 전자의 경우는 정해진 시간 내에 작업의 결과가 절대적으로 출력되어야 하는 시스템이다. 간단한 예를 들면 전투기의 비행 제어 시스템, 핵발전소의 제어 시스템, 인공위성의 제어 시스템 등 작업의 결과가 정해진 시간 내에 나오지 않게 되면 치명적인 오류가 발생하게 되는 경우에 적용된다. 반면에 후자의 경우는 정해진 시간 내에 작업의 결과가 출력되지 않더라도 경성 실시간 시스템처럼 치명적인 결과는 나오지 않는 그런 시스템일 경우이다. 즉 연성 실시간 시스템은 정해진 범위를 넘는 시간 지연이 발생하더라도 시스템에 큰 영향을 미치지 않기 때문에 오류라고 판단하기 보다는 단지 처리 지연이 발생하는 경우라고 간주될 수 있다. 이러한 실시간 운영체제에서

* 본 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임

가장 중요한 부분은 스케줄러로서 READY 상태에 있는 여러 개의 태스크 중에서 CPU 를 선점하고 실행하는 태스크를 결정하는 기능을 한다. 일반적으로 가장 높은 우선순위의 태스크를 수행시키는 방법을 사용한다. 이때, 가장 높은 우선순위의 태스크를 찾기 위해 사상(mapping) 테이블을 사용하게 되는데 그에 대한 스케줄링 방법은 아래에서 설명하도록 하겠다.

2.1 태스크 스케줄링

[그림 1]은 64 개의 우선순위를 가지는 태스크들의 스케줄링을 위한 커널의 자료구조(READY list)를 나타낸 것으로, 태스크들은 0 에서 63 까지의 우선순위 중 하나의 우선순위를 가지게 된다. READY 상태의 각 태스크들은 OSRdyGrp, OSRdyTbl[8] 두 변수로 구성된 ready 리스트에 놓여지게 된다. MK_RdyTbl[8]의 64 비트 중에서 1 로 set 된 것은 해당 위치의 태스크가 ready 되었다는 것을 의미하며, MKRdyGrp 의 8 비트 중에서 1 로 set 된 것은 해당 위치의 그룹 중에 한 개 이상의 태스크가 ready 되었다는 것을 의미한다.



[그림 1] 64 개 우선순위의 READY list

[그림 2]는 MK_MapTbl 은 0 에서 7 까지 해당 되는 bit mask 에 대하여 인덱스를 표시할 때 사용하는 것이고, MK_UnMapTbl[]은 우선순위 분석 테이블로써 가장 높은 우선순위를 갖는 태스크의 위치를 찾는데 사용된다.

```
/* Mapping Table to Map Bit Position to Bit Mask */
UCHAR  const  OSMapTbl[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80};

/* Priority Resolution Table */
UCHAR 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 2] map and unmap 테이블 리스트

태스크가 ready 리스트로 삽입될 때는 아래와 같은 코드를 수행하게 된다.

```
OSRdyGrp          |= OSMapTbl[p >> 3];
OSRdyTbl[p >> 3]  |= OSMapTbl[p & 0x07];
```

태스크가 ready 리스트에서 삭제될 때는 아래와 같은 코드를 수행하게 된다.

```
if ((OSRdyTbl[p>>3] &= ~OSMapTbl[p & 0x07])
    == 0)
  OSRdyGrp  &= ~OSMapTbl[p >> 3];
```

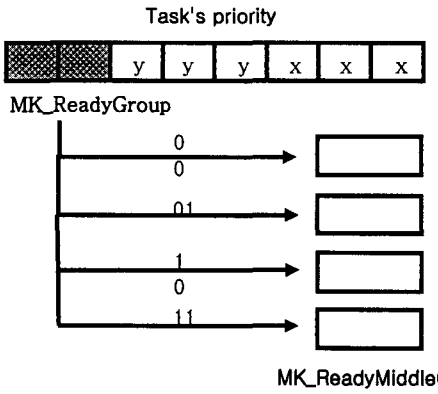
스케줄시, 최상위 우선순위 태스크를 선택할 때 아래와 같은 코드를 수행하게 된다.

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

3. 스케줄링에 대한 설계 및 구현 내용

64 개의 우선순위를 갖는 태스크의 스케줄링에서 메모리 사상 방법 은 전역변수 MK_MapTbl[], MK_UnMapTbl[]을 사용함으로써 메모리의 낭비가 있지만, 가장 높은 우선순위를 찾는 데는 매우 효율적이다. 스케줄링 정책에서 사상 테이블 UNMapTable[]과 MapTable[]을 사용하게 되는데 우선순위의 수에 따라 큰 차이를 보이는 테이블은 UNMapTable[]이다. 우선순위를 256 개로 확장하면 결국 사상 테이블의 크기가 커지면서 메모리 용량을 많이 차지하게 된다. 그래서 256 개의 우선순위를 제공하고 있는 iRTOS™ 운영체제에서는 64 개의 스케줄링 기법을 변형하여 메모리 낭비를 줄일 수 있도록 설계 하였다.

256 개의 우선순위를 가지는 태스크들의 스케줄링을 위한 커널의 자료구조에는 MK_ReadyGroup, MK_ReadyMiddleGroup[4], MK_ReadyTable[4][8] 세 변수에 의해 구성이 된다.



[그림 3] 256개 우선순위의 Task' s priority

우선순위의 최상위 비트(6~7 bit)는 [그림 3]에서 보는 바와 같이 새롭게 추가된 MK_ReadyMiddleGroup 을 가리키게 되고, MK_ReadyMiddleGroup 에서는 READY list 를 가리키게 되는데, 이는 64 개 우선순위를 기반으로 한 스케줄링 정책과 동일하다. MK_ReadyMiddleGroup 의 구성으로 [그림 2]의 UNMapTable[]을 사용하게 됨으로써 메모리 사용을 현저히 줄일 수가 있다.

새로운 우선순위의 태스크가 될 경우에는 아래와 같은 코드를 수행하게 된다.

