

# 내장형 실시간 운영체제에서 다차원 기법을 이용한 실시간 태스크 스케줄링 알고리즘

## Real-Time Task Scheduling Algorithm using a Multi-Dimensional Methodology for Embedded Real-Time Operating Systems

조문행\*, 임재석\*, 이진욱\*, 김주만\*\*, 이철훈\*  
충남대학교 컴퓨터공학과\*, 부산대학교 바이오정보전자공학과\*\*

Moon-Haeng Cho(root4567@cnu.ac.kr)\*, Jae-Seok Lim(jslim@cnu.ac.kr)\*,  
Jin-Wook Lee(ljweve@cnu.ac.kr)\*, Joo-Man Kim(joomkim@pusan.ac.kr)\*\*,  
Cheol-Hoon Lee(clee@cnu.ac.kr)\*

### 요약

오늘날의 핸드폰, PMP, 지능형 가정기기, 자동차 엔진 컨트롤 시스템과 같은 내장형 시스템은 인간의 삶과 일, 놀이 등 살아가는 환경에 대한 전환을 가져오고 있다. 사용자가 요구하는 다양한 서비스들을 지원하기 위해 내장형 시스템 응용들의 복잡성이 증가하고 있으며, 그런 내장형 컴퓨팅 플랫폼에는 시간결정성을 갖는 실시간 운영체제를 사용해야 한다. 이런 실시간 운영체제의 커널 서비스들은 각 서비스 수행에 얼마만큼의 시간이 소요되는지를 명세하여 시간결정성을 보장하면서 작은 실행 이미지 크기를 가져야만 한다. 이런 정보를 토대로, 응용 개발자들은 각 태스크가 마감시간을 준수하도록 하는 실시간 응용 소프트웨어를 설계할 수 있다. 본 논문에서는 추가적인 메모리 오버헤드 없이 2 $\tau$  레벨의 우선순위를 갖는 시스템에서 다차원 기법을 이용하여 고정상수 시간 내에 최상위 우선순위를 결정하는 완전 일반화된 실시간 스케줄링 알고리즘을 제안한다.

■ 중심어 : | 내장형 시스템 | 실시간 운영체제 | 실시간 스케줄링 알고리즘 | 시간결정성 |

### 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, services application to guarantee various requirements of users become increasingly sophisticated and complicated, such embedded computing platforms 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. Kernel services should also be deterministic by specifying how long each service call will take to execute. Having this information allows the application designers 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 real-time scheduling algorithm using multi-dimensional methodology to determine the highest priority in the ready list with 2 $\tau$  levels of priorities in a constant time without additional memory overhead.

■ keyword : | Embedded Systems | Real-Time Operating Systems | Real-Time Scheduling Algorithm | Determinism |

\* 본 연구는 지식경제부의 IT R&D 지원으로 수행되었습니다.[2008-F-048, 웨어러블 컴퍼니언 개발 사업].  
접수번호 : #090901-006  
접수일자 : 2009년 09월 01일  
심사완료일 : 2009년 11월 23일  
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

## 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].

- 경성(hard) 실시간 시스템 : 시스템이 주어진 종료 시한을 만족시키지 못한 경우에 막대한 재산적 손실이나 인명의 피해를 주는 시스템.
- 연성(soft) 실시간 시스템 : 온라인 시스템과 같이 시간제약 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 기법을 이용한 실시간 태스크 스케줄링 알고리즘을 소개한다.

본 논문의 구성은, 관련연구에서 실시간 태스크 스케줄링 알고리즘 오버헤드와  $\mu\text{C}/\text{OS}$ 의 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 알고리즘을 제안한다. 4장에서는 제안한 알고리즘과 기존 알고리즘을 비교 분석하고 마지막 장에서 결론을 맺는다.

