다차원 기법을 이용한 실시간 태스크 스케줄링 알고리즘*

Real-time Task Scheduling Algorithm using Multi-dimensional Methodology

조문행, 이철훈
충남대학교 컴퓨터공학과

Cho Moon-Haeng, Lee Cheol-Hoon
Chungnam National Univ., Computer Engineering, System Software Laboratory

요약

오늘날의 핸드폰, PMP, 지능형 가정기기, 자동차 엔진 컨트롤 시스템과 같은 내장형 시스템은 인간의 삶과 일, 놀이 등 살아가는 환경에 대한 전환을 가져오고 있다. 이에 따라, 내장형 시스템 응용들의 복잡성이 증가하고 있으며, 그런 내장형 컴퓨팅 플랫폼에는 시간결정성을 갖는 실시간 운영체제를 사용해야 한다. 이런 실시간 운영체제들은 예측 가능한 서비스들 뿐만 아니라 작은 실행 이미지 크기를 가져야만 하고, 커널 서비스들은 각 서비스 수행에 얼마만큼의 시간이 소요되는지를 명세하여 시간결정적 이어야만 한다. 이런 정보를 토대로, 응용 개발자들은 각 태스크가 마감시간을 준수하도록 하는 실시간 응용 소프트웨어를 설계할 수 있다. 본 논문에서는 2r 레벨의 우선순위를 갖는 태스크들 중 최상위 우선순위를 결정하는데 있어 고정 상수시간을 보장하는 다차원 기법의 실시간 태스크 스케줄링 알고리즘을 기술한다.

Abstract

In recent years, embedded systems such as cellular phones, Portable Multimedia Player, intelligent appliance, automobile engine control are reshaping the way people live, work, and play. Thereby, applications for embedded systems become increasingly sophisticated and complicated, such embedded computing platforms must use real-time operating systems (RTOSs) with time determinism. These RTOSs must not only provide predictable services but must also be efficient and small in size and it's kernel services should also be deterministic by specifying how long each service call will take to execute. Having this information allows the application developers to better plan their real-time application software so as not to miss the deadline of each task. In this paper, we present the complete generalized algorithm using multi-dimensional methodology to determine the highest priority in the ready list with 2r levels of priorities for an arbitrary integer number of r.

Ⅰ. 서론

실시간 시스템은 논리적 정확성뿐만 아니라 시간적 정확성을 보장해야 하는 시스템으로 정의한다. 즉, 시스템의 수행 결과가 기능적으로 정확해야 할 뿐만 아니라, 결과가 도출되는 시간 역시 주어진 제약 조건을 만족해야 한다. 실시간 시스템은 주어진 시간의 엄격성에 따라 크게 3가지로 분류할 수 있다[1][2].

- 경성(hard) 실시간 시스템 : 시스템이 주어진 종료시한을 만족시키지 못한 경우에 막대한 재산적 손실이나 인명의 피해를 주는 시스템.
- 연성(soft) 실시간 시스템 : 온라인 시스템과 같이 시

* 본 연구는 지식경제부의 IT R&D 지원으로 수행되었습니다.[2008-F-048, 웨어러블 컴퓨너언 개발 사업]

간제약 조건을 만족시키지 못하더라도 경성의 경우처럼 치명적이지 않고 종료시한을 넘겨 수행을 마쳐도 계산의 결과가 의미가 있는 시스템.

•준경성(firm) 실시간 시스템 : 경성과 연성의 중간 형태로 종료시한을 넘겨 수행을 마치는 것은 무의미한 경우이지만 시간초과에 대한 손실이 치명적이지 않은 시스템.

위와 같은 실시간 시스템의 특성은 실시간 운영체제가 제공하는 시스템 수준에서의 커널 서비스를 통해 보장할 수 있다. 또한 실시간 운영체제는 모든 실시간 태스크에 대해 그 종료 시한을 만족할 수 있도록 하며, 실행 준비 상태의 높은 우선순위 태스크가 존재하는 데도 낮은 우선순위의 태스크가 CPU를 점유하는 우선순위 역전현상이 발생하지 않도록 보장한다. 또한, 주어진 종료시한을 만족하기 위해 스케줄러는 시간 결정성을 보장해야 한다. 이런 시간 결정성을 보장하기 위해서는 시간 결정성을 갖는 스케줄링 알고리즘이 필요하며, 그에 따른 메모리 오버헤드가 존재한다.

