로봇 컴포넌트에 실시간성을 지원하기 위한 프레임워크 구현 및 성능분석

Implementation and Performance analysis of a Framework to Support Real-Time of Robot Components

최찬우, 조문행, 박성종, 이철훈
충남대학교 컴퓨터공학과

Chan-Woo Choi(cwchoi00@cnu.ac.kr), Moon-Haeng Cho(root4567@cnu.ac.kr),
Seong-Jong Park(prime@cnu.ac.kr), Cheol-Hoon Lee(clee@cnu.ac.kr)

요약

유비쿼터스 환경에서 지능형 서비스 로봇의 실시간성 기술은 QoS를 보장하기 위해서 필수 불가결한 요소이다. 본 논문에서는 지능형 서비스 로봇에 실시간성을 지원하는 실시간 프레임워크를 설계 및 구현한 내용을 기술한다. 실시간 스케줄링 서비스를 제공하는 실시간 프레임워크는 범용 운영체제를 기반으로 동작하며, 범용 운영체제에서 제공하는 스케줄러의 실시간성 미 지원 문제를 해결한다. 본 논문에서는 또한 실시간 로봇 애플리케이션에 QoS를 보장하기 위한 실시간 스케줄링 서비스를 제안한다. 제안된 실시간 프레임워크의 성능 평가를 위해 윈도우즈 운영체제 상에 구현하였다. 실험 결과를 통해 쓰레드의 응답 시간 향상과 실시간 프레임워크 탑재에 따른 성능상의 오버헤드가 62μs로 미미하다는 것을 알 수 있다.

■ 중심어 : | 실시간성 | 실시간 프레임워크 | 로봇 컴포넌트 |

Abstract

In ubiquitous environments, the real-time features are necessary to insure the QoS of the intelligent service robots. In this paper, we design and implement a real-time framework for intelligent service robots to support real-time features. The real-time framework to support real-time scheduling services is implemented on the general operating systems. We solve the problem that the scheduler of a general operating system can not support real-time features. This paper also proposes realtime scheduling services to guarantee the QoS of real-time robot applications. We implemented the proposed real-time framework on the Windows operating system and conducted some performance experiments. The experimental results show that the proposed real-time framework can improve thread response times and it has slight performance overhead of 62μs.

■ keyword : | Real-time | Real-time Framework | Robot Components |

* 본 연구는 지식경제부 및 정보통신연구진흥원의 IT성장동력기술개발사업의 일환으로 수행하였음. [2008-S-030-01, RUPI-클라이언트 기술 개발]

접수번호 : #081028-007
접수일자 : 2008년 10월 28일
심사완료일 : 2009년 04월 06일
교신저자 : 이철훈, e-mail : clee@cnu.ac.kr

I. 서 론

빠르게 IT 기술이 발전하면서 산업 현장에서만 사용되던 로봇들이 인간들과 공존하면서 일상생활의 곳곳에서 사용되고 있다. 병원에서 간호사를 보조하는 간호 보조 로봇, 가정집에서 청소를 담당하는 청소 로봇등과 같은 다양한 로봇들이 연구 개발되고 있다[1]. 이러한 로봇에 대한 연구는 점차적으로 증가하고 있으며, 현재는 지능형 서비스 로봇에 많은 관심과 투자가 이루어지고 있다. 지능형 서비스 로봇을 위한 미들웨어로 OROCOS[2-4], RSCA[5], MSRS[6][7], MIRO[8][9] 등과 같은 로봇 미들웨어들이 존재한다. 하지만, 국내에서는 로봇에 대한 관심이 부족하기 때문에 로봇 분야에 대한 빠른 확산을 위해 로봇을 연구하고 개발하는 많은 기업 및 학교에서 쉽게 로봇 분야에 접근할 수 있도록 하는 국산화된 로봇 미들웨어를 제공해야 한다.

본 논문에서는 국내 기술로 개발되는 로봇 미들웨어에서 실시간성을 지원하는 실시간 프레임워크(Real-Time Framework)를 설계 및 구현한다. 실시간 프레임워크는 범용 운영체제에서 제공하는 스케줄러의 실시간성 미 지원 문제점을 보완하기 위해 우선순위 기반 스케줄러를 포함한 프레임워크 형태로 구현하였으며, 이를 통해 단일 노드 상에서 로봇 컴포넌트를 구성하는 쓰레드(Thread)에 실시간 스케줄링 기법을 적용함으로써 로봇 미들웨어에 실시간성을 지원한다. 실시간 프레임워크의 기반을 범용 운영체제로 선택한 이유는 현재 상용화되어 사용되고 있는 로봇들과의 호환성을 유지하기 위함이다.

실시간 프레임워크의 로봇 컴포넌트를 구성하는 쓰레드에 실시간성을 지원하기 위해서는 운영체제에서 지원하는 실시간 스케줄러(Real-Time Scheduler)를 사용하여 쓰레드를 스케줄링(Scheduling)해야 한다. 하지만, 범용 운영체제의 커널에서 제공되는 스케줄러는 실시간 스케줄링 기법을 제공하지 않기 때문에, 범용 운영체제에서 제공하는 쓰레드 관리 함수와 POSIX 라이브러리[10][11]를 사용하여 운영체제의 쓰레드를 직접 실행(Resume), 정지(Suspend)시켜 제어함으로써 실시간 스케줄링 기법을 적용한 스케줄러를 포함하는 실시간 프레임워크를 설계 및 구현하였다.

실시간 프레임워크는 우선순위 기반 스케줄러(Priority-based Scheduler), 쓰레드 제어 매니저(Thread Control Manager), 세마포어 제어 매니저(Semaphore Control Manager)를 포함하며, 실시간 프레임워크내의 자체적인 스케줄링을 위해 타이머 함수를 포함하고 있다. 실시간 프레임워크의 우선순위 기반 스케줄러를 사용해 쓰레드의 응답시간을 향상시킴으로써 실시간성 지원을 위한 논리적 정확성 및 시간 결정성을 보장한다.

본 논문의 구성은, 2장에서는 관련 연구로써 상용화된 로봇 미들웨어와 윈도우즈(Windows) 스케줄링 기법에 대해 기술하며, 3장에서는 실시간 프레임워크의 설계 및 구현 내용을, 4장에서는 실시간 프레임워크의 성능 측정 방법에 대해서 기술한다. 마지막으로, 5장에서는 결론 및 향후 연구과제에 대해서 기술한다.

II. 관련연구

현재 상용화된 로봇 미들웨어인 OROCOS, MSRS, MIRO의 개요 및 실시간성 지원 사항에 대해 기술하며, 실시간 프레임워크의 스케줄러와 비교하기 위해 범용 운영체제 중에 하나인 윈도우즈 운영체제의 스케줄링 기법에 대해 기술한다.

1. OROCOS

OROCOS(Open RObot COntrol Software)는 2001년 9월부터 EURON(European Robotics Research Network)의 지원으로 4개국의 대학에서 연구가 시작되었다. 로봇과 머신 제어(Machine control)를 위한 플랫폼에 독립적인 프레임워크이며, 로봇 애플리케이션 개발을 위해 4개의 C++ 라이브러리를 제공한다[2-4].

1.1 OROCOS의 구조

[그림 1]은 OROCOS의 전체적인 구조를 나타낸 것이다. OROCOS에서 실시간성을 지원하며 로봇 컴포넌트 개발에 필요한 라이브러리를 포함하고 있는 RTT(Real

-Time Toolkit)는 로봇 컴포넌트 간의 통신에 필요한 메서드 호출, 명령어 전달, 데이터 포트 등과 같은 다양한 기법을 지원한다.

