Journal of the KIMST, Vol. 19, No. 3, pp. 371-378, 2016
ISSN 1598-9127
DOI http://dx.doi.org/10.9766/KIMST.2016.19.3.371

Research Paper 정보 · 통신 부문

# 스케줄링 가능성 분석을 통한 무장모의기 확장 설계 및 분석

장택수*,1) · 김용호2) · 나범철1) · 박근국1)

1) LIG넥스원㈜ S/W연구센터
2) 국방과학연구소 제1기술연구본부

# Design and Analysis of Weapon Simulator using Schedulability Analysis

Tasksoo Jang*,1) · Yongho Kim2) · Beomcheol Na1) · Keunkuk Park1)

1) *S/W R&D Lab, LIG Nex1 Co., Ltd., Korea*
2) *The 1st Research and Development Institute, Agency for Defense Development, Korea*

(Received 23 October 2015 / Revised 5 February 2016 / Accepted 29 April 2016)

ABSTRACT

The most important things in real-time systems are that a system guarantees to meet its deadline and to operate in its predictable range. When we design a real-time system, we need to verify whether the system can meet its deadline through schedulability analysis. There are several kinds of schedulability analysis technique for fixed priority scheduling systems. But as we all know, we can't perform schedulability analysis in design time because we can't estimate upper bounds on execution time of each task. So we used a similar real-time system to estimate upper bounds on execution time for our system, and then we performed schedulability analysis and verified that our system designed can meet its deadline.

Key Words : Schedulability Analysis(스케줄링 가능성 분석), Fixed Priority Scheduling(고정 우선순위 스케줄링), Real-Time System(실시간시스템)

## 1. 서 론

실시간시스템에서 가장 중요한 것은 시스템이 정해진 데드라인을 만족하면서 예측 가능한 범위에서 동작함을 보장하는 것이다. 이를 위해 시스템 설계 시 스케줄링 가능성 분석을 수행하여 시스템의 데드라인 만족 여부를 검증할 필요가 있다.

실시간시스템에 대한 스케줄링 가능성 분석을 수행하는 방법으로 고정 우선순위 스케줄링 시스템에서 사용률[1]을 기반으로 판단하거나 최대응답시간[2-4]을 기반으로 판단하는 방법이 있다. 사용률 기반의 스케줄링 가능성 분석은 분석 대상 태스크들에 의한 전체 CPU 사용률을 계산하여 데드라인 만족 여부를 검증한다.

* Corresponding author, E-mail: taeksoo.jang@lignex1.com

최대응답시간 기반의 스케줄링 가능성 분석은 분석 대상 태스크들의 개별적인 최대응답시간을 계산하여 데드라인과 비교함으로써 데드라인 만족 여부를 검증한다.

이 방법들을 사용하여 스케줄링 가능성을 분석하려면 시스템에서 수행되는 태스크들에 대한 주기, 우선순위, 실행시간 및 데드라인 정보가 필요하다. 보통 각 태스크의 주기, 우선순위 및 데드라인은 설계 단계나 그 이전에 정해진다. 하지만 실행시간은 시스템 구현이 어느 정도 진행되고 실행 가능한 코드가 있어야 추정이 가능하다. 따라서 설계단계에서 실시간시스템에 대한 스케줄링 가능성 분석은 사실상 불가한 경우가 많다.

본 논문에서는 실시간시스템의 설계 단계에서 기능이 유사한 실시간시스템의 데이터를 기반으로 스케줄링 가능성 분석을 수행한다. 이를 통해 설계한 시스템의 데드라인 만족 여부를 검사하고 설계의 타당성을 확인한다. 스케줄링 가능성 분석의 궁극적인 대상은 무장 4조를 모의하는 무장모의기이며 분석 데이터를 얻기 위한 유사 실시간시스템은 무장 1조를 모의하는 무장모의기이다.

2장에서는 고정 우선순위 스케줄링에서 스케줄링 가능성 분석을 수행하는 방법과 실행시간 측정 방법을 소개한다. 3장에서는 유사 실시간시스템에 대한 소개와 스케줄링 가능성 분석을 수행하고 이를 기반으로 4장에서 확장 설계된 시스템에 대한 스케줄링 가능성 분석을 수행한 후 5장에서 결론을 맺는다.

