소형 다관절로봇을 위한 운용 소프트웨어 구현

Implementation of Operating Software for Small Multi-Jointed Robots

손 현 승, 김 우 열, 김 영 철*
(Hyun-Seung Son, Woo-Yeol Kim, and Robert Young-Chul Kim)

Abstract: The small multi-jointed robots for most education are developed with the way of firmware. But the firmware may be very difficult to develop as gradually increasing throughputs and control routines. Due to limit of firmware we try to use on RTOS, but hard to adapt on the small multi-jointed robots. It would be hard to install RTOS into the small multi-jointed robots because of the size capacity of RTOS, and lack of libraries for control of the particular hardware. Moreover, even its RTOS with many functions causes its to make overheads scheduling, TCB (Task Control Block), and task states. Also to keep maintenance of RTOS, it is composed of components for the whole structure of my proposed RTOS. Additionally, We provided with libraries of servo motor and sensor control and developed RMS (Rate Montonic scheduler) to handle tasks on real time and reduce overheads. Therefore, It is possible to work the fixed priority and task preemptive way. We show one example of the multi-jointed robot installed with my proposed RTOS, which shows to obstacle avoidance and climbing up the slope.

Keywords: multi-joint robots, RMS (Rate Monotonic Scheduler), RTOS (Real Time Operating System), operating software, embedded system

I. 서론

기존의 소형 다관절로봇은 펌웨어로 개발되었다[1]. 그러나 펌웨어는 작업량이 많아지면 소프트웨어의 복잡도가 높아져 개발이 어려워진다. 이런 펌웨어의 한계로 RTOS (Real Time Operating System)를 적용한다[2]. RTOS를 적용하면 다음과 같은 장점이 있다. 첫 번째로 하드웨어에 종속되지 않고 프로그램 호환성을 높일 수 있다. 그러면 하드웨어 선정이나 교체에 따른 부담을 줄일 수 있다. 두 번째로 태스크 단위별로 프로그램을 제작하므로 재 사용성이 높아진다. 재사용성이 높아지면 개발시간이 단축되고 검증된 코드를 사용으로 시험기간이 줄어든다. 세 번째로 다중의 프로세서에도 프로세서간 통신이 통일성을 갖고 간편하게 통신 구현이 가능하다. 마지막으로 프로그램의 오류를 사전에 예방, 정정해주어 시스템이 다운되는 것을 방지하여 안정성을 높인다. 이러한 장점 때문에 일반적인 로봇에는 RTOS를 사용한다[3].

하지만 기존의 RTOS는 소형 다관절로봇 적용시 해결해야하는 몇 가지 문제가 있다. 첫 번째로 RTOS의 용량 문제를 살펴보면, 대부분의 RTOS는 표 1과 같은 크기의 프로그램 메모리를 사용한다. 이것은 응용프로그램을 포함하지 않은 RTOS의 자체의 크기이다. 여기에 환경에 맞는 응용프로그램을 작성하면 크기가 더 커진다.

그러나 소형 기기제어를 위한 마이크로컨트롤러의 프로그램 메모리는 1Kbyte~4Kbyte 크기를 가진다[4]. 그러므로 기존의 RTOS는 소형 마이크로컨트롤러에 탑재하기 어렵다.

* 책임저자(Corresponding Author)
논문접수: 2009. 5. 20., 수정: 2009. 6. 5., 채택확정: 2009. 6. 22.
손현승, 김우열: 홍익대학교 전자전산공학과
(son@selab.hongik.ac.kr/john@hongik.ac.kr)
김영철: 홍익대학교 컴퓨터정보통신공학과(bob@hongik.ac.kr)
※ 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터육성지원사업의 연구결과로 수행되었음 (C1090-0903-0004).

표 1. RTOS의 프로그램 크기 비교.

Table 1. Comparing RTOS based program size.

OS	Nucleus Plus	ThreadX	Integrity	LynxOS	Windows CE	QNX Neutrino	VxWorks
크기	4KB 이상	10KB 이상	10KB 이상	250KB 이상	300KB 이상	250KB 이상	36KB 이상

두 번째로는 다수의 모터제어 문제이다. 모터 하나는 주기적으로 펄스 신호가 공급 되어야 동작 하는데, 모터의 수가 많을 경우 태스크의 문맥전환과 시스템 지연으로 기다림 상태가 된다. 기다림 상태가 되면 모터에 펄스가 공급되지 않아 떨림 현상이나 오동작을 한다. 그렇기 때문에 최적화된 스케줄러가 필요하다.