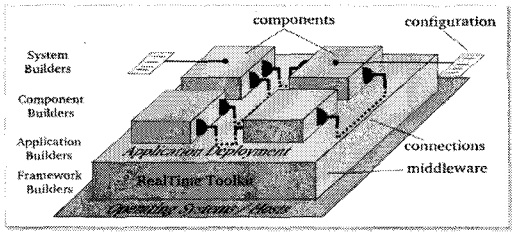


그림 1. OROCOS의 아키텍처

1.2 OROCOS에서 제공하는 실시간성

RTT는 OROCOS의 로봇 컴포넌트에 실시간성 기능을 제공한다. RTT에서 실시간성 지원과 관련된 기능은 운영체제의 실시간 스케줄러를 상위 로봇 컴포넌트에서 사용할 수 있도록 연결시켜주는 추상화 계층(Abstraction Layer)을 제공함으로써 실시간성을 지원한다.

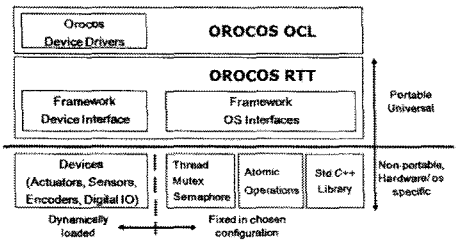


그림 2. OROCOS의 운영체제 추상화 계층

[그림 2]의 추상화 계층은 쓰레드, 뮤텍스(Mutex), 세마포어(Semaphore)와 같이 운영체제에서 제공하는 최소한의 집합으로 구성된 C++ 인터페이스이다. 추상화 계층에 포함된 쓰레드 제어 API를 사용하여 생성된 쓰레드는 운영체제의 실시간 스케줄러에 의해 스케줄링 되기 때문에 실시간성을 가지게 된다. 이렇게 생성된 쓰레드를 사용하여 로봇 컴포넌트를 개발함으로써 다수의 로봇 컴포넌트로 구성되는 최상위 로봇 응용 프로그램에 실시간성을 지원하게 된다. 하지만, RTT는 실시간 스케줄러를 포함하고 있는 eCos[12], RTAI/LXRT[13][14], Xenomai[15]와 같은 운영체제를 기반으로 동작할 때 진정한 실시간성을 지원하게 된다. 반면, 실시간 스케줄링을 지원하지 않는 GNU Linux 계열의 운영체제를 기반으로 OROCOS가 동작하게 된다면 실시간성을 지원하지 못하게 된다[2-4].

2. MSRS

MSRS(Microsoft Robotics Studio)는 일반인들이 프로그래밍에 대한 기본적인 지식만 가지고 있으면, 다양한 로봇 하드웨어 상에서 필요로 하는 다양한 로봇 애플리케이션을 쉽게 개발 할 수 있도록 지원하는 개발 툴 및 환경이다. MSRS의 로봇 운영체제로는 윈도우즈 계열 운영체제를 사용한다[6][7].

2.1 MSRS의 동시처리 및 조정기술(CCR)

MSRS에서는 CCR(Concurrency and Coordination Runtime) 기술을 제공한다. CCR은 고도의 동시성, 쓰레드, 락(Lock), 세마포어를 사용하지 않고 오케스트레이션(Orchestration)을 지원하는 메시지 기반 프로그래밍 모델이다.

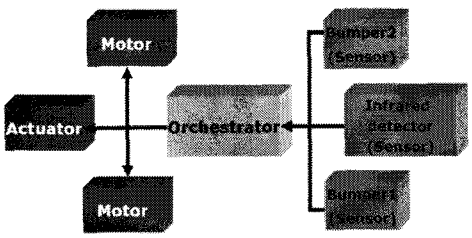


그림 3. 오케스트레이터 동작 과정

[그림 3]은 컴포넌트들의 메시지를 처리하는 오케스트레이터의 동작 과정을 나타낸 것이다. CCR의 오케스트레이터는 4가지 메시지 처리 정책 시나리오를 정의하고 있으며, 개발자가 선택한 메시지 처리 정책에 의해서 컴포넌트로부터 입력된 메시지를 처리한다. 즉, 메시지에 대한 처리 순서를 개발자가 아닌 CCR에 포함된 오케스트레이터에 의해서 내부적로 처리하게 되는 것이다. 이처럼 CCR은 메시지에 대한 처리를 로봇 미들웨어에서 자체적으로 수행하며 개발자에게 메시지 처

리에 대한 스케줄링 권한을 주지 않으며, 실시간성 지원을 위한 별도의 API를 제공하지 않는다.

3. MIRO

MIRO(Micro-Middleware for Mobile Robot Application)는 모바일 로봇 제어를 위한 CORBA기반의 분산 객체지향 프레임워크로, MIRO의 코어 컴포넌트들은 ACE(Adaptive Communication Environment)의 뒷받침 아래 리눅스 환경에서 C++로 개발되었다. ACE는 동시처리방식 통신 소프트웨어의 많은 핵심 패턴들을 구현한 오픈소스 기반의 객체지향 프레임워크로, 네트워크 프로그래밍의 단점을 해결하기 위해 만들어졌다[8][9].

3.1 TAO 기반의 실시간성

MIRO는 TAO(The ACE ORB)를 기반으로 동작하며, TAO는 DOC(Distributed Object Computing) 그룹에서 만들어진 DRE(Distributed Real-time Embedded) 시스템을 위한 오픈소스의 분산 미들웨어 플랫폼이다. TAO는 앞서 말한 ACE에 의해 구현된 프레임워크 컴포넌트 패턴을 사용하여 구축된 CORBA이다[16][17].

MIRO 내부에는 실시간성을 지원하는 라이브러리 및 API를 제공하지 않는다. 대신, TAO를 기반으로 하기 때문에 TAO에서 제공하는 실시간성 기능을 사용하여 로봇 미들웨어에 실시간성 기능을 지원하게 된다. 그러나 TAO도 VxWorks, LynxOS와 같은 실시간 운영체제를 기반으로 동작해야 진정한 실시간성을 상위 미들웨어인 MIRO에게 지원하게 되는 것이다.

4. 윈도우즈 쓰레드 스케줄링 기법

4.1 프로세스와 쓰레드의 우선순위 스케줄링

범용 운영체제에서 특정 임무를 수행하기 위해서 프로세스와 쓰레드가 존재한다. 쓰레드는 프로세스에 포함되어 코드(Code), 데이터(Data) 영역을 공유하며 자신만의 스택(Stack) 영역을 가지고 맡은 임무를 수행한다. 쓰레드는 유저 레벨 쓰레드(User-level thread)와 커널 레벨 쓰레드(Kernel-level thread)로 나누어진다. 윈도우즈에서는 커널 레벨 쓰레드 방식을 사용한다. 이는 쓰레드가 커널 스케줄러(Kernel scheduler)에 의해서 CPU를 할당받아 스케줄링 되며, 시스템 콜 사용 시 블로킹(Blocking) 되지 않는다. [그림 4]에서 보여 지는 것처럼 프로세스의 우선순위에 관계없이 운영체제상에 생성된 모든 쓰레드의 우선순위에 의해서 스케줄링 되는 스케줄링 기법이다.

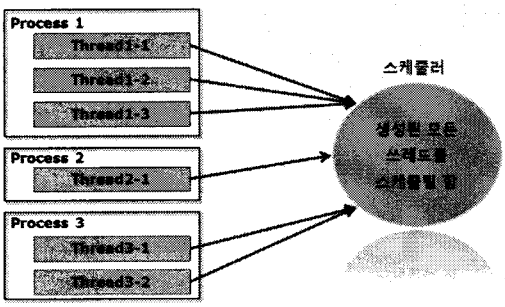


그림 4. 윈도우즈 운영체제의 쓰레드 스케줄링

윈도우즈 운영체제는 라운드로빈 우선순위기반 스케줄러(Round-robin Priority-based Scheduler)를 제공한다. 쓰레드 큐(Queue)에 쓰레드를 삽입할 때는 선입선출(First-In First-Out)방식이 아닌 우선순위를 기반으로 해서 삽입하게 된다. 큐에 삽입된 첫 번째 쓰레드에 CPU 권한을 주어 실행시키며, 타임 슬라이스(Time slice)가 지난 후에 큐의 두 번째 쓰레드에게 CPU 권한을 넘겨주어 실행한다[18-20].