본 논문에서는 시간 결정성을 보장하면서도 메모리 오버헤드가 없는 다차원 실시간 태스크 스케줄링 기법에 대해 설계 및 구현한 내용을 소개한다.

본 논문의 구성은, 관련연구에서 실시간 태스크 스케줄링 알고리즘 오버헤드와 μ C/OS의 실시간 태스크 스케줄링 알고리즘, 2차원 기법을 활용한 실시간 태스크 스케줄링 알고리즘에 대해 기술하고, 3장에서 시간 결정성을 보장하면서 메모리 오버헤드를 제거한 다차원 기법의 실시간 태스크 스케줄링 알고리즘에 대해 제안한다. 4장에서는 제안한 알고리즘과 기존 알고리즘을 비교 분석한 결과를 제시하고 마지막 장에서 결론을 맺는다.

Ⅱ. 관련연구

1. 실시간 태스크 스케줄링 알고리즘 오버헤드

태스크 스케줄링 오버헤드($\triangle t$)는 스케줄러 코드에 의해 실행되는 시간이며, 스케줄러 코드는 실행이 중단되거나 재개될 때 실행 준비 상태의 태스크들 중 가장 우선순위가 높은 태스크를 선택하고 관리할 수 있도록 하는 실행 코드이다. 태스크의 실행이 중단되었을 경우, 실시간 운영체제는 실행 중단될 태스크를 관리하기 위한 일부 데이터 정보를 변경하고 새로 실행할 태스크를 선택해야 하는데, 태스크를 중단하기 위해 소요될 시간을 블록킹 오버헤드($\triangle t_b$)로, 새로 실행할 태스크를 선택하기 위해 소요되는 시간을 선택 오버헤드($\triangle t_s$)로 구분한다. 이와 유사하게, 실행 중단된 태스크를 실행 재개하기 위해서 소모되는 시간을 언블록킹 오버헤드($\triangle t_u$)로 표현하고 언블록킹 시에도 현재 수행중인 태스크가 실행 준비 상태의 태스크보다 우선순위가 낮은 경우 문맥교환이 발생하게 되는데, 이때에도 실행 준비 상태의 태스크들 중 가장 우선순위가 높은 태스크를 찾기 위한 선택 오버헤드가 존재한다[3].

2. μ C/OS의 실시간 태스크 스케줄링 알고리즘

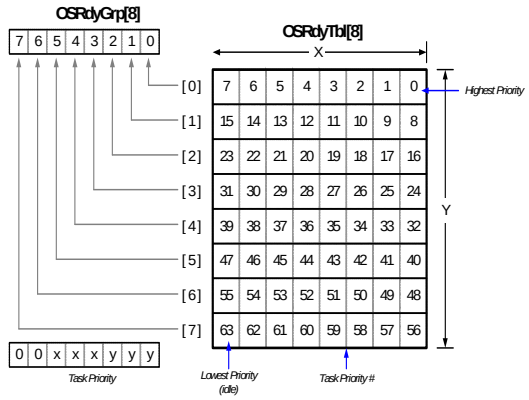
μ C/OS[4]는 맵 테이블과 언맵 테이블이라는 독창적인 데이터 구조체를 사용하여 64단계의 우선순위를 지원하는 시간 결정적인 스케줄러를 제안하였으며, 스케줄링 오버헤드($\triangle t$)는 응용프로그램에서 생성하는 태스크의 제한된 수(64개) 내에서 항상 상수의 시간을 갖는다.