## 2. 관련 연구

### 2.1 고정 우선순위 스케줄링

고정 우선순위 스케줄링에서 스케줄러는 실행 가능한 태스크 중 가장 높은 우선순위의 태스크가 실행될 수 있도록 처리한다. 예를들어, 낮은 우선순위 태스크가 실행되고 있을 때 높은 우선순위 태스크가 실행 가능한 상태가 된 경우(외부 이벤트 발생 또는 실행 주기 도래), 스케줄러는 낮은 우선순위 태스크의 실행을 중지시키고 높은 우선순위 태스크가 실행을 시작하도록 처리한다. 높은 우선순위 태스크가 실행되고 있을 때 중간 우선순위 태스크가 실행 가능한 상태가 된 경우, 스케줄러는 높은 우선순위 태스크의 실행 상태 유지하고 중간 우선순위 태스크는 실행 가능한 상태로 대기시킨다. 이후 높은 우선순위 태스크가 실행을 완료하면 스케줄러는 중간 우선순위 태스크가 실행을 시작하도록 처리한다. 얼마 후 중간 우선순위 태스크도 실행을 완료하면 낮은 우선순위 태스크가 실행을 재개하도록 처리한다.

고정 우선순위 기반의 실시간시스템에서 태스크의 우선순위를 할당하는 방식으로 Rate Monotonic[1,6]과 Deadline Monotonic[2,6] 방식이 있다.

### 2.2 스케줄링 가능성 분석

#### 2.2.1 사용률 기반 스케줄링 가능성 분석

태스크의 우선순위 할당에서 Rate Monotonic 방식은 태스크 우선순위 할당 시 주기가 짧을수록 더 높은 우선순위를 부여하는 방법이다. 스케줄링 가능성 검사를 위한 우선순위 할당 시 모든 태스크의 우선순위는 유일해야 하므로, 동일한 주기의 태스크가 존재하는 경우 임의로 다르게 할당하도록 한다.

Rate Monotonic 방식으로 태스크의 우선순위를 할당하고 고정 우선순위의 선점 스케줄링으로 태스크를 스케줄링할 때 식 (1)을 만족하는 경우 모든 태스크는 데드라인을 만족시킨다[1,6]. 식 (1)의 적용 조건은 모든 태스크는 1) 주기적이고 2) 데드라인이 주기와 동일하며 3) 서로 독립적이고 자료를 공유하지 않는 경우이다.

$$\sum_{i=1}^{i=n}\left(\frac{C_i}{T_i}\right) \le n(2^{\frac{1}{n}}-1) \quad (1)$$

n은 시스템의 태스크 개수, $C_i$는 태스크 i의 최대 실행시간, $T_i$는 태스크 i의 주기이다. 부등식의 왼쪽은 태스크의 사용률이고, 오른쪽은 사용률 한계(Utilization Bound)이다.

태스크 개수에 따른 사용률 한계는 Table 1과 같다. n의 개수가 ∞인 경우 사용률 한계는 ln2 ≈ 0.6931이 된다. 즉, CPU 사용률이 69.31 % 보다 작다면 모든 태스크는 데드라인을 만족한다고 말할 수 있다. 스케줄링 가능성 분석 시 주의할 점은 사용률이 한계를 초과하는 경우 데드라인을 만족하는지 아닌지를 판단할 수 없다는 것이다. 즉, 사용률 한도가 넘어도 데드라인을 만족하는 경우가 있음을 유의해야 한다.

정확한 분석은 아니지만 주기와 데드라인이 일치하지 않는 경우에는 다음 식을 사용하여 계산할 수도 있다. 하지만 사용률이 식 (1)에 비해 과도하게 나올

수도 있기 때문에 최대응답시간 기반의 스케줄링 가능성 분석을 사용하는 것이 좀 더 정확한 분석일 것이다.

$$\sum_{i=1}^{i=n}\left(\frac{C_i}{D_i}\right) \le n(2^{\frac{1}{n}}-1) \quad (2)$$