4.2 프로세스와 쓰레드의 우선순위

윈도우즈 커널은 실행 준비가 된 쓰레드들 중 가장 높은 우선순위를 가진 쓰레드를 실행한다. 쓰레드가 어떠한 이유로 대기나 일시 정지, 차단 상태이면 실행 준비 상태로 간주하지 않으며 따라서 실행이 되지 않는다.

쓰레드에서는 쓰레드가 속한 프로세스의 우선순위 클래스(Priority class)에 상대적인 우선순위가 부여된다. 프로세스 우선순위 클래스는 총 네 개이며 CreateProcess() 호출 시에 최초로 설정된다. 그리고 각 클래스마다 기준 우선순위(base priority)가 있다.

[표 1]은 우선순위 클래스들과 그 기준 우선순위 이다.

표 1. 우선순위 클래스(Priority Class)

우선순위 클래스 이름	설명
IDLE_PRIORITY_CLASS	기준 우선순위 4
NORMAL_PRIORITY_CLASS	기준 우선순위 9 또는 27
HIGH_PRIORITY_CLASS	기준 우선순위 13
REALTIME_PRIORITY_CLASS	기준 우선순위 24

가장 낮은 클래스와 가장 높은 클래스는 극단적인 사례로, 별로 쓰이지 않는다. 대부분의 경우에는 그 사이의 두 클래스들이 쓰인다. 윈도우즈 NT(모든 버전)는 실시간 운영체제가 아니지만, 윈도우즈 CE는 실시간 운영체제이다. 따라서 윈도우즈 CE의 경우 한 프로세스에 REALTIME_PRIORITY_CLASS를 지정하면 다른 프로세스들에게는 실행의 기회가 돌아가지 않을 수 있다. 보통 우선순위(NORMAL_PRIORITY_CLASS)의 경우 윈도우가 키보드 입력 포커스를 가지고 있을 때에는 기준 우선순위가 0이고 그렇지 않으면 7이다.

쓰레드의 우선순위는 프로세스 기준 우선순위에 상대적으로 결정된다. 쓰레드 생성 시에는 기본적으로 프로세스의 기준 우선순위와 같은 우선순위가 설정된다. 쓰레드 우선순위는 프로세스 기준 우선순위의 더하기 2 또는 빼기 2 범위이다. 즉, 총 5개의 상대적 우선순위들이 가능한데, 그에 해당하는 상수들은 다음과 같다.

표 2. 우선순위 레벨(Priority Level)

우선순위 레벨 이름	값
THREAD_PRIORITY_LOWEST	-2
THREAD_PRIORITY_BELOW_NORMAL	-1
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_ABOVE_NORMAL	1
THREAD_PRIORITY_HIGHEST	2

III. 로봇 컴포넌트에 실시간성을 지원하기 위한 프레임워크 설계 및 구현

1. 설계 시 고려사항

로봇 컴포넌트에 실시간성을 지원하기 위한 실시간 프레임워크는 기존에 개발되어 상용화되고 있는 로봇들과의 호환성을 고려하여, 로봇 플랫폼의 운영체제로 많이 사용되고 있는 범용 운영체제를 기반으로 설계 및 구현하였다.

실시간 프레임워크에서 쓰레드에 대한 제어를 수행하기 위해 쓰레드의 생성, 삭제 및 관리를 위한 API를 제공하는 쓰레드 제어 매니저를 설계 및 구현 하였다. 또한, 실시간 프레임워크의 우선순위 기반 스케줄러에 의해서 스케줄링 되는 쓰레드가 자원을 요청해서 기다릴 때 범용 운영체제내의 Pending List(운영체제 내부의 Pending List)에 추가되면 쓰레드의 제어권이 실시간 프레임워크에서 범용 운영체제로 넘어가게 되어 쓰레드를 직접 제어할 수가 없다. 그렇기 때문에 자원을 요청 및 반환하는 쓰레드를 실시간 프레임워크에서 직접 제어하기 위해서 운영체제에 독립적인 세마포어 제어 매니저를 설계 및 구현하였다.

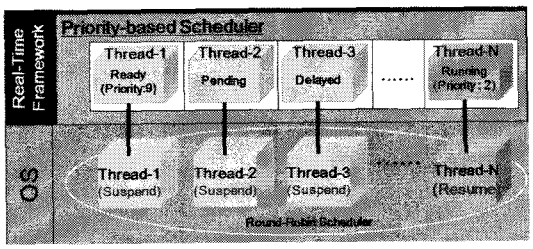


그림 5. 실시간 프레임워크의 쓰레드와 운영체제 쓰레드와의 관계

실시간 프레임워크의 쓰레드는 [그림 5]에서 처럼 운영체제의 쓰레드와 1대1로 매칭 된다. 이때 실시간 프레임워크의 쓰레드는 Ready, Pending, Delayed 및 Running과 같은 다양한 쓰레드의 상태를 갖는다. 실시간 프레임워크는 사용자 영역(User-level)에서 쓰레드들을 관리하지만, 실제적으로 쓰레드에 대한 실행(Resume)과 정지(Suspend) 상태를 조절하기 위해서 커널 영역(Kernel level)에서 제공하는 ResumeThread(), SuspendThread()와 같은 시스템 호출 함수를 사용하여 쓰레드가 사용자 영역에서 우선순위와 상태 정보를 기반으로 스케줄링 되는 우선순위 기반 스케줄러를 설계 및 구현하였다.

2. 실시간 프레임워크의 구조

[그림 6]은 실시간 프레임워크의 전체적인 구조를 나타낸 것이다. 실시간 프레임워크의 스케줄러는 범용 운영체제에서 제공하는 쓰레드 관리 함수와 POSIX 라이브러리를 사용하여 운영체제의 쓰레드(Thread)를 직접 실행(Resume), 정지(Suspend)시켜 제어하는 우선순위 기반 스케줄러를 구현하여 범용 운영체제에서 제공하는 스케줄러의 실시간성 미 지원 문제점을 보완하였다. 실시간 프레임워크는 다음 3가지 모듈로 구성된다 :

- 쓰레드 제어 매니저(Thread Control Manger)
- 우선순위 기반 스케줄러(Priority-based Scheduler)
- 세마포어 제어 매니저(Semaphore Control Manager)

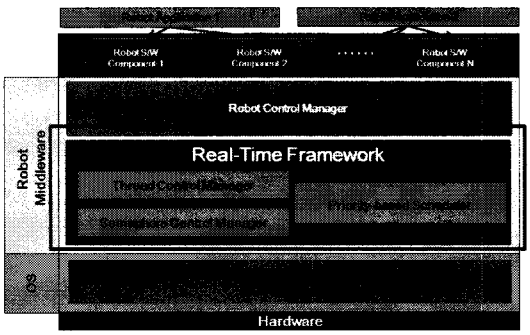


그림 6. 실시간 프레임워크의 구조

3. 실시간 프레임워크 모듈의 구현

3.1 쓰레드 제어 매니저

로봇 애플리케이션은 다수의 독립적인 컴포넌트로 구성되며, 각각의 독립적인 컴포넌트는 다수의 쓰레드에 의해서 구성되어 컴포넌트의 역할을 수행한다. 각 쓰레드는 특정 임무를 수행하기 위해서 실행 함수(Execution function), 우선순위(Priority) 및 타임 슬라이스(Time slice)를 포함하며, 쓰레드 제어 매니저는 쓰레드에 대한 생성, 삭제 및 관리를 위한 API를 제공한다. [표 3]은 쓰레드 제어 매니저에서 쓰레드의 생성 및 삭제 역할을 수행하는 API이다.