즉, μ C/OS는 준비리스트에 있는 태스크들 중에 가장 우선순위가 높은 태스크를 찾기 위하여 준비리스트에 있는 모든 태스크를 비교하는 방법을 사용하지 않고 연산에 의해서 최고의 우선순위 태스크를 찾는다. 이러한 연산에 사용되는 그림 1의 우선순위 변환용 언맵 테이블은 미리 여러 개의 태스크들이 존재할 때 각각에서 가장 우선순위가 높은 우선순위의 위치를 정해 놓은 일종의 우선순위 변환 테이블이다.

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

                /* Priority Resolution Table */
  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 1. 맵 테이블과 언맵 테이블



▶▶ 그림 2. 준비리스트

시스템이 64개의 우선순위를 가질 경우 준비 그룹과 준비 테이블은 그림2와 같이 나타낼 수 있다. 수행 준비상태의 태스크는 준비테이블(OSRdyTbl[])과 준비그룹(OSRdyGrp)으로 구성된 준비리스트에 위치한다. 준비그룹은 8개의 태스크를 하나의 그룹으로 지정하며, 각 비트는 해당 그룹에 수행 준비상태의 태스크가 있는지를 나타낸다. 태스크가 수행 준비되면 그와 관련된 준비테이블의 8개 변수 중 우선순위에 대응하는 수행대기 비트를 설정한다. 수행할 태스크는 스케줄러가 언맵 테이블을 활용하여 준비그룹의 가장 낮은 비트를 선택하여 최상위 우선순위의 준비그룹을 선택하고, 그 준비그룹의 준비테이블에 설정된 가장 낮은 위치의 비트를 선택하여 최상위 우선순위의 태스크를 결정한다.

μ C/OS는 64단계의 우선순위를 지원하기 위해 맵 테이블과 언맵 테이블을 사용하여 시간 결정성을 보장하지만, 맵 테이블(8byte)과 언맵 테이블(256byte)을 위해 총 264byte의 메모리 오버헤드가 존재한다. 그러나, 태스크의 우선순위 확장에 따른 메모리 오버헤드는 2의 지수승으로 증가하는 문제점이 있으며, 이는 메모리 제약사항이 있는 내장형 시스템에 적합하지 않다.

3. 2차원 기법의 실시간 태스크 스케줄링 알고리즘

2차원 기법의 실시간 태스크 스케줄링 알고리즘[5]은 μ C/OS의 맵 테이블과 언맵 테이블을 변경없이 사용하면서, 64단계 이상의 우선순위 확장이 가능한 알고리즘이다. 하지만, 128단계 이상의 우선순위 확장 시 언맵 테이블에는 변경이 없지만, 맵 테이블의 크기가 커져 메모리 오버헤드(언맵 테이블 + 맵 테이블)가 발생하며, 그림 3의 알고리즘에 따른 성능 저하가 나타나게 된다.

```
        int i, j, a, b, c;
        i = 0, a = r − 1, b = OSRdyGrp;
loop1: if ((b & mask[2^a]) == 0)
            i = i + 2^a, b = b >> 2^a;
        else  b = b & mask[2^a];
        a = a − 1;
        if (a > 0) goto loop1;
        y = i + OSUnMapTbl[b];
        j = 0, a = r − 1, c = OSRdyTbl[y];
loop2: if ((c & mask[2^a]) == 0)
            j = j + 2^a, c = c >> 2^a;
        else  c = c & mask[2^a];
        a = a − 1;
        if (a > 0) goto loop2;
        x = j + OSUnMapTbl[c];
        p = (y << r) + x;
```

▶▶ 그림 3. 2차원 기법의 실시간 태스크 스케줄링 알고리즘

표1에서는 μ C/OS와 2-D 실시간 태스크 스케줄링 알고리즘의 우선순위 확장에 따른 메모리와 성능상의 오버헤드를 비교 분석한다.