## II. 관련연구

### 1. 실시간 태스크 스케줄링 알고리즘 오버헤드

태스크 스케줄링 오버헤드( $\Delta t$ )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 블로킹 오버헤드( $\Delta t_b$ )로, 새로 실행할 태스크를 선택하기 위해 소요되는 시간을 선택 오버헤드( $\Delta t_s$ )로 구분한다. 이와 유사하게, 실행 중단된 태스크를 실행 재개하기 위해서 소모되는 시간을 언블로킹 오버헤드( $\Delta t_u$ )로 표현하고 언블로킹 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[2][3].

### 2. $\mu\text{C}/\text{OS}$ 의 실시간 태스크 스케줄링 알고리즘

$\mu\text{C}/\text{OS}$ [4][5]는 맵 테이블과 언맵 테이블이라는 독창적인 데이터 구조체를 사용하여 64단계의 우선순위를 지원하는 시간 결정적인 스케줄러를 제안하였으며, 스케줄링 오버헤드( $\Delta t$ )는 응용프로그램에서 생성하는 태스크의 제한된 수(64개) 내에서 항상 상수의 시간을 갖는다.

즉,  $\mu\text{C}/\text{OS}$ 는 준비리스트에 있는 태스크들 중에 가장 우선순위가 높은 태스크를 찾기 위하여 준비리스트에 있는 모든 태스크를 비교하는 방법을 사용하지 않고 연

산에 의해서 최고의 우선순위 태스크를 찾는다. 이러한 연산에 사용되는 [그림 1]의 우선순위 변환용 언맵 테이블은 여러 개의 태스크들이 존재할 때 가장 우선순위가 높은 태스크의 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

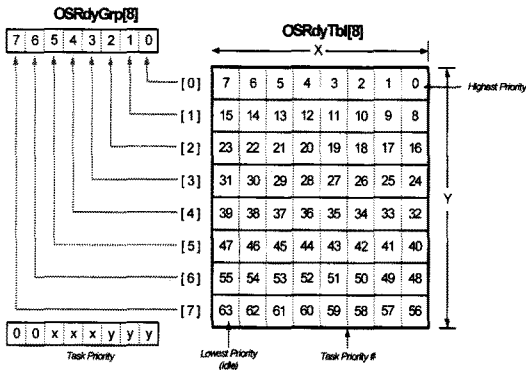


그림 2. 준비리스트

시스템이 64개의 우선순위를 가질 경우 준비 그룹과 준비 테이블은 [그림 2]와 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 비트 중 우선순위에 대응하는 수행 대기 비트를 설정한다. 실행 중지된 태스크가 해제될 때 또는 수행 중에 보다 더 높은 우선순위의 태스크에 의해 CPU 점유권을 빼앗긴 경우에, 수식 (1)과 같은 몇 개의 코드 연산에 의해서 준비리스트에 수행준비 상태임을 표시한다.

$$\begin{aligned}
 \text{OSRdyGrp} &|= \text{OSMapTbl}[p \gg 3]; \\
 \text{OSRdyTbl}[p \gg 3] &|= \text{OSMapTbl}[p \& 0x07];
 \end{aligned}
 \tag{1}$$

$p$ 는 태스크의 우선순위이며, 이러한 연산에 의한 방법은 중단된 태스크의 해제 동작이 고정된 수의 명령어 처리에 의해 수행됨을 의미한다.

수행이 중단되거나 더 높은 우선권의 태스크가 준비상태가 되는 등의 사건이 발생하여 우선순위가  $p$ 인 태스크를 준비리스트에서 제거하는 경우에는 수식 (2)와 같은 코드의 연산에 의해서 제거된다.

$$\begin{aligned}
 \text{if}((\text{OSRdyTbl}[p \gg 3] \& \sim \text{OSMapTbl}[p \& 0x07]) == 0) \\
 \text{OSRdyGrp} \&= \sim \text{OSMapTbl}[p \gg 3];
 \end{aligned}
 \tag{2}$$

위의 코드를 살펴보면, OSRdyTbl[p >> 3]에서 중지된 태스크의 수행준비 비트를 지우고 해당 그룹의 모든 태스크들이 수행대기상태가 아니면(즉, OSRdyTbl[p >> 3]이 0일 때), 준비그룹의 해당되는 그룹 비트를 지운다.