```
MK_ReadyGroup |=
MK_MapTable[GroupValue>>6];
  MK_ReadyMiddleGroup[GroupValue>>6] |=
    MK_MapTable[MiddleGroupValue>>3];
  MK_ReadyTable[GroupValue>>6][MiddleGrou
    pValue>>3] |= MK_MapTable[TableValue];
```

[그림 4] 새로 추가된 우선순위 반영

태스크가 ready 리스트에서 삭제될 때는 아래와 같은 코드를 수행하게 된다.

```
  MK_ReadyTable[GroupValue][MiddleGroupVal
    ue] &= ~MK_MapTable[TableValue];
  if(MK_ReadyTable[GroupValue][MiddleGroupV
    alue] == 0)
  {
    MK_ReadyMiddleGroup[GroupValue] &=
      ~MK_MapTable[MiddleGroupValue];
    if(MK_ReadyMiddleGroup[GroupValue] == 0)
      MK_ReadyGroup &=
~MK_MapTable[GroupValue];
  }
```

[그림 5] 삭제된 우선순위 반영

스케줄시, 최상위 우선순위 태스크를 선택할 때 아래와 같은 코드를 수행하게 된다.

```
  GroupValue =
MK_UnMapTable[MK_ReadyGroup];
  Temp = MK_ReadyMiddleGroup[GroupValue];
  MiddleGroupValue = MK_UnMapTable[Temp];
  Temp =
    MK_ReadyTable[GroupValue][MiddleGroupVal
    ue];
  TableValue = MK_UnMapTable[Temp];
  High = (GroupValue<<6) |
    (MiddleGroupValue<<3) |  ableValue;
```

[그림 6] 가장 높은 우선순위 구하기

4. 테스트 환경 및 결과

본 논문에서 기술하고 있는 실시간 운영체제는 iRTOS™ 이고, 컴파일러는 ARM SDT 2.51 을 사용하며, ARM 920T 를 기반으로 한 삼성 S3C2400™ 32-bit RISC MicroProcessor 에서 테스트 하였다.

iRTOS™ 는 우선순위 기반의 선점형 실시간 운영체제이고, 256 개 우선순위를 제공해주고 있다. 스케줄링 정책에서는 사상테이블 MK_MapTbl[], MK_UnMapTbl[]을 사용하게 되는데, 이는 우선순위의 수에 따라 현저히 값이 증가하게 된다. 그래서 본 논문에서는 MK_ReadyMiddleGroup 의 구성으로 64 개 우선순위를 기반으로 한 사상테이블을 사용함으로써 메모리의 사용을 줄일 수 가 있었다.

5. 결론 및 향후 연구 과제

특정 기능을 수행하는 임베디드 시스템에서는 적은 용량의 메모리를 가지고 있기 때문에 메모리 공간을 효율적으로 이용하는 것이 중요하다. 본 논문에서는 256 개의 우선순위를 제공하는 운영체제에서 보다 효율적으로 메모리 공간을 이용할 수 있는 스케줄링 정책을 설계해 보았다.

iRTOS™ 스케줄링 정책에서 메모리 사용을 줄이면서 [그림 4], [그림 5], [그림 6]에서 보는 것처럼 수행되는 연산의 크기가 큰 것을 알 수 있다. 향후 연구과제로는 iRTOS™의 스케줄링 정책에서 보다 적은 연산을 수행 할 수 있도록 보완해야 할 것이다.

6. 참고 문헌

[1] Jean J. Labrosse, *uC/OS: The Real-Time Kernel*, R&D Publications, 1995.

[2] Embedded System, RTOS, http://www.inestech.com

[3] Accelerated Technology, *Nucleus PLUS Internals Manual*, 1996.

[4] IEEE Std 1003.1b, *Portable Operating System*, 1993.

[5] 박희상, 정명조, 조희남, 이철훈, " *Design of Open Architecture Real-Time OS Kernel*" , 한국정보과학회, Vol. 29, No. 2(I), pp.418-420, 2002.10

[6] David Stepner, Nagarajan Rajan, David Hui, " *Embedded Application Design Using a Real-Time OS*" , Design Automation Conference, 36th, pp.151-156, 1999