본 논문에서는 기존의 RTOS를 소형 다관절로봇인 6족 로봇에 적용하기 위해 경량화하고, 다수모터제어를 지원해 줄 수 있는 운용 소프트웨어 개발을 목적으로 한다. 그리고 개발된 운용 소프트웨어가 실시간성을 지원하기 위해 RMS (Rate Monotonic Scheduler)[5]를 적용하였다.

본 논문의 구성은 다음과 같다. 제 II 장에서는 관련연구로서 기존 임베디드 시스템 개발 방법과 RTOS 스케줄러에 대하여 설명한다. 제 III 장은 소형 다관절로봇용 운용 소프트웨어를 개발하기 위한 경량화, RMS 적용에 대해 언급한다. 제 IV 장은 사례연구로서 6족 로봇에 맞게 개발된 운용 소프트웨어를 탑재한 다음, 응용프로그램의 개발과 수행 결과를 확인한다. 마지막장에서는 결론 및 향후 연구 과제를 언급한다.

II. 관련연구

1. 기존 임베디드 시스템 개발 방법

기존의 다관절로봇 개발은 크게 펌웨어를 이용한 방법, Non-RTOS 이용한 방법, RTOS를 이용한 방법이 있다. 펌웨어 기반 개발 방법은 병렬성을 지원 하지 않고, 적은 규모

에서 사용되며, 하드웨어 의존도가 높아 제한된 로봇에만 적용되는 단점이 있다. 또한 복잡성이 높은 반면에 수행속도는 빠르다. 그리고 non-RTOS기반 방법과 RTOS 기반 방법은 운영체제가 요구되며, 병렬성이 지원되고, 쓰레드 단위별로 작업 분할이 가능하여 중대 규모의 소프트웨어의 개발이 가능하다. 하드웨어 의존도가 낮아 여러 시스템에 탑재 가능하며 복잡성이 낮은 장점이 있지만 펌웨어보다 느린 단점이 있다.

Non-RTOS와 RTOS의 가장 큰 차이점은 실시간성 지원 여부이다. 실시간성은 수행되는 태스크의 마감시간을 꼭 지켜야 되는 시스템을 말한다. 대부분의 임베디드 시스템에서 마감시간은 꼭 지켜야 한다. 하드웨어 동작은 정해진 시간에 펄스를 입력해야 작동되기 때문이다. 시스템에서 마감시간을 준수하지 못하면 하드웨어 장비는 오동작을 일으킨다.

실시간에는 경성, 연성의 두 가지 형태가 있다[6]. 경성 실시간은 실시간 시스템 중에는 어떤 작업을 일정 시간 안에 반드시 처리하고, 그 시간이 지난 후의 결과 값은 정확해도 소용이 없다. 예로는 군사장비, 비행기, 의료장비 등이 속한다. 즉, 오류로 인해 인명 피해가 발생되는 시스템을 포함한다. 연성 실시간 시스템은 어떤 시간 안에 처리하면 좋지만 그렇지 못한 경우에 그 간에서 약간 경과한 후의 값도 인정하는 경우이다. 예로는 전화기, 라우터, 동영상 플레이어, MP3 등이 있다. 즉, 오류로 인명피해는 발생되지 않지만 소프트웨어의 품질이 낮아지는 시스템을 포함한다.

2. 기존 RTOS 스케줄러

대표적인 실시간 스케줄러에는 RMS (Rate Monotonic Scheduling)[5], DMS (Deadline Monotonic Scheduling) [7], EDS (Earliest Deadline Scheduling)[8,9]가 있다.

RMS는 정적 우선순위 알고리즘으로 태스크 집합에 대해, 우선순위를 주기에 반비례하도록 지정한다. 모든 태스크가 주기적이고 데드라인이 주기의 끝에 있다고 가정한다.

DMS는 태스크 집합에 대해 태스크의 우선순위가 마감시간의 크기에 반비례이다. 또한 주기(period)와 마감시간(deadline)이 다를 수 있다. RMS는 DMS의 특수 케이스로 볼 수 있다. 주기와 상대적인 마감시간으로 태스크 우선순위를 고려하기 때문이다.