표 3. 쓰레드 제어 매니저의 API

함수명	함수 기능
RO_CreateThread()	쓰레드 생성
RO_DeleteThread()	쓰레드 삭제

[그림 7]은 실시간 프레임워크 내부의 쓰레드를 생성하는 RO_CreateThread() 함수의 소스이다. [그림 7]의 1번 소스는 쓰레드의 데이터를 저장하는 쓰레드 제어 블록(Thread Control Block, TCB)을 초기화하는 코드이며, 2번 소스는 쓰레드를 관리하는 리스트를 초기화하는 코드이다. [그림 7]의 3번 소스는 POSIX 라이브러리를 사용해 쓰레드를 생성하고 윈도우즈 운영체제의 Win32API 함수를 사용하여 실시간 프레임워크에서 생성된 쓰레드를 정지(Suspend)시켜서 제어하는 코드이다.

```
RO_THREAD* RO_CreateThread(
                RO_THREAD *roThread,
                int priority,
                const char* name,
                long timeSlice,
                void *(*start_routine)(void*),
                void *arg)
{
   int pid;
   int Flags;

1  roThread->status      =       RO_THREAD_SUSPEND;
   roThread->priority    =       priority;
   roThread->thName = malloc(sizeof(name)+1);
   strcpy(roThread->thName, name);
   roThread->t_RemainedTimeSlice= timeSlice;
   roThread->t_TimeSlice        = timeSlice;
   roThread->t_UsedTimeSlice    = 0;
   roThread->wrapper            = start_routine;
   roThread->arg                = arg;
   roThread->isResumed          = RO_FALSE;
   roThread->t_pThreadNext      = NULL;
   roThread->t_pThreadPrev      = NULL;

2  if(RO_pThreadListHead == NULL)
   {
       RO_pThreadListHead = roThread;
       RO_pThreadListTail = roThread;
   }
   else
   {
       RO_pThreadListTail->t_pThreadNext = roThread;
       roThread->t_pThreadPrev = RO_pThreadListTail;
       RO_pThreadListTail = roThread;
   }

   roThread->t_pThreadReadyNext = NULL;
   roThread->t_pThreadReadyPrev = NULL;
   roThread->t_pDelayedNext     = NULL;
   roThread->t_pDelayedPrev     = NULL;
   roThread->t_pPendingNext     = NULL;
   roThread->t_pPendingPrev     = NULL;

3  // Create thread
   pid = pthread_create( &(roThread->pthreadID), 0, RO_ThreadShell, roThread );

   // Get thread handler for Windows
   roThread->thHandler = pthread_getw32threadhandle_np(roThread->pthreadID);
   SuspendThread(roThread->thHandler);

   return NULL;
}//end of CreateThread()
```

그림 7. RO_CreateThread() 소스

3.2 우선순위 기반 스케줄러

3.2.1 쓰레드 제어 블록(Thread Control Block)

실시간 프레임워크는 쓰레드의 실행(Resume) 및 정지(Suspend)와 같은 제어를 범용 운영체제의 스케줄러에 의존하지 않고 직접 제어가 가능하도록 하기 위해서 우선순위 기반 스케줄링 기법을 적용한 스케줄러를 설계 및 구현 하였다. 실시간 프레임워크 내부적으로 구현한 우선순위 기반 스케줄러에 의해 실행되는 쓰레드는 자신만의 문맥(Context)을 저장하는 TCB(Thread Control Block)를 소유하게 되며, TCB는 실시간 프레임워크 상에서 쓰레드의 상태 변화에 대한 데이터를 저장할 수 있는 다음과 같은 각종 변수들을 포함한다:

- 쓰레드의 상태(State)
- 우선순위(Priority)
- 쓰레드 핸들러(Handler)
- 타임 슬라이스(Time slice)
- 실행 함수 포인터(Execution function pointer)

[표 4]는 실시간 프레임워크에서 스케줄링을 수행하는 API이다.

표 4. 우선순위 기반 스케줄러의 API

함수명	함수 기능
RO_Schedule()	우선순위 기반 스케줄러 - 내부 타이머에 의해서 주기적으로 스케줄링 수행
RO_Resume()	쓰레드를 Ready 상태로 전환
RO_Suspend()	쓰레드를 Suspend 상태로 전환
RO_ContextSwitch()	실행중인 쓰레드를 중지시키고 다른 쓰레드를 실행시킴
RO_SleepTicks()	쓰레드를 주어진 시간동안 Delayed 상태로 전환
RO_InsertThread ToReadyList()	쓰레드를 Ready 리스트에 추가
RO_DeleteThread FromReadyList()	쓰레드를 Ready 리스트에서 삭제

3.2.2 쓰레드의 상태 천이

실시간 프레임워크 상의 쓰레드는 [그림 8]에 보이는 것처럼 5개의 상태로 구별된다. 쓰레드가 생성되어지면 기본적으로 Suspend 상태가 되며, 쓰레드가 활성화되면 CPU 제어권을 얻어 실행 될 수 있는 Ready 상태가 된다. Ready 상태인 쓰레드가 CPU 제어권을 얻으면 Running 상태가 되어 쓰레드의 TCB에 포함된 정보를 기본으로 하여 쓰레드의 실행 함수가 수행된다. 또한, 특정 자원(resource)을 기다리기 위해서는 Pending 상태로 전환이 되어 지며, 이때 특정 시간 동안만큼만 자원을 기다리게 된다면 Pending 상태와 Delayed 상태가 된다. Delayed 상태는 임의의 시간동안 CPU 제어권을 스스로 다른 쓰레드에게 넘겨주는 상태이며, 정해진 시간이 지난 후에 쓰레드는 Delayed 상태에서 Ready 상태로 전환된다.

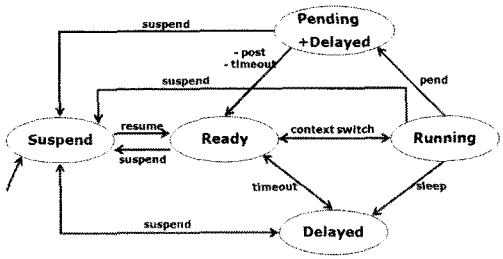


그림 8. 쓰레드의 상태 천이도

3.2.3 실시간 프레임워크의 스케줄링 기법

실시간 프레임워크의 스케줄러는 CPU 제어권을 우선순위가 가장 높은 Ready 상태의 쓰레드에게 할당한다. 실시간 프레임워크는 선점형(preemptive)이기 때문에, 실행중인 쓰레드보다 높은 우선순위를 가진 쓰레드가 Ready 상태가 되면 CPU 제어권을 새로운 쓰레드에게 넘겨주게 된다. 또한, 동일 우선순위를 지원하며, 동일 우선순위 쓰레드는 라운드 로빈(Round-robin) 방식으로 스케줄링 된다.