Table 1. Utilization limit of rate monotonic scheduling

| 태스크 수 | 사용률 한계 | 태스크 수 | 사용률 한계 |
|---|---|---|---|
| 2 | 82.84 % | 13 | 71.20 % |
| 3 | 77.98 % | 14 | 71.06 % |
| 4 | 75.68 % | 15 | 70.94 % |
| 5 | 74.35 % | 16 | 70.84 % |
| 6 | 73.48 % | 17 | 70.75 % |
| 7 | 72.86 % | 18 | 70.60 % |
| 8 | 72.41 % | 19 | 70.59 % |
| 9 | 72.05 % | 20 | 70.53 % |
| 10 | 71.77 % | 30 | 70.12 % |
| 11 | 71.55 % | 40 | 69.92 % |
| 12 | 71.36 % | ∞ | 69.31 % |

#### 2.2.2 최대응답시간 기반 스케줄링 가능성 분석

태스크의 최대응답시간을 계산한다면 스케줄링 가능성 분석은 단순히 각 태스크별로 최대응답시간이 데드라인보다 작음(R ≤ D)을 확인하기만 하면 된다[2,3,6]. 이 방법은 사용률 기반 스케줄링 가능성 검사의 적용 제약사항인 D = T를 D ≤ T로 완화시킨다. 이는 시스템들이 주기에 비해 데드라인이 짧은 경우나 긴 주기를 가지지만 빠른 응답시간을 요하는 경우, 이를 분석하기 위해 고안된 방법이라 할 수 있다.

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad (3)$$

$R_i$는 태스크 i의 최대응답시간, $C_i$는 태스크 i의 실행시간, $T_i$는 태스크 i의 주기, hp(i)는 태스크 i보다 높은 우선순위의 태스크 집합을 의미한다. 수식은 태스크 i보다 높은 우선순위의 태스크들이 태스크 i의 최대응답시간 동안에 여러 번 실행되는 경우를 고려하여 최대응답시간을 산출한다.

$R_i$를 계산하는 방법은 다음 점화식을 사용하여 값이 수렴하거나 $D_i$를 넘을 때까지 반복해서 계산한다. 계산 시 $R_i^0$ = 0으로 하여 계산을 시작한다.

$$R_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \quad (4)$$

#### 2.2.3 자원 공유를 고려한 스케줄링 가능성 분석

위 2.2.1과 2.2.2에서 제시한 스케줄링 가능성 분석에서는 자원 공유에 따른 블로킹 문제를 고려하고 있지 않다. 잘 알려진 우선순위 역전(Priority Inversion) 과 데드락 문제의 해결책으로 Priority Ceiling Protocol이 있다. 우선순위 역전은 두 태스크 간의 자원 공유 시 낮은 우선순위 태스크가 높은 우선순위 태스크의 실행을 지연시키는 문제이다. 데드락은 두 태스크가 서로 남이 가진 자원을 요구하면서 양쪽 모두 작업 수행을 할 수 없이 대기 상태가 되는 현상을 말한다.

Priority Ceiling Protocol은 세마포어를 락(lock)/언락(unlock)하는 방법에 대한 세 가지 제약사항이 있다. 첫째, 태스크는 호출(invocation) 간에 세마포어를 락하고 있을 수 없다. 즉, 태스크가 실행을 완료할 때 반드시 모든 세마포어를 언락해야 한다. 둘째, 태스크는 세마포어 락을 피라미드 방식으로 해야 한다. 예를들어, 두 개의 세마포어 $s_1$과 $s_2$가 있다면 다음과 같이 락한다.

lock($s_1$) ... lock($s_2$) ... unlock($s_2$) ... unlock($s_1$)

아래와 같은 방식은 안 된다.

lock($s_1$) ... lock($s_2$) ... unlock($s_1$) ... unlock($s_2$)

셋째, 태스크 i가 세마포어 s를 락하려면 태스크 i의 우선순위는 다른 태스크들에 의해 락된 모든 세마포어의 최고우선순위(ceiling)보다 더 높아야 한다. 당연히 태스크 i의 우선순위가 최고우선순위보다 낮다면 태스크 i는 블러킹된다. 최고우선순위는 세마포어를 사용하는 태스크들 중 가장 높은 우선순위를 말한다.