EDS는 태스크 집합에 대해 태스크의 우선순위가 절대적인 마감시간의 임박성에 비례 한다. 동적 우선순위 알고리즘으로 태스크의 절대적인 마감시간에 따라 매번 우선순위가 변할 수 있다. 동적 우선순위 알고리즘은 정적 우선순위 알고리즘을 포함하므로 EDS는 모든 스케줄링 알고리즘 중에서도 최적이다.

알고리즘의 완성도로 살펴보면 EDS > DMS > RMS 순이다. EDS는 알고리즘으로 우수하지만, RMS에 비해 복잡성이 높고 예측이 어려워서 실질적으로 RMS가 많이 쓰인다.

III. 제안한 운용 소프트웨어

제안한 운용 소프트웨어는 소형 다관절로봇을 위해 소프트웨어 경량화, RMS 적용을 제안한다.

1. 소프트웨어 경량화

기존의 RTOS는 태스크, 타이머, 메모리 관리뿐만 아니라

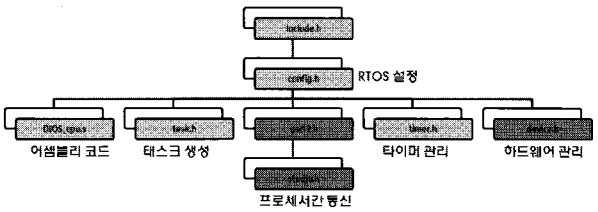


그림 1. 제안한 운용 소프트웨어의 구조.

Fig. 1. The structure of proposed operating software.

태스크간 통신을 위한 메일박스, 플래그, 세마포, 큐잉 등의 많은 기능을 제공한다. 하지만 소형의 다관절로봇을 위해서는 이러한 기능들이 모두 필요하지 않다. 때문에 불필요한 태스크간 통신기능인 메일박스, 플래그, 세마포, 큐잉 기능을 제거하여 초경량화 한다. 그 결과 제안하는 운용 소프트웨어의 구조는 그림 1과 같이 운영체제를 위한 필수 기능인 태스크관리, 문맥교환, 타이머를 제공하고 소형 다관절로봇을 위해 프로세서 통신, 하드웨어 관리를 추가하여 설계하였다.

각 구성요소를 살펴보면, "include.h"는 다른 라이브러리를 포함하는 파일이다. "config.h"는 운용 소프트웨어에 필요한 매크로와 구조체 공유 변수들을 설정하는 파일이다. "DIOS_cpu.s"는 타이머 인터럽트, 문맥교환, 운영체제 시작을 위한 Idle태스크 들이 어셈블리언어로 분할되어 있다. "task.h"는 TCB 풀리스트 생성, TCB 초기화, Idle 태스크 생성, 큐 처리, 태스크 생성을 처리한다. "uart2.h"는 UART2를 사용하여 타겟 시스템의 결과를 PC상에 확인 가능하도록 한다. "timer.h"는 시간처리를 위한 것으로 타임틱 생성과 태스크 지연 처리를 한다. "device.h"는 하드웨어 제어를 위한 영역이다.

제안한 운용 소프트웨어는 일반적인 운영체제가 가진 상태를 경량화 하여 그림 2와같이 TASK_KILL, TASK_WAIT, TASK_READY, TASK_RUN의 4가지 상태만 사용한다. TASK_KILL은 태스크가 수행 되기전 초기화된 상태이다. 제안한 운용 소프트웨어는 태스크를 삭제하는 기능이 없으므로 TASK_KILL 상태는 초기화만 되어있는 모든 태스크를 말한다. TASK_READY는 태스크를 바로 수행할 수 있도록 준비된 상태이다. TASK_RUN은 현재 실행하고 있는 태스크를 말한다. TASK_WAIT는 태스크가 잠시 대기하는 상태이다.

2. RMS 적용

RMS (Rate Monotonic Scheduler)는 개별 태스크의 주기의 역수인 빈도율(rate) 이 높을수록, 즉 주기가 짧을수록 더 높은 우선순위를 부여하는 방식이다. 따라서 각 태스크의 주기가 주어지면 태스크들 간의 우선순위는 정적으로 결정될 수 있으며 이 우선순위는 시스템이 수행되는 동안 고정된다. 결국 RMS에서는 이용률로 우선순위를 결정한다[5,10].