[그림 9]는 실시간 프레임워크의 우선순위 기반 스케줄러의 RO_Schedule() 함수의 소스이다. 쓰레드의 상태(State) 변화로 인해 높은 우선순위의 쓰레드가 활성화될 경우, 자원에 대한 요청과 반환으로 인해 쓰레드의 상태가 변경될 경우, 실행중인 쓰레드가 CPU 권한을 스스로 다른 쓰레드에게 넘겨줄 경우, 타이머에 의해서 주기적으로 틱 이벤트가 발생할 경우에 RO_Schedule() 함수가 실행되어 스케줄링이 수행된다.

```
void RO_Schedule(void)
{
  RO_THREAD  *pOldThread, *pNewThread;
  int   CurrentPriority;
  int   HighestPriority;

  if(RO_ContextSwitchFlags == RO_TRUE)
  {
     pOldThread = RO_pCurrentThread;
     CurrentPriority = pOldThread->priority;

1    if(RO_pThreadReadyListHead[CurrentPriority]
                    != RO_pThreadReadyListTail[CurrentPriority])
     {
        if(pOldThread->t_RemainedTimeSlice <= 0)
        {
           if(pOldThread->status & RO_THREAD_READY)
           {
              // ...
           }
           pOldThread->t_RemainedTimeSlice = pOldThread->t_TimeSlice;
        }
     }

     HighestPriority = RO_GetHighPriority();

2    if( !(pOldThread->status & RO_THREAD_READY)
                    || (HighestPriority < CurrentPriority) )
     {
        pNewThread = RO_pThreadReadyListHead[HighestPriority];
        RO_pCurrentThread = pNewThread;

        RO_ContextSwitch(pOldThread, pNewThread);
     }
3    else if(RO_pThreadReadyListHead[HighestPriority]
                    != RO_pThreadReadyListTail[HighestPriority])
     {
        pNewThread = RO_pThreadReadyListHead[HighestPriority];
        RO_pCurrentThread = pNewThread;

        RO_ContextSwitch(pOldThread, pNewThread);
     }
4    else       /* When Only One Task exists */
     {
        pOldThread->t_RemainedTimeSlice = pOldThread->t_TimeSlice;
     }
  }
}//end of RO_Schedule()
```

그림 9. RO_Schedule() 소스

[그림 9]의 1번 소스는 현재 실행중인 쓰레드와 동일 우선순위의 쓰레드를 처리하는 코드이다. 실행중인 쓰레드의 남아있는 타임 슬라이스(t_RemainedTimeSlice) 값이 0이거나 작은 경우에 실행중인 쓰레드를 동일 우선순위 쓰레드 리스트의 첫 번째에서 마지막으로 이동시킨다. [그림 9]의 2번 소스는 실행중인 쓰레드보다 높은 우선순위를 가진 쓰레드가 존재하면 문맥 교환(Context switch)을 통해 CPU권한을 넘겨준다. [그림 9]의 3번 소스는 실행중인 쓰레드보다 높은 우선순위의 활성화된 쓰레드가 없고, 동일 우선순위의 쓰레드가 존재할 경우 CPU권한을 동일 우선순위의 쓰레드에게 넘겨준다. 마지막으로 [그림 9]의 4번은 현재 생성된 쓰레드가 Idle 쓰레드와 실행중인 쓰레드만 존재할 경우 실행중인 쓰레드의 타임 슬라이스 값을 갱신시킨다.

```
void RO_ContextSwitch(RO_THREAD *pOldThread, RO_THREAD *pNewThread)
{
   RO_pCurrentThread = pNewThread;

   if( !pNewThread->isResumed )
   {
      pNewThread->isResumed = RO_TRUE;
      ResumeThread(pNewThread->thHandler);Win32 API
   }

   if( pOldThread->isResumed & pOldThread!=pNewThread )
   {
      pOldThread->isResumed = RO_FALSE;
      SuspendThread(pOldThread->thHandler)Win32 API
   }
}//end of RO_ContextSwitch()
```

그림 10. RO_ContextSwitch() 소스

[그림 10]은 범용 운영체제에 탑재한 실시간 프레임워크에 대한 테스트를 위해 윈도우즈 운영체제에서 구현한 문맥 교환 소스이다. [그림 10]에서 ResumeThread(), SuspendThread() 함수는 윈도우즈 운영체제에서 제공하는 Win32 API로써, 윈도우즈에서 생성된 쓰레드를 실행(Resume), 정지(Susepnd)와 같이 제어할 수 있는 함수이다. 이 함수들을 사용하여 윈도우즈의 쓰레드를 윈도우즈 스케줄러의 제어를 벗어나, 실시간 프레임워크 내부의 우선순위 기반 스케줄링 기법을 적용한 스케줄러에 의해서 직접 제어 가능하게 된다.

실시간 프레임워크는 하드웨어에서 제공하는 클록 틱 인터럽트(Clock tick interrupt)를 직접 사용할 수 없어 범용 운영체제에서 제공하는 타이머(Timer) API를 사용하여 주기적으로 틱 이벤트를 발생시키도록 하였다.

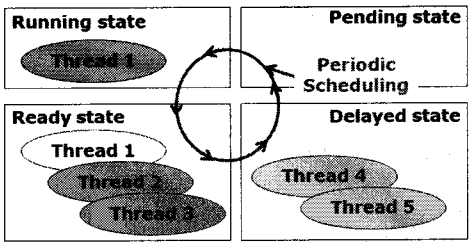


그림 11. 스케줄러의 주기적인 스케줄링

[그림 11]은 틱 이벤트가 발생하였을 경우 스케줄링 되는 것을 나타낸 것이다. 틱 이벤트가 발생하면 실시간 프레임워크는 RO_TimeTick() 함수를 사용하여 실행중인 쓰레드의 타임 슬라이스를 1만큼 감소시키며, RO_Wakeup() 함수를 호출하여 Delayed 리스트에 존재하는 쓰레드의 대기 시간을 나타내는 틱 값을 1씩 감소시킨다. 그리고 상태가 변한 쓰레드들을 스케줄링 하여 가장 높은 우선순위를 가지는 Ready 상태의 쓰레드에게 CPU 권한을 넘겨준다. 이처럼 실시간 프레임워크는 내부에서 쓰레드의 타임 슬라이스를 직접 관리하는 타이머를 포함하고 있으며, 슬립(Sleep) 함수를 사용하여 Delayed 상태로 전이 될 때 윈도우즈에서 제공되는 슬립 함수가 아닌 프레임워크의 내부 타이머를 이용한다. 구현된 슬립 함수를 사용함으로써 각각의 쓰레드별로 다른 타임 슬라이스를 주어 실행시킬 수 있다. [표

5]는 쓰레드의 주기적인 스케줄링 및 Delayed 상태의 쓰레드를 관리하기 위한 API를 나타낸다.

표 5. 실시간 프레임워크의 내부 타이머 API

함수명	함수 기능
RO_TimeTick()	내부 타이머 - 프레임워크에서 주기적인 타이머에 의해서 실행되는 함수로써, 실행중인 쓰레드와 Delayed 상태인 쓰레드의 타임 슬라이스 값을 변경 - 주기적으로 RO_Schedule() 함수를 호출하여 스케줄링 수행
RO_Wakeup()	- RO_TimeTick() 함수에 의해서 실행되는 함수로써, Delayed 리스트에 있는 쓰레드의 타임 슬라이스 값을 변경하고, Delayed 상태가 만료된 쓰레드를 Ready 상태로 전환시킴

3.2.4. 실시간 프레임워크의 우선순위 기반 스케줄러의 장점

실시간 프레임워크의 우선순위 기반 스케줄러는 범용 운영체제의 스케줄러와 크게 2가지 차이점을 가지고 있다.