준비리스트에 있는 태스크들 중에 가장 우선순위가 높은 태스크를 찾기 위하여 준비리스트에 있는 모든 태스크를 비교하는 방법을 사용하지 않고 연산에 의해서 최고의 우선순위 태스크를 찾는다. 즉, 스케줄러가 언맵 테이블을 활용하여 준비그룹(OSRdyGrp)의 가장 낮은 비트를 선택하고 최상위 우선순위의 준비그룹을 결정한 후, 그 준비그룹의 준비테이블(OSRdyTbl[])에서 언맵 테이블을 이용하여 설정된 가장 낮은 위치의 비트를

얻어내어 최상위 우선순위의 태스크를 결정한다.

$$\begin{aligned}
 y &= \text{OSUnMapTbl}[\text{OSRdyGrp}]; \\
 x &= \text{OSUnMapTbl}[\text{OSRdyTbl}[y]]; \\
 p &= (y \lll 3) + x;
 \end{aligned}
 \tag{3}$$

수행준비 중인 태스크들 중 가장 높은 우선순위 태스크를 결정하는 것은 태스크의 수에 따라 변하는 탐색방법이 아닌 수식 (3)과 같이 고정된 몇 개의 명령어 처리를 수행하는 것만으로 가능하다. 즉 이러한 연산을 이용한 태스크 스케줄링 방법은 응용 프로그램에서 생성된 태스크의 수에 관계없이 일정하므로, 스케줄러는 시간결정성을 갖는다.

μC/OS는 64단계의 우선순위를 지원하기 위해 맵 테이블과 언맵 테이블을 사용하여 시간 결정성을 보장한다. 하지만, 태스크의 우선순위 확장에 따른 메모리 오버헤드는 [표 1]과 같이 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 [6][7]은 μC/OS의 맵 테이블과 언맵 테이블을 변경없이 사용하면서, 64단계 이상의 우선순위 확장이 가능한 알고리즘이다. 준비리스트에 삽입하는 알고리즘은 수식 (4)와 같이 고정된 수의 명령어 처리에 의해 수행된다.

$$\begin{aligned}
 \text{OSRdyGrp} &|= \text{OSMapTbl}[p \ggg 4]; \\
 \text{OSRdyTbl}[p \ggg 4] &|= \text{OSMapTbl}[p \& 0x0F];
 \end{aligned}
 \tag{4}$$

또한, 준비리스트에서 삭제 알고리즘은 수식 (5)와 같이 고정된 수의 명령어 처리에 의해 수행된다.

$$\begin{aligned}
 \text{if}((\text{OSRdyTbl}[p \ggg 4] \&= \sim \text{OSMapTbl}[p \& 0x0F]) == 0) \\
 \text{OSRdyGrp} \&= \sim \text{OSMapTbl}[p \ggg 4];
 \end{aligned}
 \tag{5}$$