그래서 태스크를 모두 생성한 후 각 태스크의 이용률 계산하여 우선순위를 자동으로 할당 한다. 우선순위를 할당하는 방법은 그림 3과 같다.

우선순위 할당 순서는 먼저 TCB 리스트에서 C(태스크

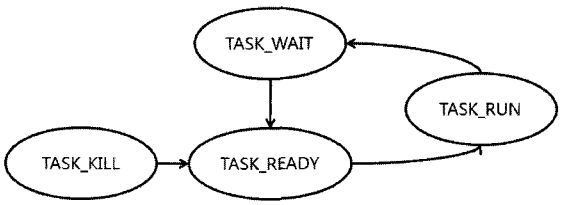


그림 2. 제안한 운용 소프트웨어의 상태다이어그램.
Fig. 2. State diagram of proposed operating software.

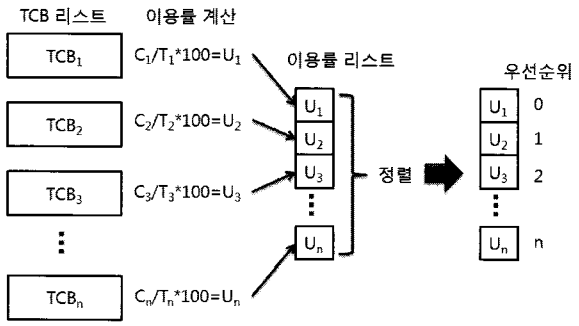


그림 3. 우선순위 할당 방법.
Fig. 3. Allocation method of priority.

수행시간), T(태스크 주기)를 이용하여 이용률을 계산한다. 그다음 이것을 리스트에 삽입한 후 이용률을 정렬하여 우선순위를 매긴다. 이때 정렬된 이용률 리스트를 순서대로 카운트하면 겹치지 않는 우선순위를 만들 수 있다.

RMS의 경우 식 (1)과 같은 부등식이 성립해야 모든 경성 데드라인들을 만족시킬 수 있다[5,10]. 그래서 부등식을 만족할 때만 실행하게 해야 한다.

$$\sum_{N=1}^{N} \frac{C_N}{T_N} \leq N(2^{1/N} - 1) \qquad (1)$$

(C=연산시간, T=주기, N=태스크 개수)

그림 4와 같이 태스크 개수와 주기정보를 이용하여 태스크 생성할 수 있는 지 검사한다. 만약 수행할 수 있으면 true, 수행할 수 없으면 false을 발생시켜 경성 실시간을 지키도록 한다.

```
INT8U CheckDIOS() {
    DIOS_TCB     *pTCB;
          pTCB = g_pTCBPoolHeder;
    INT16U sum = 0;
    INT8U taskCount = 0;
          while(pTCB != NULL) {
      sum += pTCB->nTCBUse/pTCB->nTCBCycle;
                    pTCB = pTCB->pTCBNext;
          taskCount++;
          }
    if(taskCount * (pow(2, 1/taskCount) - 1) - sum >= 0)
        return TRUE;
    return FALSE;
}
```

그림 4. 태스크 개수와 주기검사.
Fig. 4. Number of task and periodic inspection.

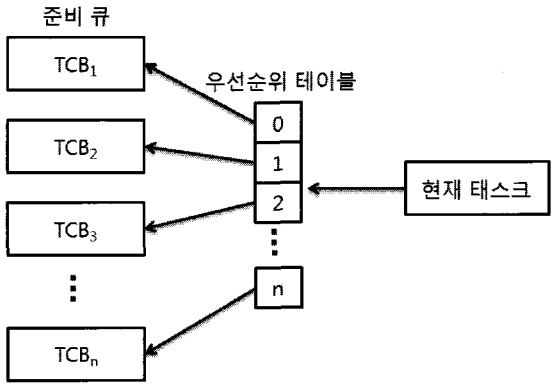


그림 5. 최상위 우선순위를 찾는 방법.
Fig. 5. How to find a top priority.

```
void        DIOS_Scheduler() {
    //준비큐에서 최상위 우선순위를 가져온다
  g_pTCBHighRdy = FindHighRdyQue();
  g_nHighPrio = g_pTCBHighRdy->nTCBPrio;
//준비큐에 아무것도 없으면 Idle실행
  if(NULL == g_pTCBHighRdy) {
    g_pTCBHighRdy = g_pTCBIdle;
    g_nHighPrio = g_pTCBIdle->nTCBPrio;
  }
  if(g_pTCBHighRdy != g_pTCBCur) {//RMS schedul
    if(g_nCurPrio < g_nHighPrio)
    {//우선순위가 높으면  컨텍스트 스위칭
      ContextSwitch();
            //Runing 하고있는것은 Ready 큐로 삽입
      g_pTCBCur->nTCBStat = TASK_RUN;
    }
  }
}
```

그림 6. 소형 다관절로봇을 위한 스케줄러.
Fig. 6. Scheduler for small multi-jointed robots.