첫 번째, 범용 운영체제의 스케줄러는 라운드로빈 우선순위 기반 스케줄러(Round-robin Priority-based Scheduler)를 제공한다. 쓰레드 큐(Queue)에 쓰레드를 삽입할 때는 선입선출(First-In First-Out)방식이 아닌 우선순위를 기반으로 해서 삽입하게 된다. [그림 12]의 (a)는 범용 운영체제에서 우선순위가 다른 3개의 쓰레드를 생성하여 스케줄링 되는 과정을 나타낸 것이며, 쓰레드의 숫자는 우선순위를 나타낸다. 우선순위가 높은 쓰레드가 큐에 가장 앞에 추가되어 CPU 권한을 얻어 실행된 후에 타임 슬라이스가 지나면 쓰레드의 모든 작업을 종료하지 않았어도, 그 다음 우선순위가 높은 쓰레드에 CPU 권한을 넘겨주게 된다. 이러한 스케줄링 기법은 우선순위가 높은 쓰레드에게 시간 결정성을 보장해 주지 못하기 때문에 실시간성을 지원하지 않게 된다. 이러한 단점을 보완하기 위하여, 실시간 프레임워크의 스케줄러는 우선순위 기반의 스케줄링 기법을 지원한다. [그림 12]의 (b)는 실시간 프레임워크의 우선순위 기반 스케줄러에 의해서 3개의 쓰레드가 스케줄링 되는 것을 보여준다. 쓰레드 2가 실행 중에 높은 우선순위를 가지는 쓰레드 0과 1이 활성화됨에 따라서 CPU 권한을 쓰레드 0에게 넘겨주어 실행된다. 이처럼 더 높은 우선순위의 쓰레드가 활성화될 경우 CPU권한을 넘겨주어 바로 실행될 수 있도록 한다. 실시간 프레임워크의 쓰레드 0과 쓰레드 1은 [그림 12]의 (b)에서 보이는 것처럼 범용 운영체제보다 빠른 응답 시간을 갖게 된다. 이로 인해 실시간 프레임워크에서는 우선순위가 높은 쓰레드의 시간 결정성을 보장 할 수 있게 된다.

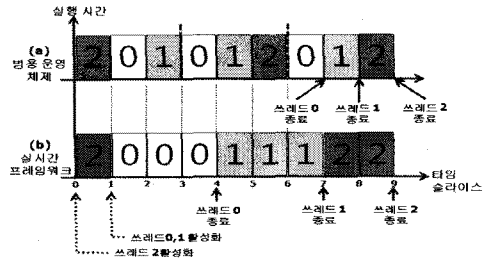


그림 12. 범용 운영체제와 실시간 프레임워크에서의 쓰레드 스케줄링

두 번째, 각각의 쓰레드에 타임 슬라이스 값을 다르게 할당할 수 있다. 범용 운영체제는 각 쓰레드에 다른 타임 슬라이스 값을 할당할 수 있는 API를 제공하지 않으며, 쓰레드는 운영체제에 의해 정해진 타임 슬라이스 값을 가진다. 이 값은 범용 운영체제의 커널 데이터를 수정하여 변경할 순 있지만 모든 쓰레드에 동일하게 적용이 된다. [그림 13]은 실시간 프레임워크에서 동일 우선순위의 4개의 쓰레드가 다른 타임 슬라이스를 가지고 스케줄링 되어 실행되는 것을 나타낸 것이다. 각 쓰레드의 타임 슬라이스는 쓰레드-0은 3, 쓰레드-1은 1, 쓰레드-2는 2, 쓰레드-3은 1의 값을 가진다. 동일 우선순위의 실행 순서는 생성된 쓰레드의 순서에 따라서 주어진 타임 슬라이스를 가지고 라운드 로빈(Round-Robin) 방식으로 스케줄링 된다.

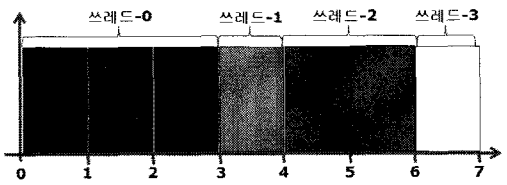


그림 13. 상이한 타임 슬라이스 값을 가진 동일 우선순위의 쓰레드 스케줄링

3.3 세마포어 제어 매니저

세마포어(Semaphore)는 쓰레드들의 자원 공유를 관리하며, 쓰레드들 간의 동기화(Synchronization)시에 사용되어진다. 세마포어는 바이너리(Binary) 세마포어, 카운팅(Counting) 세마포어로 나누어진다.

실시간 프레임워크의 세마포어 제어 매니저는 범용 운영체제에서 제공하는 세마포어 API를 사용하지 않고, 실시간 프레임워크 내부에 자체적인 세마포어 구조체를 선언하여 세마포어 관리 API를 설계 및 구현하였다. 이는 쓰레드가 세마포어를 요청하고 반환할 때와 같은 이벤트가 발생되었을 경우에 쓰레드의 상태는 변하게 된다. 쓰레드의 상태 변화에 따라 즉각적인 스케줄링을 수행함으로써 Ready 상태이면서 우선순위가 가장 높은 쓰레드에게 CPU권한을 넘겨주어 실행함으로써 실시간성을 지원하게 된다. [표 6]은 세마포어 제어 매니저에서 제공하는 API이다.

표 6. 세마포어 제어 매니저의 API

함수명	함수 기능
RO_CreateSemaphore()	세마포어 생성
RO_SemaphorePend()	세마포어 요청
RO_SemaphorePost()	세마포어 반환

IV. 실험 환경 및 결과

본 논문에서 구현한 실시간 프레임워크는 윈도우즈 운영체제를 기반으로 [표 7]과 같은 H/W를 사용하였다. 쓰레드 실행의 정확한 성능 측정을 위하여 윈도우즈의 커널 쓰레드 이외의 사용자 프로세스 및 쓰레드를 정지시키고 성능 측정을 수행하였다.

표 7. 실험 서버환경

컴포넌트	특징
Operating System	Windows XP Professional ServicePack 3
CPU	Intel Pentium IV 3.00 GHz
Cache	512MB
RAM	2.00 GB
NIC	10Mbps and 100Mbps
LAN Cable	100Mbps

1. 실험 방법

로봇 컴포넌트에 실시간성을 지원하기 위해서는 논리적 정확성 및 시간 결정성을 보장해야 한다. 논리적 정확성의 보장을 위해 동일한 조건하에 실행되는 쓰레드는 스케줄러에 의해서 항상 동일하게 실행 돼야 한다. 또한, 시간 결정성의 보장을 위해 우선순위에 따른 쓰레드의 응답 시간은 최소화 되어야 한다.

이에 대한 성능 측정을 하기 위해 윈도우즈 운영체제에 실시간 프레임워크를 탑재하여 우선순위가 다른 다수의 쓰레드를 생성하여 윈도우즈 운영체제와 실시간 프레임워크에서의 쓰레드의 응답 시간을 비교 분석하며, 실시간 프레임 워크의 각 모듈이 실행 될 때 걸리는 오버헤드를 측정한다.

2. 쓰레드의 응답 시간 측정

실시간 프레임워크의 쓰레드 응답시간을 측정하기 위해 3가지 경우를 가정하여 윈도우즈 스케줄러와의 응답 시간을 비교 분석하였다. 우선순위가 다른 쓰레드 3개, 5개, 10개를 생성하여 루프를 100회 돌면서 printf를 사용하여 각 쓰레드의 우선순위를 화면에 출력하는 테스트 애플리케이션을 작성한다. 윈도우즈의 테스트 애플리케이션은 Win32 API에서 제공하는 쓰레드를 생성하는 CreateThread() 함수, 쓰레드의 우선순위를 설정하는 SetTheadPriority() 함수, 생성된 쓰레드를 Ready 상태로 활성화 시키는 ResumeThread() 함수를 사용하여 구현하며, 테스트 애플리케이션에서 쓰레드 생성 조건은 다음과 같다 :

1) N개의 쓰레드를 생성한다.(N=3, 5, 10)
2) "쓰레드1, 쓰레드2, ..."처럼 쓰레드의 번호는 각 쓰레드의 우선순위를 나타내며, 낮은 정수 값은 높은 우선순위를 나타낸다.
3) N번째 쓰레드가 처음에 유일하게 활성화되어 실행되며, N번째 쓰레드의 실행 중간에서 나머지 쓰레드가 동시에 활성화된다.