하지만, 128단계 이상의 우선순위 확장 시 언맵 테이블에는 변경이 없지만, 맵 테이블의 크기가 커져 메모리 오버헤드(언맵 테이블 + 맵 테이블)가 발생하며, 최상위 우선순위를 선택하는데 있어서 아래의 알고리즘 [8][9]에 따른 성능 저하가 나타나게 된다.

```

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

[표 1]은 μC/OS와 2-D 실시간 태스크 스케줄링 알고리즘의 우선순위 확장에 따른 메모리와 성능상의 오버헤드이다.

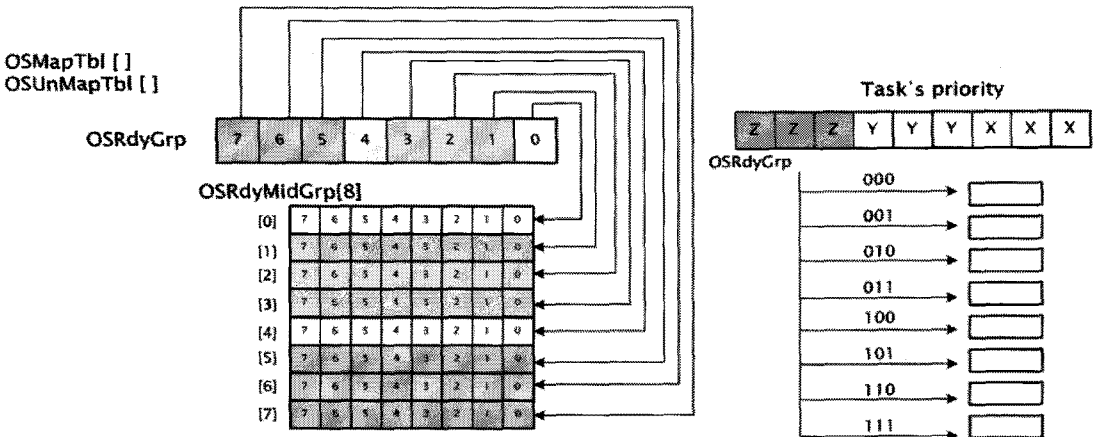
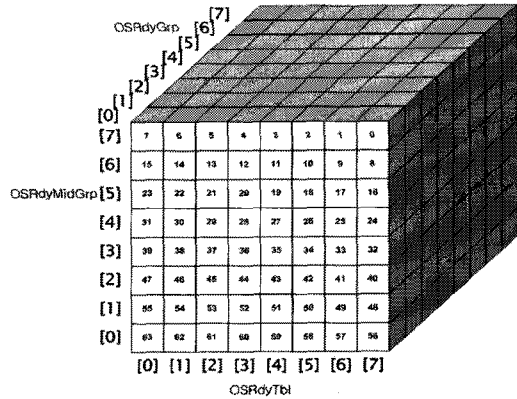
표 1. 우선순위 확장에 따른 μC/OS와 2-D 알고리즘의 실행 명령수와 메모리 오버헤드

우선순위	μC/OS		2-D	
	실행 명령수	메모리 오버헤드	실행 명령수	메모리 오버헤드
64(2 <sup>6</sup> )	8	2 <sup>8</sup> +8	8	2 <sup>8</sup> +8
256(2 <sup>8</sup> )	8	2 <sup>16</sup> +8	39	2 <sup>8</sup> +32
512(2 <sup>9</sup> )	8	2 <sup>32</sup> +32	46	2 <sup>8</sup> +128
1024(2 <sup>10</sup> )	8	2 <sup>32</sup> +128	53	2 <sup>8</sup> +128
2048(2 <sup>11</sup> )	8	2 <sup>64</sup> +512	60	2 <sup>8</sup> +512
4096(2 <sup>12</sup> )	8	2 <sup>64</sup> +512	67	2 <sup>8</sup> +512
16384(2 <sup>14</sup> )	8	2 <sup>128</sup> +2048	81	2 <sup>8</sup> +2048

### 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  $\mu$ C/OS의 맵 테이블과 언맵 테이블을 수정 없이 사용하며, 우선순위 확장에 따라 중간 그룹들이 추가된다.



OSRdyTbl[8][8]

[0]	[1]	[2]	[3]
7 6 5 4 3 2 1 0	71 70 69 68 67 66 65 64	135 134 133 132 131 130 129 128	196 196 197 196 195 194 193 192
15 14 13 12 11 10 9 8	79 78 77 76 75 74 73 72	143 142 141 140 139 138 137 136	207 206 205 204 203 202 201 200
23 22 21 20 19 18 17 16	87 86 85 84 83 82 81 80	151 150 149 148 147 146 145 144	215 214 213 212 211 210 209 208
31 30 29 28 27 26 25 24	95 94 93 92 91 90 89 88	159 158 157 156 155 154 153 152	223 222 221 220 219 218 217 216
39 38 37 36 35 34 33 32	103 102 101 100 99 98 97 96	167 166 165 164 163 162 161 160	231 230 229 228 227 226 225 224
47 46 45 44 43 42 41 40	111 110 109 108 107 106 105 104	175 174 173 172 171 170 169 168	239 238 237 236 235 234 233 232
55 54 53 52 51 50 49 48	119 118 117 116 115 114 113 112	183 182 181 180 179 178 177 176	247 246 245 244 243 242 241 240
63 62 61 60 59 58 57 56	127 126 125 124 123 122 121 120	191 190 189 188 187 186 185 184	255 254 253 252 251 250 249 248