Priority Ceiling Protocol은 다음 특성을 가진다[4]. 첫째, 데드락을 방지한다. 둘째, 태스크 i는 더 낮은 우

선순위 태스크에 의해 기껏해야 1번만 지연된다. 지연은 당연히 크리티컬 섹션(Critical Section)만큼이다. 블러킹을 감안한 스케줄링 가능성 검사식은 다음과 같다[7].

$$R_i^{n+1} = C_i + B_i + \sum_{\forall_j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \quad (5)$$

$$B_i = \max_{\{k,s \mid k \in lp(i) \wedge s \in uses(k) \wedge ceil(s) \geq pri(i)\}} cs_{k,s}$$

$cs_{k,s}$는 세마포어 s를 사용하는 태스크 k의 크리티컬 섹션의 길이, $lp(i)$는 태스크 i보다 낮은 우선순위의 태스크 집합, $uses(k)$는 태스크 k에 의해 사용되는 세마포어의 집합, $ceil(s)$는 세마포어 s를 사용하는 태스크 중 가장 높은 우선순위, $pri(i)$는 태스크 i의 우선순위를 말한다.

### 2.3 실행시간 추정

스케줄링 가능성 검사를 수행하기 위해, 먼저 각 태스크의 실행시간을 알아내야 한다. 실행시간을 알아내는 방법에는 분석(analysis)과 측정(measurement) 등이 있다.

분석 방법을 사용한 실행시간 추정은 태스크의 실행경로에 존재하는 모든 명령어(instruction)에 소요되는 시간을 합산하는 것이다. CPU의 명령어 당 소요되는 시간을 합산해야 하므로 당연히 컴파일된 기계어(machine code)를 분석해야 한다. 컴파일된 기계어를 분석하여 태스크의 실행흐름을 분석하는 것은 상당히 어려운 작업이며, 분석한 명령어들이 CPU에서 실행될 때 소요되는 시간을 산정하는 것도 상당히 어려운 작업이다. 실제로 CPU cacheing과 pipelining, 메모리 접근 방식 등이 실행시간에 영향을 주기 때문이다. 분석 방법에 기반한 툴로 aiT, Bound-T, RapiTime, SymTa/P, Heptane 등이 있다[5].

측정 방법을 사용한 실행시간 추정은 타겟 H/W에서 실행되는 태스크의 실행시간을 측정하는 것이다. 태스크의 실행시간은 태스크 실행마다 측정된 실행시간 중 가장 긴 시간을 선택하게 된다. 따라서 측정된 실행시간이 태스크의 최대 실행시간이라고 확신할 수 없을 수도 있다. 가능하다면 태스크의 최대 실행시간을 찾기 위해, 태스크 코드를 분석하여 가장 긴 실행흐름을 가지는 초기 상태와 이벤트를 찾고 이를 모의할 필요가 있다. 실행시간 측정 시 타겟 H/W의 시스템 클락을 사용하는 경우 클락 해상도 문제로 정밀한 측정이 어려운 경우가 있다. 이 경우 시간을 측정할 코드부분을 여러 번 수행한 시간을 측정하여 반복횟수로 나누어 줌으로써 문제를 해결할 수 있다[8]. 측정 시 주의할 점은 공유 자원 접근을 위한 태스크 블로킹 시간은 실행시간에서 제외되어야 한다는 점이다.

Table 2. Worst-case execution time analysis tool

| Tool | Hardware platform |
|---|---|
| aiT | Motorola PowerPC MPC 555, 565, and 755; Motorola ColdFire MCF 5307; ARM7 TDMI, HCS12/STAR12, TMS320C33, C166/ST10, Renesas M32C/85, Infineon TriCore 1.3 |
| Bound-T | Intel-8051, ADSP-21020, ATMEL ERC32, Renesas H8/300, ATMEL AVR, ATmega, ARM7 |
| RapiTime | Motorola PowerPC family, HCS12 family, ARM, NecV850, MIPS3000 |
| SymTA/P | various ARM (RealView Suite), TriCore, i8051, C167 |
| Heptane | Pentium1, StrongARM 1110, Hitachi H8/300 |
| Vienna S.<br>Vienna M.<br>Vienna H. | M68000, M68360, C167<br>C167, PowerPC<br>HCS12, Pentium |
| SWEET | ARM9 core, NEC V850E |
| Florida | MicroSPARC I, Intel Pentium, StarCore SC100, Atmel Atmega, PISA/MIPS |
| Chalmers | PowerPC |
| Chronos | SimpleScalar out-of-order processor model with MIPS-like instruction-set-architecture (PISA) |