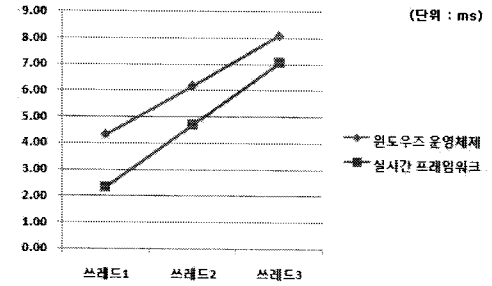


그림 14. 쓰레드3개인 경우의 응답시간 성능 측정 그래프

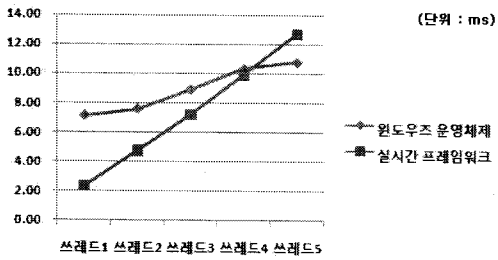


그림 15. 쓰레드5개인 경우의 응답시간 성능 측정 그래프

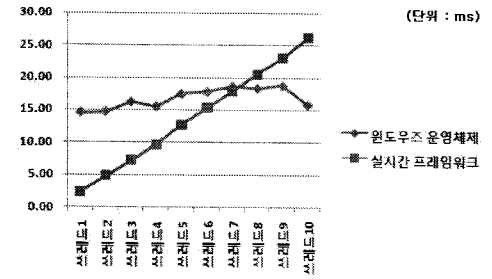


그림 16. 쓰레드10개인 경우의 응답시간 성능 측정 그래프

[그림 14-16]은 앞에서 설명한 쓰레드 생성 조건에 맞추어 우선순위가 다른 쓰레드 3개, 5개, 10개를 생성하여 실시간 프레임워크와 윈도우즈 운영체제에서 쓰레드 응답시간에 대한 성능을 측정한 결과를 나타낸 것이다. 단, 쓰레드 10개를 가지고 테스트되는 애플리케이션은 5개의 그룹으로 나뉘며 각 그룹은 2개의 쓰레드를 포함한다. 한 그룹 내의 쓰레드는 동일 우선순위를 갖으며, 각 그룹의 우선순위는 상이하다. 쓰레드1, 2 그룹이 우선순위가 높고, 쓰레드 9, 10 그룹이 우선순위가 낮다.

표 8. 실시간 프레임워크와 윈도우즈 운영체제에서의 쓰레드 1에 대한 응답시간 (단위 : 밀리세컨드(millisecond))

	실시간 프레임워크	윈도우즈 운영체제
쓰레드 3개인 경우	2.33	4.32
쓰레드 5개인 경우	2.32	7.13
쓰레드 10개인 경우	2.35	14.56

실험 결과를 통해 쓰레드가 CPU 권한을 얻어 실행 중에 더 높은 우선순위를 가진 쓰레드(쓰레드1)가 활성화되어질 경우, 실시간 프레임워크에서의 쓰레드 1의 응답시간이 윈도우즈에서의 쓰레드 1의 응답 시간보다 빠름을 확인할 수가 있다. [표 8]은 실시간 프레임워크와 윈도우즈 운영체제에서의 쓰레드 1의 응답시간이다.

[그림 14-16]에서 나타나듯이 실시간 프레임워크는 우선순위가 가장 높은 쓰레드 1의 응답 시간은 쓰레드의 개수에 상관없이 일정한 수치를 보장하지만, 쓰레드의 개수가 증가함에 따라 우선순위가 가장 낮은 쓰레드의 응답 시간은 쓰레드의 개수에 비례하여 늦춰지게 된다.

3. 실시간 프레임워크와 윈도우즈에서의 쓰레드 스케줄링 비교 분석

쓰레드 생성 및 실행 방법은 앞의 "2. 쓰레드의 응답시간 측정"에서 쓰레드 3개를 생성하여 테스트한 경우와 동일하다.

[그림 17]은 [그림 12]의 (b)에서 표현한 실시간 프레임워크의 우선순위를 기반으로 하는 3개의 쓰레드가 스케줄링 되는 과정을 나타낸 것이다. 우선순위 2인 쓰레드 2가 실행 중에, 더 높은 우선순위를 가지는 쓰레드 0이 활성화되면서 CPU권한은 쓰레드 0이 갖게 된다. 쓰레드 0이 실행 후 다음으로 우선순위가 높은 쓰레드 1이 실행된 후에 마지막으로 쓰레드 2가 실행됨을 확인할 수 있다. [그림 17]의 실행 결과로 실시간 프레임워크는 우선순위를 기반으로 하여 스케줄링 되는 것을 알 수 있다.

그림 17. 실시간 프레임워크에서 3개의 쓰레드 스케줄링 화면

[그림 18]은 [그림 12]의 (a)에서 표현한 윈도우즈에서 우선순위가 다른 3개의 쓰레드가 스케줄링 되는 과정을 나타낸 것이며, 라운드 로빈 스케줄링 방식에 의해 3개의 쓰레드가 스케줄링 된다.

그림 18. 윈도우즈 운영체제에서 3개의 쓰레드 스케줄링 화면

[그림 19]는 [그림 13]에서 표현한 실시간 프레임워크에서의 동일 우선순위에 타임 슬라이스를 상이하게 할당하여 쓰레드가 스케줄링 되는 과정을 나타낸 것이다. 쓰레드는 4개를 생성하였으며 모두 동일한 우선순위를 가지며, 쓰레드 0의 타임 슬라이스는 3, 쓰레드 1의 타임 슬라이스는 1, 쓰레드 2의 타임 슬라이스는 2, 쓰레드 3의 타임 슬라이스는 1로 설정하였다. 또한, 각 쓰레드는 500번의 루프를 돌면서 자신의 번호를 printf로 출력하는 역할을 수행하며, 쓰레드 0부터 쓰레드 4의 순서로 쓰레드가 활성화 된다.

그림 19. 실시간 프레임워크에서 상이한 타임 슬라이스를 가진 쓰레드의 스케줄링 화면

[그림 19]에서 쓰레드 0은 타임 슬라이스 3을 가지고 실행함으로써 한 번의 CPU 권한을 얻어 모든 작업을 완료한다. 두 번째로 쓰레드 1이 수행되며, 작업을 완료하지 못하더라도 타임 슬라이스 1이 지난 후에 CPU권한을 쓰레드 2에게 넘겨준다. 쓰레드 2의 타임 슬라이스 값은 2이기 때문에 쓰레드1이 수행한 작업보다 약 2배 정도의 작업을 수행하였다. 이처럼 각 쓰레드에 설정된 타임 슬라이스 동안만 CPU 권한을 얻어 실행하고, 타임 슬라이스가 만료된 후에 다른 쓰레드에게 CPU 권한을 넘겨주게 된다.