[4]	[5]	[6]	[7]
263 262 261 260 259 258 257 256	327 326 325 324 323 322 321 320	391 390 389 388 387 386 385 384	455 454 453 452 451 450 449 448
271 270 269 268 267 266 265 264	335 334 333 332 331 330 329 328	399 398 397 396 395 394 393 392	463 462 461 460 459 458 457 456
279 278 277 276 275 274 273 272	343 342 341 340 339 338 337 336	407 406 405 404 403 402 401 400	471 470 469 468 467 466 465 464
287 286 285 284 283 282 281 280	351 350 349 348 347 346 345 344	415 414 413 412 411 410 409 408	479 478 477 476 475 474 473 472
295 294 293 292 291 290 289 288	359 358 357 356 355 354 353 352	423 422 421 420 419 418 417 416	487 486 485 484 483 482 481 480
303 302 301 300 299 298 297 296	367 366 365 364 363 362 361 360	431 430 429 428 427 426 425 424	495 494 493 492 491 490 489 488
311 310 309 308 307 306 305 304	375 374 373 372 371 370 369 368	439 438 437 436 435 434 433 432	503 502 501 500 499 498 497 496
319 318 317 316 315 314 313 312	383 382 381 380 379 378 377 376	447 446 445 444 443 442 441 440	511 510 509 508 507 506 505 504

그림 3. 3차원 기법의 준비리스트와 전개도

1. 3차원 기법의 실시간 스케줄링 알고리즘

512단계의 우선순위를 지원하기 위해서는 [그림 3]과 같이 3차원 형태의 준비리스트가 필요하다. 512단계로 우선순위를 확장할 경우 준비테이블(OSRdyTbl[[]])과 중간준비그룹(OSRdyMidGrp[]), 준비그룹(OSRdyGrp)에 실행 준비 상태의 태스크를 삽입하며, 삽입 알고리즘은  $\mu\text{C}/\text{OS}$ 에서와 유사하다.

다음 수식 (6)에 의해 실행 준비상태의 태스크를 준비리스트에 추가하며, 고정된 수의 명령어 처리에 의해 수행된다.

$$\begin{aligned}
 \text{OSRdyGrp} &|= \text{OSMapTbl}[p \gg 6]; \\
 \text{OSRdyMidGrp}[p \gg 6] &|= \text{OSMapTbl}[(p \& 0x38) \gg 3]; \\
 \text{OSRdyTbl}[p \gg 6][ (p \& 0x38) \gg 3] &|= \text{OSMapTbl}[p \& 0x07];
 \end{aligned} \tag{6}$$

다음 수식 (7)은 실행 중인 태스크가 준비상태에서 제거될 때 수행코드로 고정된 수의 명령어 처리에 의해 수행된다.

$$\begin{aligned}
 &\text{if}((\text{OSRdyTbl}[p \gg 6][ (p \& 0x38) \gg 3] \\
 &\quad \&= \sim \text{OSMapTbl}[p \& 0x07]) == 0) \\
 &\{ \\
 &\quad \text{if}((\text{OSRdyMidGrp}[p \gg 6] \\
 &\quad \quad \&= \sim \text{OSMapTbl}[(p \& 0x38) \gg 3]) == 0) \\