운영체제에서 스케줄러는 수행될 태스크를 선별하는 작업이다. 스케줄러는 태스크들을 감시하며 현재 태스크 보다 높은 우선순위가 발생되었을 때 문맥교환을 수행해 다른 태스크를 수행할 수 있게 한다. 이러한 방법을 이용해서 서로 다른 여러 개의 태스크를 동시 수행이 가능하다.

태스크의 수행은 항상 준비큐에 있는 태스크 컨트롤 블록을 가지고 수행한다. 틱 처리기에서 이미 사용될 태스크를 준비 큐에 넣어주면, 스케줄러는 준비 큐에 있는 가장 높은 우선순위를 수행한다. 여기서 중요한 것은 준비 큐로부터 높은 우선순위를 빠르게 찾아서 스케줄링을 해야 한다. 그래서 제안한 운용 소프트웨어는 태스크 생성시 만든 우선순위를 테이블로 유지하고 준비 큐에 있는 태스크를 할당하게 한다. 현재 태스크의 우선순위를 중심으로 높은 우선순위를 찾아야 되기 때문에 테이블에서 위쪽 방향만 검사하면 높은 우선순위를 찾을 수 있게 한다. 그림 5는 최상위 우선순위를 찾기 위한 방법을 도식화한 것이다.

제안한 운용 소프트웨어는 그림 6과 같이 스케줄링 한다. 현재 우선순위보다 높은 태스크 컨트롤 블록을 준비 큐에서 가져와 스케줄링을 수행한다.

IV. 적용사례

본장에서는 제안한 운용 소프트웨어로 6족 로봇의 동작을 확인하여 운용소프트웨어의 수행가능성을 확인한다.

1. 하드웨어 사양

6족형 로봇은 그림 7과 같다. 본 예제의 6족 로봇은 한 다리 당 3개의 관절을 사용하여 총 18개의 모터를 사용하는 구조로 되어 있다.

6족 다관절로봇은 모터제어부에 Atmega128를 센서 제어부에 Atmega8535를 사용한 2개의 마이크로컨트롤러를 가지고 있다. 그리고 센서 모듈과 바디 모듈은 UART를 통해서 프로세서간 통신으로 작동 된다. 센서 모듈은 Atmega8535를 사용하며 4개의 초음파 센서와 2개의 자이로 센서로 구성된다. PC0~3은 초음파센서를 제어하기 위한 포트 이고 입력 출력을 사용한다. 그리고 PA0, PA1은 자이로 센서를 제어하기 위한 포트이다. 바디 모듈은 Atmega128을 사용하며 18개의 모터로 구성된다. PA0~7, PB0~7, PC3, PC7번 포트를 이용하여 18개의 모터를 제어한다. 그리고 UART0을 통해서 센서 모듈의 센서 데이터 값을 받아 처리한다.

2. 경량화된 운용 소프트웨어 테스트

운용 소프트웨어의 작동을 위해 태스크 생성 후 수행 결과를 확인한다. 운용 소프트웨어의 개발은 Atmel에서 제공하는 통합 개발 도구인 AVR studio 4를 사용하고, 컴파일러는 AVR-GCC를 사용 하였다.

운용 소프트웨어를 테스트하기 위해 첫 번째 단계에서는 테스트를 위한 테스트코드를 작성한다. 테스트코드는 운용 소프트웨어의 태스크 생성, 컨텍스트 스위칭, 인터럽트 발생을 테스트할 수 있도록 작성 하였다. 두 번째 단계는 컴파일러로 생성한 hex파일을 PonyProg2000을 이용하여 6족 로봇에 다운로딩 하는 것이다.