3. 실시간 프레임워크의 모듈별 성능 분석

표 8. 실시간 프레임워크로 인한 모듈별 성능 오버헤드 (단위 : 마이크로세컨드(microsecond))

구 분	기 능	실행시간
쓰레드 제어 매니저	RO_CreateThread() - 쓰레드 생성	62
세마포어 제어 매니저	RO_CreateSemaphore() - 세마포어 생성	0.68
	RO_SemaphorePend() - 세마포어 자원을 요청 후 Pending 상태로 전환 후 다른 쓰레드가 실행될 때까지 걸리는 시간	35
	RO_SemaporePost() - 세마포어 자원을 기다리고 있는 다른 쓰레드에게 자원을 반납하는데 걸리는 시간	8
우선순위 기반 스케줄러	RO_Schedule() - 스케줄러 함수의 실행 시간	0.8
	RO_TimeTick() - 주기적인 스케줄링을 위해 수행되는 타이머 함수의 실행 시간	0.95
	RO_SleepTick() - 쓰레드가 Delayed 상태로 전환되어 우선순위가 높은 쓰레드에게 우선권을 넘겨주는데 걸리는 시간	36.81

[표 8]은 실시간 프레임워크를 구성하는 3가지 모듈의 기능을 수행하는데 걸리는 실행 시간을 측정한 결과이며, 이러한 실행 시간은 성능상의 오버헤드를 발생시킬 수 있다. 하지만 이런 성능상의 오버헤드는 62μs로 로봇 컴포넌트가 수용 가능한 오버헤드이며, 로봇의 실시간 컴포넌트에게는 실시간성 보장이 필수불가결한 요소이다. 또한, 예측 가능한 쓰레드의 실행 순서를 통해 실시간성에서 보장해야 할 특징 중에 하나인 논리적 정확성을 지원할 수 있다.

V. 결론

본 논문에서는 범용 운영체제를 기반으로 하는 로봇

미들웨어의 컴포넌트에 실시간성을 지원하는 실시간 프레임워크를 설계 및 구현하였다. 실시간 프레임워크는 기존 범용 운영체제 스케줄러의 실시간성 미 지원 문제점을 보완하였다. 실시간 프레임워크는 로봇 컴포넌트의 특정 임무를 수행하는 쓰레드를 우선순위 기반 스케줄러를 사용하여 스케줄링 함으로써 시간 결정성 및 논리적 정확성을 보장하여, 최상위 단계인 로봇 애플리케이션에 실시간성을 제공한다. 또한, 동일 우선순위의 쓰레드에게 상이한 타임 슬라이스 값을 할당하여 실행할 수 있도록 하였다.

향후 연구과제로는 단일 노드라는 범위를 벗어나 분산 환경에서의 로봇 컴포넌트에 실시간성을 지원할 수 있는 기법에 대한 연구가 필요하다. 로봇 플랫폼에서 사용되는 통신 미들웨어인 TAO 및 RT-CORBA는 경성 실시간 운영체제(Hard Real-Time Operating System)를 기반으로 해야 진정한 실시간성을 보장하게 된다. 하지만 경성 실시간 운영체제가 아닌, 범용 운영체제 상에서 RM(Rate Monotonic)과 같은 경성 실시간 스케줄러(Hard Real-Time Scheduler)를 포함한 프레임워크를 구현함으로써 범용 운영체제에 탑재되는 로봇 미들웨어에 다양한 스케줄링 기법을 지원할 수 있도록 지속적으로 연구를 진행해야 할 것이다.

참 고 문 헌

[1] D. Yu, H. Lin, R. Guo, J. Yang, and P. Xiao, "Research on real-time middleware for open architecture controller," Embedded and Real-Time Computing Systems and Applications, 2005. Proceddings. 11th IEEE International Conference on, pp.80-83, 2005(8).

[2] http://www.orocos.org/

[3] H. Bruyninckx, P. Soetens, and B. Koninckx, "The Real-Time Motion Control Core of the Orocos Project," IEEE International Conference on Robotics & Automation, Vol.2, pp.2766-2771, 2003.

[4] H. Bruyninckx, "Open Robot Control Software: the OROCOS project," IEEE International Conference on Robotics & Automation, Vol.3, pp.2523-2528, 2001.

[5] J. Lee, J. Park, S. Han, and S. Hong., "RSCA:middleware supporting dynamic reconfiguration of embedded software on the distributed URC robot platform," The First International Conference on UbiCon-trol Laboratory, quitous Robots and Ambient Intelligence(ICURAI), pp.426-437, Seoul, Korea, 2004(12).

[6] Microsoft Robotics Studio 소개, Microsoft

[7] CCR 및 DSS 사용자 가이드, Microsoft

[8] Miro Manual, Ver0.9.4, 2006(1).

[9] Hans Utz, Stefan Sablatnog, Stefan Enderle, Gerhard Kraetzschmar, "Miro-Middleware for Mobile Robot Applications," IEEE Transactions on Robotics and Automation, Vol.18, No.4, 2002(8).

[10] IEEE Standards Project P1003.4, "Draft Standard for Information Technology - Portable Operating System Interface(POSIX) - Part 1: System Application: Realtime Extension [C Language]," Draft13, The Institute of Electrical and Electronics Engineers, 1992.

[11] http://sourceware.org/pthreads-win32/

[12] http://ecos/sourceware.org

[13] E. Bianchi, L. Dozio, G. L. Ghiringhelli, and P. Mantegazza, "Complex Control Systems, Applications of DIAPM RTAI at DIAPM," Realtime Linux Workshop, 1999.

[14] P. Cloutier, P. Mantegazza, S. Papacharalambous, I. Soanes, S. Hughes, and K. Yaghmour, "DIAPM-RTAI Position paper," Real Time Peration Systems Workshop, pp.1-28, 2000.

[15] http://www.xenomai.org

[16] C. S. Douglas, L. L. David, and M. Sumedh, "The Design of the TAO Real-Time Object Request Broker," Computer Communications of Elsivier Science, Vol.21, No.4, 1998(4).

[17] D. G. Christopher, L. L. David, and C. S. Douglas, "The Design and Performance of a Real-Time CORBA Scheduling Service," The International Journal of Time-Critical Computing Systems, 1998(8).

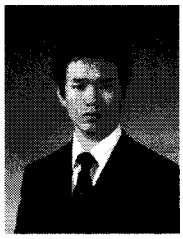
[18] JOHNSON M. HART, Windows 시스템 프로그래밍 3판, 정보문화사, 2008.

[19] MSDN, Scheduling Priorities of Windows, http://msdn.microsoft.com/en-us/library/ms685100(VS.85).aspx

[20] MSDN, Process and Thread Functions, http://msdn.microsoft.com/en-us/library/ms684847(VS.85).aspx

저 자 소 개

최 찬 우(Chan-Woo Choi) 준회원



- 2007년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2007년 3월 ~ 현재 : 충남대학교 컴퓨터 공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 로봇 실시간 프레임워크, 임베디드 시스템

조 문 행(Moon-Haeng Cho) 정회원

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

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

박 성 종(Seong-Jong Park) 준회원

- 2007년 2월 : 충남대학교 컴퓨터공학과(공학사)
- 2007년 3월 ~ 현재 : 충남대학교 컴퓨터 공학과 석사과정 재학

<관심분야> : 실시간 운영체제, 임베디드 시스템, 고장허용 컴퓨팅

이 철 훈(Cheol-Hoon Lee) 정회원

- 1983년 2월 : 서울대학교 전자공학과(공학사)
- 1988년 2월 : 한국과학기술원 전기및전자공학과(공학석사)
- 1992년 2월 : 한국과학기술원 전기및전자공학과(공학박사)
- 1983년 3월 ~ 1986년 2월 : 삼성전자 컴퓨터사업부 연구원
- 1992년 3월 ~ 1994년 2월 : 삼성전자 컴퓨터사업부 선임연구원
- 1994년 2월 ~ 1995년 2월 : Univ. of Michigan 객원 연구원
- 1995년 2월 ~ 현재 : 충남대학교 컴퓨터공학과 교수
- 2004년 2월 ~ 2005년 2월 : Univ. of Michigan 초빙 연구원

<관심분야> : 실시간시스템, 운영체제, 고장허용 컴퓨팅, 로봇 미들웨어