 &\quad \quad \text{OSRdyGrp} \&= \sim \text{OSMapTbl}[p \gg 6]; \\
 &\}
 \end{aligned} \tag{7}$$

준비리스트에서 최상위 우선순위를 선택하는 알고리즘은 다음 수식 (8)과 같으며, 고정된 수의 명령어 처리에 의해 수행된다.

$$\begin{aligned}
 z &= \text{OSUnMapTbl}[\text{OSRdyGrp}]; \\
 y &= \text{OSUnMapTbl}[\text{OSRdyMidGrp}[z]]; \\
 x &= \text{OSUnMapTbl}[\text{OSRdyTbl}[z][y]]; \\
 p &= (z \ll 6) + (y \ll 3) + x;
 \end{aligned} \tag{8}$$

2. 4차원 기법의 실시간 스케줄링 알고리즘

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

다음 수식(9)에 의해 실행 준비상태의 태스크를 준비리스트에 추가한다.

$$\begin{aligned}
 \text{OSRdyGrp} &|= \text{OSMapTbl}[p \gg 9]; \\
 \text{OSRdyHiMidGrp}[p \gg 9] &|= \text{OSMapTbl}[(p \& 0x1C0) \gg 6]; \\
 \text{OSRdyLoMidGrp}[p \gg 9][ (p \& 0x1C0) \gg 6] &|= \text{OSMapTbl}[p \& 0x07]; \\
 \text{OSRdyTbl}[p \gg 9][ (p \& 0x1C0) \gg 6][ (p \& 0x38) \gg 3] &|= \text{OSMapTbl}[p \& 0x07];
 \end{aligned} \tag{9}$$

다음 수식 (10)은 실행 중인 태스크가 준비상태에서 제거될 때 수행되는 코드이다.

$$\begin{aligned}
 &\text{if}((\text{OSRdyTbl}[p \gg 9][ (p \& 0x1C0) \gg 6][ (p \& 0x38) \gg 3] \\
 &\quad \&= \sim \text{OSMapTbl}[p \& 0x07]) == 0) \\
 &\{ \\
 &\quad \text{if}((\text{OSRdyLoMidGrp}[p \gg 9][ (p \& 0x1C0) \gg 6] \\
 &\quad \quad \&= \sim \text{OSMapTbl}[(p \& 0x38) \gg 3]) == 0) \\
 &\quad \quad \{ \\
 &\quad \quad \quad \text{if}((\text{OSRdyHiMidGrp}[p \gg 9] \\
 &\quad \quad \quad \quad \&= \sim \text{OSMapTbl}[(p \& 0x1C0) \gg 6]) == 0) \\
 &\quad \quad \quad \quad \text{OSRdyGrp} \&= \sim \text{OSMapTbl}[p \gg 9]; \\
 &\quad \quad \quad \} \\
 &\quad \} \\
 &\}
 \end{aligned} \tag{10}$$

준비리스트에서 최상위 우선순위를 선택하는 알고리즘은 다음 수식 (11)과 같다.

$$\begin{aligned}
 z &= \text{OSUnMapTbl}[\text{OSRdyGrp}]; \\
 y &= \text{OSUnMapTbl}[\text{OSRdyHiMidGrp}[z]]; \\
 x &= \text{OSUnMapTbl}[\text{OSRdyLoMidGrp}[z][y]]; \\
 w &= \text{OSUnMapTbl}[\text{OSRdyTbl}[z][y][x]]; \\
 p &= (z \ll 9) + (y \ll 6) + (x \ll 3) + w;
 \end{aligned} \tag{11}$$

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