표1. 우선순위 확장에 따른 μC/OS와 2-D 알고리즘의 실행 명령수와 메모리 오버헤드

우선순위	μC/OS		2-D	
	실행 명령수	메모리 오버헤드	실행 명령수	메모리 오버헤드
64(2^6)	8	2^8+8	8	2^8+8
256(2^8)	8	2^{16}+8	8	2^8+8
512(2^9)	8	2^{32}+32	17	2^8+32
1024(2^{10})	8	2^{32}+128	21	2^8+128
2048(2^{11})	8	2^{64}+512	29	2^8+512
4096(2^{12})	8	2^{64}+512	37	2^8+512
32768(2^{15})	8	2^{256}+8192	101	2^8+8192

Ⅲ. 다차원 기법의 실시간 태스크 스케줄링 알고리즘

본 논문에서 구현한 다차원 기법의 실시간 태스크 스케줄링 알고리즘은 태스크의 우선순위 확장에 따른 맵 테이블과 언맵 테이블의 메모리 오버헤드가 없으며, 내장형 실시간 시스템이 수용 가능한 성능 상의 오버헤드만이 발생한다.

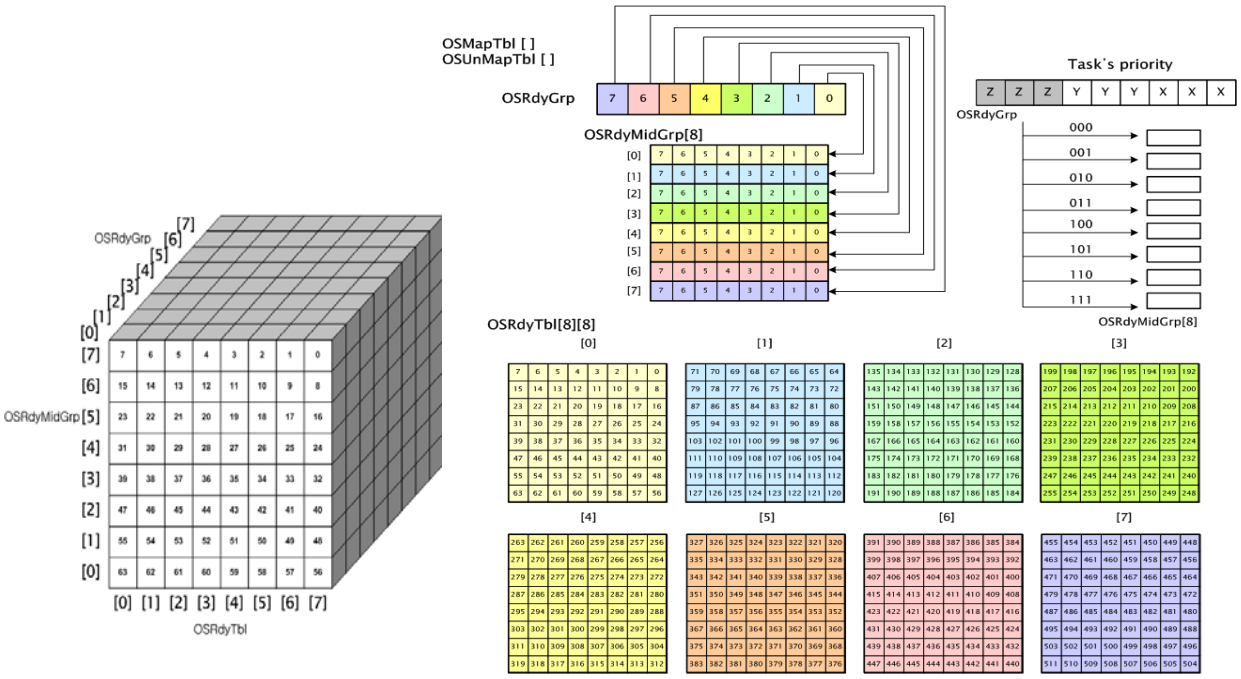
다차원 기법의 실시간 태스크 스케줄링 알고리즘은 μ C/OS의 맵 테이블과 언맵 테이블을 수정 없이 사용

하며, 우선순위 확장에 따라 그룹이 추가된다. 그림 4와 같이 512단계(3차원)로 우선순위를 확장할 경우 준비테이블(OSRdyTbl[][])과 중간준비그룹(OSRdyMidGrp[]), 준비그룹(OSRdyGrp)에 실행 준비 상태의 태스크를 삽입하며, 삽입 알고리즘은 μ C/OS에서와 유사하다.