PC에서 LPT 케이블을 사용하여 6족 로봇에 다운로딩을 한다. 세 번째와 네 번째 단계는 프로그램 실행과 결과 확인이다. 로봇의 전원을 키면 장착된 블루투스를 통해 무선으로 PC의 COM1으로 데이터가 전송된다. 그러면 PC에서 하이퍼 터미널을 COM1으로 19200 bit/s, 데이터 비트 8bit, 패리티 없음, 정지 비트 1, 흐름제어 없음으로 세팅하여 데이터를 수신 받는다. 그리고 작성한 응용프로그램과 수행된 결과 값이 같은지 확인한다.

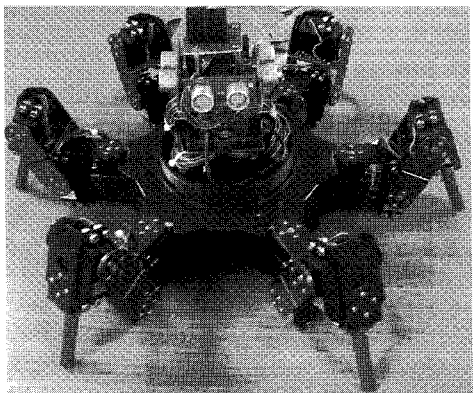


그림 7. 6족형 소형 다관절로봇.

Fig. 7. Small multi-jointed robots of six leg.

2.1 테스트코드 작성

운용 소프트웨어의 테스트를 위해서 다른 주기를 갖는 태스크 1, 2, 3을 생성하여 각 각 "task1", "task2", "task3"를 출력 하도록 하였다.

그림 8은 작성한 테스트코드이다. 코드를 보면 간단하게 작성되었지만 출력 결과로 컨텍스트 스위칭, 인터럽트 루틴, 태스크 생성이 정상으로 작동되는지 확인 가능하다. 그리고 PutString함수는 사용자가 입력한 텍스트를 PC에 전송한다. PutString함수를 통해 PC와 로봇간의 통신이 수행되는지도 확인 가능하다.

2.2 타겟 시스템에 다운로딩

텍스트 형태로 작성한 코드는 컴파일해서 hex파일로 생성한다. 컴파일러로 생성한 hex파일은 PonyProg2000을 사용하여 타겟 시스템으로 다운로딩이 가능하다.

PonyProg2000을 사용하기 위해서는 interface setup으로 연결된 케이블에 맞는 설정을 해야 한다. 6족 로봇은 다운로딩하기 위해 보통의 시리얼 케이블을 사용하는 것이 아니라 LPT 케이블을 사용한다. 그래서 I/O port는 parallel, Avr ISP I/O, LPT1으로 설정하여 사용한다. 그다음 program option을 reload files, erase, write program memory를 체크한다. 그러면 Program버튼 한번 클릭으로 설정한 파일을 다시 연결할 필요 없이 자동으로 로딩해주고 기존의 프로그램을 지운다음 프로그램 메모리를 플래시 메모리에 써주게 된다. 플래시 메모리에 쓰기 작업을 완료하면 다운로드한 프로그램과 원본을 비교하는 검증 작업 수행 후 완료한다.

2.3 수행 및 결과 확인

그림 9는 6족 로봇에 운용 소프트웨어를 탑재하여 수행한 결과이다. 운영소프트웨어의 테스트를 위해 TASK를 3개 생성하였다. TASK1은 200㎳마다 "task1"을 출력, TASK2는 500㎳마다 "task2"를 출력, TASK3은 1000㎳마다 "task3"을 출력하게 하였다. 출력결과를 확인 해보면, 그림 12의 ①과 같이 태스크가 2번 수행되어 400㎳를 소비하고 task2가 수행된다. ②는 태스크 사이에 task3, task1, task1, task1이 수행되어 약 500㎳가 수행한 다음 task2가 반복되는 것을 확인 할 수 있다. ③은 task1의 수행은 5번, task2의 수행

```
void task1(void *data) {
  while(1) {
    PutString("task1\r\n");
      mdelay(200);    }
}
void task2(void *data) {
  while(1) {
    PutString("task2\r\n");
      mdelay(500);    }
}
void task3(void *data) {
  while(1) {
    PutString("task3\r\n");
      mdelay(1000);    }
}
```

그림 8. 테스트코드.

Fig. 8. Test code.