또한, 블록킹 오버헤드( $\Delta t_b$ )와 선택 오버헤드( $\Delta t_s$ ), 언블록킹 오버헤드( $\Delta t_u$ )는 식 (6)~(11)과 같이 모두 고정 코드만 수행하면 되기 때문에, 고정상수( $O(1)$ ) 시간을 보장한다.

#### IV. 알고리즘 비교 분석

본 논문에서 구현한 알고리즘과  $\mu$ C/OS의 스케줄러, 2차원 기법의 알고리즘에 대해 우선순위 확장에 따른 성능 상의 오버헤드인 실행 명령수(# of inst)와 메모리 오버헤드(M·O)를 비교분석한 결과는 아래 [표 2]와 같다.

표 2. 우선순위 확장에 따른 각 알고리즘의 오버헤드 비교

우선순위	$\mu$ C/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)	$2^8+8$

[표 2]에서의와 같이,  $\mu$ C/OS의 스케줄러는 우선순위 확장에 의한 성능상의 오버헤드는 발생하지 않지만, 메모리 오버헤드는 2의 지수승 형태로 증가한다. 이는 메모리 자원이 한정된 내장형 시스템에  $\mu$ C/OS의 알고리즘을 사용하여 시간결정성을 보장하는 것은 적합하지 않다는 것을 의미한다. 이에 반해, 2차원 알고리즘은 언맵 테이블로 인한 메모리 오버헤드는 없지만 맵 테이블로 인한 메모리 오버헤드와 성능 상의 오버헤드가 발생하

지만 내장형 시스템에 적용은 가능하다.

본 논문에서 구현한 다차원 기법의 알고리즘은  $\mu$ C/OS의 스케줄러 보다 근소한 성능상의 오버헤드는 발생하지만, 언맵 테이블로 인한 메모리 오버헤드는 발생하지 않는다. 또한, 2차원 기법의 알고리즘 보다 메모리 오버헤드가 적고 성능상의 오버헤드 또한 적다는 것을 알 수 있다. 그러므로, 본 논문에서 구현한 다차원 기법의 실시간 스케줄링 알고리즘은 메모리 자원이 한정된 내장형 시스템에 적용하기에 적합하다.

#### V. 결론

내장형 실시간 시스템은 기존의 산업, 군사 분야에서 시스템의 소형화, 성능 증가와 발맞추어 많은 양의 정보처리 기능을 가진 다양한 응용 분야로 발전하고 있으며, 그 적용사례도 다기능의 정보가전, 휴대폰, PDA, PMP, MP3 등과 같은 휴대용 기기로 확대되고 있다. 이들 실시간 응용장치들이 복잡한 응용프로그램을 구동함에 따라 실시간 운영체제의 시스템 콜 수행의 정확한 시간과 각 태스크의 마감시간내의 수행을 요구하고 있다. 또한, 시스템의 업무수행 과정 중 스케줄링은 가장 빈번이 일어나며 이 특성에 따라서 시스템 성능에 영향을 줄 수 있다.

본 논문에서 구현한 다차원 기법의 실시간 태스크 스케줄링 알고리즘은 우선순위 확장에 따라  $\mu$ C/OS 실시간 스케줄러 보다 근소한 성능상의 오버헤드는 있지만,  $\mu$ C/OS의 2의 지수승으로 증가하는 메모리 오버헤드 문제를 해결하였다. 또한, 제안 알고리즘은 기 제안된 2차원 스케줄링 알고리즘에 비해 성능면에서 우수하고 맵 테이블 증가에 따른 메모리 오버헤드 문제를 해결하였다. 이에 따라, 본 논문에서 구현한 다차원 기법의 실시간 태스크 스케줄러는 디지털 컨버전스 제품과 같이 다수의 응용이 탑재되어 다수 단계의 우선순위를 필요로 하는 내장형 플랫폼에 적합하다.

향후연구과제는 본 논문에서 제시한 알고리즘과  $\mu$ C/OS 스케줄러, 2차원 스케줄링 알고리즘을 실시간 운영체제 상에 구현하여 실제 구동 시에 발생할 수 있는 성능상의 오버헤드에 대해 비교 분석하는 것이다.

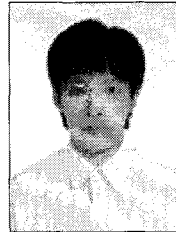
참고문헌

- [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] L. Cucu, R. Kocik and Y. Sorel, "Real-time scheduling for systems with precedence, periodicity and latency constraints," RTS Embedded Systems 2002, Paris March, pp.26-28, 2002.