## 3. 유사 실시간시스템 스케줄링 가능성 분석

### 3.1 유사 실시간시스템 소개

스케줄링 가능성 검사를 수행한 실시간시스템은 1조의 무장에 대한 발사절차를 모의하는 무장모의기이

다. 무장모의기 H/W 사양과 외부인터페이스는 Table 3 및 Fig. 1과 같다. 시리얼 인터페이스의 전송속도는 1 Mbps이며 무장모의기가 송수신하는 개별 메시지의 최대 사이즈는 256 Byte이다.

스케줄링 가능성 분석 대상 태스크 정보는 Table 4와 같다. 이는 무장모의기에서 동시에 실행될 수 있는 태스크의 집합이다. 태스크는 크게 주기 태스크 6개와 비주기 태스크 2개로 구성된다.

Table 3. H/W specifications of simulator

| 구성품 | 사양 |
|---|---|
| FPGA | PowerPC440(400MHz) on VIRTEX5 FPGA |
| Memory | DDR2 SDRAM 256MB |
| Serial | RS-422 SDLC(1Mbps) |
| OS | Vxworks 6.7 |

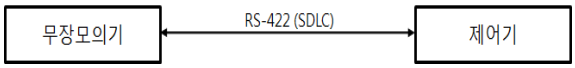


Fig. 1. External interface of simulator

주기 태스크인 tTask2, tTask7, tTask8, tTask9은 주기적으로 제어기에 메시지를 송신하는 태스크이고 tDiscrete은 개별신호와 무장모의기 전면판의 오류모의 스위치 입력상태를 감지하는 태스크이다. tPeriodic은 10 ms 주기의 타이머 인터럽트를 받아서 주기 태스크들과 tDiscrete을 실행시키는 역할을 한다. 비주기 태스크인 tSdlcRx는 제어기로부터 메시지를 수신하는 태스크이며 수신 메시지에 대한 응답은 주기 태스크(tTask2, tTask7)나 비주기 태스크(tTaskResp)로 전달한다.

비주기 태스크의 최소 주기는 제어기에서 정한 메시지 전송 최소 간격인 100 ms로 하며 데드라인은 무응답 판단기준의 1/3인 20 ms로 한다. tSdlcRx와 tTaskResp는 메시지 수신에서 응답까지 20 ms 이내에 완료되어야 하므로 모두 20 ms 보다 작은 데드라인을 가져야 하지만 계산상의 편의를 위해 20 ms로 한다.

스케줄링 가능성 검사에 필요한 태스크별 실행시간은 측정을 통해 획득하였다. 측정에 사용한 타이머는 CPU 클락을 사용하여 2.5 μsec까지 측정 가능하다. 태스크별 실행시간은 태스크 루프의 실행시간 최대값과 태스크 실행 전환에 소요되는 시간(0.052 ms ~ 0.136 ms)의 최대값을 합산한 시간이다.

Table 4. Main tasks of simulator

| 태스크명 | | 우선순위 | 주기 | 실행시간 | 데드라인 |
|---|---|---|---|---|---|
| 주기 | tPeriodic | 1 | 10 | 0.329 | 10 |
| | tDiscrete | 2 | 10 | 0.289 | 10 |
| | tTask2 | 3 | 20 | 0.298 | 20 |
| | tTask7 | 4 | 20 | 0.280 | 20 |
| | tTask8 | 5 | 100 | 0.263 | 20 |
| | tTask9 | 6 | 100 | 0.249 | 20 |
| 비주기 | tSdlcRx | 7 | 100 | 0.406 | 20 |
| | tTaskResp | 8 | 100 | 0.388 | 20 |

### 3.2 스케줄링 가능성 분석

Table 4에는 비주기 태스크가 있지만 스케줄링 가능성 분석의 편의성을 위해 비주기 태스크를 주기 태스크로 간주하고 분석을 수행한다. 또한 주기와 데드라인이 동일하지 않으므로 사용률 기반 스케줄링 가능성 분석은 수식 (2)를 사용하도록 한다.