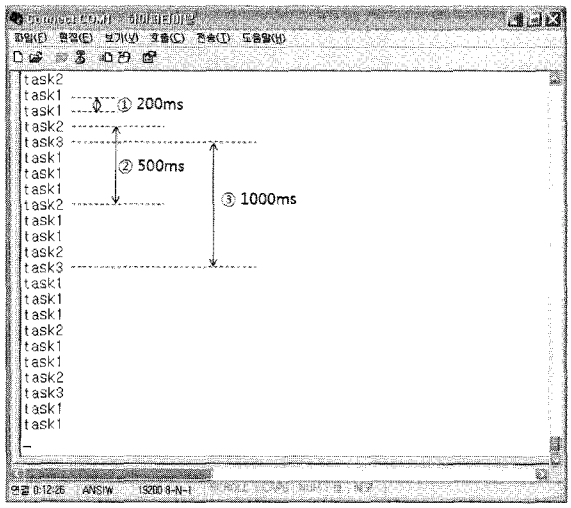


그림 9. 운용 소프트웨어 수행 결과.

Fig. 9. Execution result of operating software.

표 2. RTOS의 비교표.

Table 2. Comparing RTOS.

OS	Nucleus Plus	ThreadX	VxWorks	제안한 RTOS
크기	4KB이상	10KB이상	36KB이상	1KB
로열티	X	X	O	X
특징	그래픽 프로토 타이핑, 커널, 미들웨어 및 이클립스 기반의 개발툴로 구성	소형이면서도 신속한 실시간 응답을 요구하며, 대량 생산을 타겟으로 함	MMU지원, IPC 메시징 인터페이스, UNIX와 유사한 프로세스모델	경량화, 오버헤드 최소화, 모터 제어 라이브러리 지원, 센서 제어 라이브러리 지원

은 2번으로 1000㎳마다 수행하는 것을 확인 가능하다. 이 결과를 통해 운용 소프트웨어가 태스크 생성, 컨텍스트 스위칭, 인터럽트 발생 및 처리를 할 수 있음을 확인하였다.

3. 결과 비교

표 2는 설명한 기존의 RTOS의 특징들과 제안한 운용 소프트웨어를 비교한 것이다. 기존의 RTOS는 상업용 제품으로 오랜 경험적인 기술을 바탕으로 신뢰성 있는 제품을 제공하여주고 다양한 네트워크, 다양한 파일시스템, 다양한 프로세서, 개발도구까지 제공한다. 하지만 사용을 위해서는 비용을 지불해야 하고 용량을 많이 차지한다. 또한 내부 코드를 공개하지 않고 바이너리 형태로 제공되기 때문에 응용 및 확장이 어려운 문제가 있다[11,12]. 결과적으로, 제안한 운용 소프트웨어는 기존에 비해 적은기능이지만 소형 다관절로봇을 개발을 위한 적합한 크기와 라이브러리를 제공한다. 그리고 소스코드 형태로 제공되어 다양한 응용 및 실험이 가능하다. 하지만 상용 RTOS에 비해 다양한 프로세서, 네트워크 지원, 파일시스템 등과 같은 다양한 기능이 제공 되지 않는 단점이 있으며, 적용분야가 6족 로봇에 한정된다.

V. 결론

기존의 소형 다관절로봇은 펌웨어를 사용하여 시스템을 개발해 왔다. 하지만 작업량과 하드웨어 제어루틴이 많아지면서 소프트웨어의 복잡성 증가로 펌웨어의 개발이 어려워진다. 또한 펌웨어는 동시처리, 인터럽트 지연 문제, 타이밍 제어 등의 복잡 설계의 어려움이 있다. 이런 문제로 RTOS를 적용하려 하지만 소형 다관절로봇에 적용하기 위해서는 크기, 오버헤드, 다수의 모터 제어의 문제를 해결이 필요하다.

개발한 운용 소프트웨어는 기존 RTOS의 문제를 해결하기 위해 경량화 하였고 실시간성 지원을 위해 RMS를 적용하였다. RTOS의 오버헤드 최소화와 크기를 위해 태스크 컨트롤 블록, 태스크 상태 등을 초경량화 하였고 다수모터를 제어하기위해 RMS를 적용하였다.

그 결과 6족 로봇에 RTOS의 기능과 개선을 거친 운용 소프트웨어를 적용 할 수 있었다. 또한 운용 소프트웨어로 다수 모터제어 문제를 쉽게 해결하였다.