다음 식 (1)에 의해 실행 준비상태의 태스크를 준비리스트에 추가한다.

```
OSRdyGrp              |= OSMapTbl[p >> 6];
OSRdyMidGrp[p>>6]|= OSMapTbl[(p&0x38)>>3];   (1)
OSRdyTbl[p >> 6][ (p&0x38) >> 3]
                      |= OSMapTbl[p&0x07];
```

위의 (1)식에서 p는 태스크의 우선순위이다.



▶▶ 그림 4. 3차원 기법의 준비 리스트와 전개도

다음 식 (2)는 실행 중인 태스크가 준비상태에서 제거될 때 수행되는 코드이다.

```
if((OSRdyTbl[p >> 6][ (p&0x38) >> 3]
                      &= ~OSMapTbl[p&0x07]) == 0)
{
    if( (OSRdyMidGrp[p >> 6]                    (2)
                  &=  ~OSMapTbl[(p&0x38) >> 3]) ==0)
          OSRdyGrp &= ~OSMapTbl[p >> 6];
}
```

준비리스트에서 최상위 우선순위를 선택하는 알고리즘은 다음 식 (3)과 같다.

```
z  =  OSUnMapTbl[OSRdyGrp];
y = OSUnMapTable[OSRdyMidGrp[z]];          (3)
x = OSUnMapTable[OSRdyTbl[z][y]] ;
p  =  (z << 6) + (y << 3) + x;
```

또한, 본 알고리즘을 통해 4096단계(4차원)의 우선순위로 쉽게 확장할 수 있다. 4096단계로 확장할 경우 준비테이블(OSRdyTbl[][][])과 상위중간준비그룹(OSRdyUpMidGrp[][]), 하위중간준비그룹(OSRdyLowMidGrp[]), 준비그룹(OSRdyGrp)에 실행 준비 상태의 태스크를 삽입한다.

다음 식(4)에 의해 실행 준비상태의 태스크를 준비리스트에 추가한다.

```
OSRdyGrp              |= OSMapTbl[p >> 9];
OSRdyMidGrp1[p>>9]
```

```
                |= OSMapTbl[(p&0x1C0)>>6];
OSRdyMidGrp2[p>>9] [(p&0x1C0)>>6]                    (4)
                |= OSMapTbl[(p&0x38)>>3];
OSRdyTbl[p>>9] [(p&0x1C0)>>6] [ (p&0x38) >> 3]
                |= OSMapTbl[p&0x07];
```

다음 식 (5)는 실행 중인 태스크가 준비상태에서 제거될 때 수행되는 코드이다.

```
if((OSRdyTbl[p >> 9] [(p&0x1C0)>>6] [ (p&0x38) >> 3]
                        &= ~OSMapTbl[p&0x07]) == 0)
{
    if((OSRdyMidGrp1[p >> 9][ (p&0x1C0) >> 6]
                    &= ~OSMapTbl[(p&0x38)>>3]) == 0)
    {
      if( (OSRdyMidGrp2[p >> 9]
            &= ~OSMapTable[(p&0x1C0) >> 6]) ==0)
          OSRdyGrp &= ~OSMapTbl[p >> 9];
    }
}
```

준비리스트에서 최상위 우선순위를 선택하는 알고리즘은 다음 식 (6)과 같다.

```
z = OSUnMapTbl[OSRdyGrp];
y  = OSUnMapTbl[OSRdyMidGrp1[z]];                (6)
x = OSUnMapTable[OSRdyMidGrp2[z][y]];
w = OSUnMapTable[OSRdyTbl[z][y][x]] ;
p  = (z << 9) + (y << 6) + (x << 3) + w;
```

본 논문에서 제안한 알고리즘은 우선순위 확장에 상관없이 최상위 우선순위를 결정하는데 소요되는 선택 오버헤드($\triangle t_s$)는 고정상수($O(1)$) 시간을 보장한다.

또한, 블록킹 오버헤드($\triangle t_b$)와 선택 오버헤드($\triangle t_s$), 언블록킹 오버헤드($\triangle t_u$)는 식 (1)(2)(3)(4)(5)(6)과 같이 모두 고정 코드만 수행하면 되기 때문에, 고정상수($O(1)$) 시간을 보장한다.