먼저 수식 (2)로 계산하면 0.1560(15.60 %)이므로, 8개 태스크에 해당하는 0.7241(72.41 %)에 비해 여유로운 CPU 사용률을 보인다. 따라서 모든 태스크는 데드라인을 만족한다.

태스크 간의 공유자원이 없기 때문에 수식 (4)으로 각각의 태스크에 대한 응답시간을 구하면 다음과 같다. 먼저 tPeriodic 태스크는 우선순위가 가장 높기 때문에 다른 태스크에 의해 대기하는 시간이 없다. 따라서 $R_{tPeriodic} = C_{tPeriodic} = 0.329$ 이다.

다음 우선순위의 tDiscrete 태스크는 자신보다 더 높은 우선순위를 가지는 tPeriodic 태스크가 있다. 점화식을 사용하여 값이 수렴할 때까지 계산을 수행해야 하지만 tDiscrete의 실행시간이 tPeriodic 태스크의 주기에 비해 너무 작기 때문에 $\lceil R_i^n / T_j \rceil$ 은 항상 1이 된다. 따라서 $R_{tDiscrete} = C_{tDiscrete} + C_{tPeriodic} = 0.618$ 이다.

나머지 태스크들도 tDiscrete와 동일한 방법으로 계산하면 Table 5 및 Fig. 2와 같은 최대응답시간을 가진다. 각 태스크의 데드라인과 최대응답시간을 비교해 보면 모든 태스크가 데드라인을 만족함을 확인할 수 있다.

Table 5의 최대응답시간으로 알 수 있는 사실은 무

장 1조를 모의하는 무장모의기의 태스크는 우선순위에 관계없이 데드라인을 만족할 수 있다는 것이다. 가장 우선순위가 낮은 tTaskResp 태스크의 최대응답시간이 2.502 ms이므로 어떠한 우선순위를 주어도 우선순위가 가장 낮은 태스크의 최대응답시간은 2.502 ms임을 알 수 있다. 따라서 무장 1조를 모의하는 무장모의기의 태스크들은 우선순위를 동일하게 주어 FIFO 방식으로 스케줄링하여도 무방함을 알 수 있다.

Table 5. Worst-case response time of simulator

| 태스크명 | 최대응답시간 | 데드라인 |
|---|---|---|
| tPeriodic | 0.329 | 10 |
| tDiscrete | 0.618 | 10 |
| tTask2 | 0.916 | 20 |
| tTask7 | 1.196 | 20 |
| tTask8 | 1.459 | 20 |
| tTask9 | 1.708 | 20 |
| tSdlcRx | 2.114 | 20 |
| tTaskResp | 2.502 | 20 |

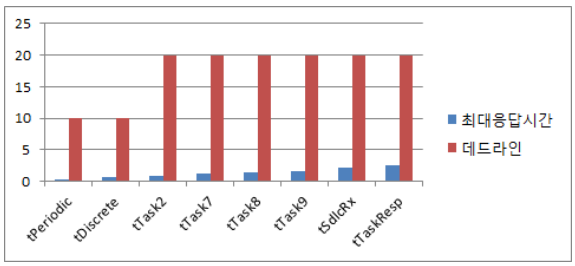


Fig. 2. Worst-case response time of simulator

## 4. 실시간시스템 확장 설계 및 분석

### 4.1 실시간시스템 확장 설계

설계할 시스템은 무장 4조를 모의하는 무장모의기로, 무장 1조를 모의하는 무장모의기와 동일한 기능을 가지며 연동통제문서(ICD: Interface Control Document) 또한 유사하다. 무장모의기 H/W는 Table 6의 사양으로 설계되며 RS-422 인터페이스는 모의하는 무장별로 각각 1개씩 Fig. 3와 같이 존재한다.