향후연구과제로 다중 프로세서 지원과 프로세서간의 상호운영을 위한 운영체제를 개발 중이다. 또한 모듈화 기능 지원하고 더 많은 하드웨어 기기를 지원하도록 운영체제의 확장이 필요하다. RTOS의 신뢰성을 향상시키기 위한 방법 연구도 필요하다.

참고문헌

[1] 엄우용, 이종호, "로봇 플랫폼을 위한 이족보행로봇 개발," 대한전자공학회, 제43권 제4호, pp. 136-143, 2006.

[2] 김용규, 이명호, "자동차 조향 장치의 동향과 전망 -기계 중심에서 Software 중심으로-," 정보처리학회지, 제15권 제5호, pp. 40-47, 2008.

[3] 김동진, 유승환, 신윤덕, 장승익, 기창두, "엔터테인먼트용 조류형 2족 보행 로봇의 설계 및 구현," 한국정밀공학회지, 제22권 제3호, pp. 38-45, 2005.

[4] 박권서, "원칩 마이크로컴퓨터 8535," 월간 전자기술, pp. 120-128, 2001.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no.1, pp. 46-61, 1973.

[6] D. E. Simon, *An Embedded Software Primer*, Addison-Wesley, 1999.

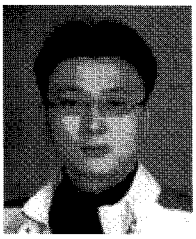
[7] P. Altenbernd, "Deadline-monotonic software scheduling for the CO-synthesis of parallel hard real-time systems," *IEEE Computer Society*, pp. 190-195, 1995.

[8] H. Chetto and M. Chetto, "Some results of the earliest deadline scheduling algorithm," *IEEE Press*, vol. 15, Issue 10, pp. 1261-1269, 1989.

[9] M. Kargahi and A. Movaghar, "A method for performance analysis of earliest-deadline-first scheduling policy," *The Journal of Supercomputing*, vol. 37, Issue 2, pp. 197-222, 2006.

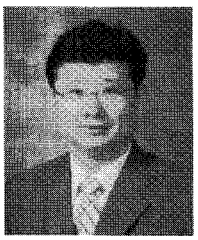
[10] 임경현, 서재현, 박경우, "실시간 시스템에서 태스크

이용율을 이용한 스케줄링 가능성 검사," 인터넷정보학회논문지 제6권 제2호, pp. 25-35, 2005.
[11] 손현승, 김우열, 김영철, "다관절 로봇의 효율적인 동작제어를 위한 기술 구현," 한국정보처리학회, 제15권 제2호, pp. 593-596, 2008.
[12] 손현승, 김우열, 김영철, "소형 다관절로봇 RTOS 구현을 위한 디자인 패턴 적용," 한국인터넷방송통신TV학회, 제7권 제1호, pp. 110-113, 2009.



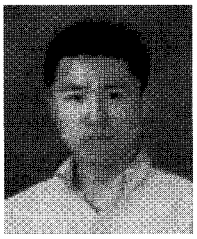
손 현 승

2007년 홍익대학교 컴퓨터정보통신 학사. 2009년 홍익대학교 일반대학원 소프트웨어공학전공 석사. 2009년~현재 홍익대학교 일반대학원 박사과정. 관심분야는 임베디드 소프트웨어 자동화 도구 개발, 임베디드 RTOS 개발, 임베디드 MDA (Model Driven Architecture) 연구, 모델 검증 기법 연구.



김 영 철

2000년 Illinois Institute of Technology 공학박사. 2000년~2001년 LG 산전 중앙연구소 Embedded system 부장. 2001년~현재 홍익대학교 컴퓨터정보통신 부교수. 관심분야는 테스트 성숙도 모델, 임베디드 S/W 개발 방법론 및 도구 개발, 모델 기반 테스팅, CBD, BPM, 사용자 행위 분석 방법론.



김 우 열

2004년 홍익대학교 컴퓨터정보통신 학사. 2006년 홍익대학교 일반대학원 소프트웨어공학전공 석사. 2006년~현재 홍익대학교 일반대학원 박사과정. 관심분야는 상호운용성, 임베디드 소프트웨어 개발 방법론 및 도구 개발, 컴포넌트 시험 및 평가, 리팩토링.