Ⅳ. 알고리즘 비교 분석

본 논문에서 구현한 알고리즘과 μ C/OS의 스케줄러, 2차원 기법의 알고리즘에 대해 우선순위 확장에 따른 성능 상의 오버헤드(# of inst)와 메모리 오버헤드(M · O)를 비교분석한 결과는 아래 표2와 같다.

표 2. 우선순위 확장에 따른 각 알고리즘의 오버헤드 비교

우선순위	µC/OS		2-D		M-D	
	# of inst	M·O	# of inst	M·O	# of inst	M·O
64 (2^6)	8	2^8+8	8	2^8+8	8 (2D)	2^8+8
256 (2^8)	8	2^{16}+32	39	2^8+32	14 (3D)	2^8+8
512 (2^9)	8	2^{32}+128	46	2^8+128		2^8+8
1024 (2^{10})	8	2^{32}+128	53	2^8+128	16 (4D)	2^8+8
2048 (2^{11})	8	2^{64}+512	60	2^8+512		2^8+8
4096 (2^{12})	8	2^{64}+512	67	2^8+512		2^8+8
16384 (2^{14})	8	2^{128}+2048	81	2^8+2048	23 (5D)	2^8+8

표2에서와 같이, μ C/OS의 스케줄러는 우선순위 확장에 의한 성능상의 오버헤드는 발생하지 않지만, 메모리 오버헤드는 2의 지수승 형태로 증가한다. 이에 반해, 2차원 알고리즘은 언맵 테이블로 인한 메모리 오버헤드는 없지만 맵 테이블로 인한 메모리 오버헤드와 성능 상의 오버헤드가 발생한다. 본 논문에서 구현한 다차원 기법의 알고리즘은 μ C/OS의 스케줄러 보다 근소한 성능상의 오버헤드는 발생하지만, 메모리 오버헤드는 발생하지 않는다.

Ⅴ. 결론

본 논문에서 구현한 다차원 기법의 실시간 태스크 스케줄링 알고리즘은 우선순위 확장에 따라 μ C/OS 실시간 스케줄러 보다 근소한 성능상의 오버헤드는 있지만, μ C/OS의 2의 지수승으로 증가하는 메모리 오버헤드 문제를 해결하였다. 또한, 제안 알고리즘은 기 제안된 2차원 스케줄링 알고리즘에 비해 성능면에서 우수하고 맵 테이블 증가의 메모리 오버헤드 문제를 해결하였다. 이에 따라, 본 논문에서 구현한 다차원 기법의 실시간 태스크 스케줄러는 디지털 컨버전스 제품과 같이 다수의

응용이 탑재되어 64단계 이상의 우선순위를 필요로 하는 내장형 플랫폼에 적합하다.

향후연구과제는 본 논문에서 제시한 알고리즘과 μC/OS 스케줄러, 2차원 실시간 스케줄링 알고리즘을 실시간 운영체제 상에 구현하고, 내장형 플랫폼에 탑재하여 다수의 응용을 구동시켜 실제 구동 시에 발생할 수 있는 성능상의 오버헤드에 대한 비교 분석을 수행하는 것이다.

■ 참 고 문 헌 ■

[1] K. G. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer science and engineering," Proc. of the IEEE, Vol,82, No.1, pp.6-24, 1994.

[2] C. M. Krishna and K. G. Shin, Real-Time Systems, McGraw-Hill Pub, 1997.

[3] K. Ramamritham and J. A. Stankovic, "Scheduling algorithms and operating system support for real-time systems," Proc. of IEEE, Vol.82, No.1, pp.55-67, 1996.

[4] Jean J. Labrosse, μC/OS: The Real-Time Kernel, R&D Pub, 1993.

[5] 조문행, 외., “시간 결정성을 보장하는 실시간 태스크 스케줄링”, 한국콘텐츠학회논문지 (IT기반기술), 제7권, 제1호, pp.73-82, 2007.