Table 6. H/W specifications of designed simulator

| 구성품 | 사양 |
|---|---|
| CPU | Intel 1rd Gen. Corei7-620LE 2.0GHz |
| Memory | DDR3-1066 8GB |
| Serial | RS-422 SDLC (1Mbps) × 4 |
| OS | Vxworks 6.9 |

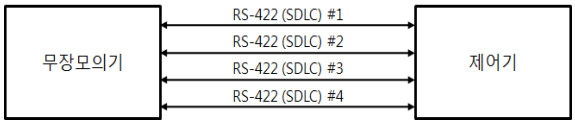


Fig. 3. External interface of designed simulator

4조의 무장을 모의하므로 Table 4에서 tPeriodic 태스크를 제외한 나머지 태스크들의 개수는 각각 4배가 된다. tPeriodic 태스크는 실제로 4조 모의를 위한 주기 태스크를 컨트롤해야 하므로 더 많은 실행시간을 가질 것이다. 하지만 기존 1조 모의용 시스템에서 수행 작업에 비해 실행시간이 과도하게 측정된 경향이 있으므로, 실행시간을 그대로 유지하고 분석하도록 한다. 설계하는 무장모의기는 연동 메시지와 모의 기능이 기존 시스템과 유사하기 때문에 주기, 실행시간 및 데드라인은 기존 시스템의 값을 유지한 상태로 스케줄링 가능성 분석을 하도록 한다. 실제로 연동 메시지 중 일부 주기 메시지는 전송 주기가 좀 더 여유롭다.

Table 7. Main tasks of designed simulator

| 태스크명 | | 우선순위 | 주기 | 실행시간 | 데드라인 |
|---|---|---|---|---|---|
| 주기 | tPeriodic | 1 | 10 | 0.329 | 10 |
| | tDiscrete_# | 2 | 10 | 0.289 | 10 |
| | tTask2_# | 3 | 20 | 0.298 | 20 |
| | tTask7_# | 4 | 20 | 0.280 | 20 |
| | tTask8_# | 5 | 100 | 0.263 | 20 |
| | tTask9_# | 6 | 100 | 0.249 | 20 |
| 비주기 | tSdlcRx_# | 7 | 100 | 0.406 | 20 |
| | tTaskResp_# | 8 | 100 | 0.388 | 20 |

### 4.2 스케줄링 가능성 분석

먼저 수식 (2)로 계산하면 0.3321(33.21 %)이므로, 29개 태스크에 해당하는 0.7015(70.15 %) 대비 절반에 못 미치는 CPU 사용률을 보인다. 따라서 모든 태스크는 데드라인을 만족한다.

수식 (4)으로 각각의 태스크에 대한 응답시간을 구하면 Table 8 및 Fig. 4와 같다. 각 태스크의 실행시간이 태스크 주기에 비해 너무 작기 때문에 $\lceil R_i^n / T_j \rceil$ 항은 항상 1이다. 각 태스크의 데드라인과 최대응답시간을 비교해보면 모든 태스크가 데드라인을 만족함을 확인할 수 있다.

Table 8. Worst-case response time of designed simulator

| 태스크명 | 최대응답시간 | 데드라인 |
|---|---|---|
| tPeriodic | 0.329 | 10 |
| tDiscrete_# | 1.485 | 10 |
| tTask2_# | 2.677 | 20 |
| tTask7_# | 3.797 | 20 |
| tTask8_# | 4.849 | 20 |
| tTask9_# | 5.845 | 20 |
| tSdlcRx_# | 7.469 | 20 |
| tTaskResp_# | 9.021 | 20 |

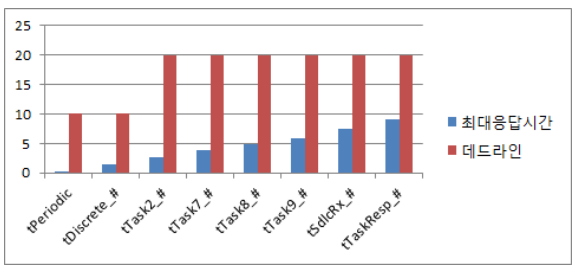


Fig. 4. Worst-case response time of designed simulator

Table 8의 최대응답시간으로 알 수 있는 사실은 무장 4조를 모의하는 태스크도 우선순위에 관계없이 데드라인을 만족할 수 있다는 것이다. 가장 우선순위가 낮은 tTaskResp_# 태스크의 최대응답시간이 9.021 ms이므로 어떠한 우선순위를 주어도 우선순위가 가장 낮은 태스크의 최대응답시간은 9.021 ms 이다. 하지만 주기가 10 ms인 tPeriodic과 tDiscrete_#는 태스크들의 실행시간이 0.979 ms 이상 늘어난다면 데드라인을 충족하지 못할 것이다. 따라서 이 두 태스크는 다른 태스크보다 높은 우선순위를 주어서 항상 데드라인 안에 실행되도록 해야 할 것이다.