- [4] Jean J. Labrosse, *μC/OS: The Real-Time Kernel*, R&D Publications, Lawrence, 1993.
- [5] Jean J. Labrosse, *μC/OS II: The Real Time Kernel 2nd Edition*, R&D Publications, Lawrence, 2002.
- [6] S. J. Oh and C. H. Lee. "Deterministic Task Scheduling for Real-Time GPS Controllers," GNSS Workshop, Vol.8, pp.36-39, 2001(11).
- [7] S.-J. Oh, "Deterministic Task Scheduling for Embedded Real-Time Operating Systems," IEICE Trans. Inf. & Syst., Vol.E87-D, No.2, pp.123-126m, 2004(2).
- [8] Myoung-Jo Jung, "Generalized Deterministic Task Scheduling algorithm for Embedded Real-Time Operating Systems," in Proc. The 2006 International Conference on Embedded Systems & Applications ESA'06, pp.79-82, 2006(6).
- [9] 조문행, "시간 결정성을 보장하는 실시간 태스크 스케줄링", 한국콘텐츠학회논문지 (IT기반기술), 제7권, 제1호, pp.73-82, 2007.

저자소개

조문행(Moon-Haeng Cho)

정회원

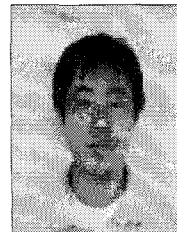


- 2004년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2006년 2월 : 충남대학교 컴퓨터 공학과(공학석사)
- 2006년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 박사과정 재학

<관심분야> : 실시간 컴퓨팅, 실시간 운영체제, 초소형 초절전 실시간 운영체제

임재석(Jae-Seok Lim)

준회원



- 2008년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2008년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 내장형 시스템, 로봇 미들웨어

이진욱(Jin-Wook Lee)

준회원



- 2009년 2월 : 충남대학교 컴퓨터 공학과(공학사)
- 2009년 3월 ~ 현재 : 충남대학교 컴퓨터공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 내장형 시스템, 로봇 미들웨어

김주만(Joo-Man Kim)

정회원



- 1984년 : 숭실대학교 전자계산학(공학사)
- 1998년 : 충남대학교 컴퓨터공학과(공학석사)
- 2003년 : 충남대학교 컴퓨터공학과(공학박사)



- 1985년 1월 ~ 2000년 2월 : ETRI 책임연구원(운영체제연구팀장)
- 2000년 3월 ~ 2006년 2월 : 밀양대학교 정보통신공학부 교수
- 2006년 3월 ~ 현재 : 부산대학교 바이오정보전자 교수

<관심분야> : 실시간 임베디드 시스템, 바이오 센서 제어, 유비쿼터스 컴퓨팅

이철훈(Cheol-Hoon Lee)

정회원



- 1983년 2월 : 서울대학교 전자공학과(공학사)
- 1988년 2월 : 한국과학기술원 전기및전자공학과(공학석사)
- 1992년 2월 : 한국과학기술원 전기및전자공학과(공학박사)

- 1994년 2월 ~ 1995년 2월 : Univ. of Michigan 객원연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙연구원

<관심분야> : 실시간시스템, 운영체제, 고장허용 컴퓨팅