## 5. 결 론

실시간시스템에서 가장 중요한 것은 시스템이 정해진 데드라인을 만족하면서 예측 가능한 범위에서 동작함을 보장하는 것이다. 이를 위해 시스템 설계 시 스케줄링 가능성 분석을 수행할 필요가 있다. 실시간시스템에 대한 스케줄링 가능성 분석을 수행하려면 각 태스크에 대한 실행시간 추정이 필요하다. 하지만 구현이 어느 정도 진행되어야 실행시간을 추정할 수 있기 때문에 설계단계에서 실시간시스템에 대한 스케줄링 가능성 분석은 사실상 불가한 경우가 많다.

본 논문은 설계단계에서 실시간시스템에 대한 스케줄링 가능성 분석을 수행하기 위한 대안을 제시한다. 제시하는 방법은 유사한 실시간 시스템을 기반으로 태스크 실행시간을 추정하여 스케줄링 가능성 분석을 수행하는 것이다. 실제로 시스템을 확장하여 개발하는 경우, 하드웨어 성능이 개선되었을 지라도, 이 방법을 사용하여 스케줄링 가능성을 분석하고 시스템의 데드라인 만족 여부를 확인하는 것은 시스템 구현 전에 반드시 수행되어야 할 것이다.

그 사례로, 본 논문에서는 무장 1조를 모의하는 무장모의기를 기반으로 무장 4조를 모의하는 무장모의기에 대한 스케줄링 가능성 분석을 수행하였다. 무장 1조를 모의하는 무장모의기의 경우 사용률 기반 스케줄링 가능성 분석 시 15.60 %의 사용률을 보였으며, 최대응답시간 기반 스케줄링 가능성 분석 시 각 태스크의 최대응답시간 또한 데드라인 대비 최대 12.51 % 수준의 빠른 응답시간을 보였다. 무장 4조를 모의하는 무장모의기의 경우 사용률 기반 스케줄링 가능성 분석 시 33.21 %의 사용률을 보였으며, 최대응답시간 기반 스케줄링 가능성 분석 시 각 태스크의 최대응답시간은 데드라인 대비 최대 45.11 % 수준의 응답시간을 보였다.

스케줄링 가능성 분석을 통해 무장 4조를 모의하는 무장모의기의 경우 주기 태스크가 비주기 태스크에 비

해 높은 우선순위를 가진다면 모든 태스크는 데드라인 안에 처리를 수행할 수 있다는 것이다. 설령 각 태스크가 수행하는 작업이나 디버깅을 위한 구문이 추가되더라도 각 태스크가 데드라인을 만족하지 못할 수준이 될 가능성은 낮다. 데드라인을 만족하지 못하려면 태스크들의 실행시간 총 합이 10 ms 이상 늘어나야 하기 때문이다.

## References

[1] C. L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment,” JACM, Vol. 20. No. 1, pp. 46-61, 1973.

[2] Neil C. Audsley, “Deadline-Monotonic Scheduling,” University of York, Department of Computer Science, 1990.

[3] M. Joseph and P. Pandya, “Finding Response Times in Real-Time Systems,” The Computer Journal, Vol. 29, No. 5, 1986.

[4] L. Sha, R. Rajkumar and J. P. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization,” IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185, 1990.

[5] R. Wilhelm et al., “The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools,” ACM Transactions on Embedded Computing Systems, Vol. 7, No. 36, 2008.

[6] Buttazzo, Giorgio C. “Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications,” Vol. 24. Springer Science & Business Media, pp. 77-107, 2011.

[7] Buttazzo, Giorgio C. “Hard real-time computing systems: predictable scheduling algorithms and applications,” Vol. 24. Springer Science & Business Media, pp. 181-221, 2011.

[8] Philip A. Laplante, “Real-Time Systems Design and Analysis,” IEEE Press, New Jersey, pp. 357-363, 2004.