RT-eCos 3.0 기반의 RMMC 및 글로벌 타임 설계 및 구현

(Design & Implementation of the RMMC and Global Time based on the RT-eCos 3.0)

한 승 연 † 김 정 국 ††
(Seoungyeon Han) (Jungguk Kim)

요 약 RT-eCos 3.0은 대표적 분산 실시간 객체 모델인 TMO(Time-triggered Message-triggered Object)의 기본적 태스크 모델 실행을 제공하기 위하여 공개소스 eCos 3.0 기반으로 개발된 초경량 경성 실시간 임베디드 운영체제이다. 본 논문에서는 RT-eCos 3.0에서, TMO 표준 분산 IPC인 RMMC(Real-time Multicast & Memeory replication Channel)를 지원하기 위한 설계 및 구현에 대해 기술한다. 또한 RMMC를 사용하는 분산 노드에서 동일 시각을 사용하기 위한 글로벌 타임의 지원 기술에 대해서도 기술한다. 개발된 글로벌 타임 기반의 RMMC는 RT-eCos 3.0과 함께 광역 분산 컴퓨팅 환경에서 동일 시간 기반으로 추상화된 고급의 분산 IPC 환경을 제공한다.

키워드 : TMO, 분산 IPC, 임베디드, RT-eCos, RMMC, 실시간, 글로벌타임

Abstract RT-eCos 3.0 is a micro-sized embedded real-time kernel that has been developed based on the open source eCos 3.0 to support the basic task model of the well-known distributed real-time object model, TMO(Time-Triggered Message-triggered Object). In this paper, the design and implementation techniques of the RMMC(Real-time Multicast & Memory replication Channel) that is a standard distributed IPC model of the TMO is described based on the RT-eCos 3.0. And the support technique of the global time for using the same time in a distributed environment using the RMMC is also described. The developed global time based RMMC supports highly abstracted distributed IPC environment in a wide area distributed computing environment with the RT-eCos 3.0.

Key words : TMO, Distributed IPC, Embedded, RT-eCos, RMMC, Real-time, Global time

1. 서 론

실시간 시스템은 작업을 수행하는데 있어서 시간 조건에 대해 보장성과 예측성이 주어지거나 시간 조건의 위반에 대해 대응하여 처리할 수 있는 시스템을 뜻하는 것으로, 분산 실시간 객체 모델 TMO에 대한 연구는 90년대 초반부터 진행되었다.

실시간 객체 모델 TMO(Time-triggered Message-triggered Object)[1]는 분산 실시간 시스템 분야에서 최근 수년간 세계적인 새로운 패러다임으로 부각되고 있는 모델로서, U.C.Irvine의 Kane Kim 교수와 Kopetz 교수에 의해 제안되었다. TMO는 정시 보장 실시간 컴퓨팅, 객체 지향, 분산 환경 등의 특징을 통합하여 가지며, 경성 및 연성 실시간 시스템에서 병렬 컴퓨팅을 위한 기능적 및 시간적 수행의 모델로 사용된다.

TMO 모델의 지원 플랫폼은 기존 OS에서 동작하는 미들웨어의 형태로 UCI에서 개발된 TMOSM[2]과, 기존 커널을 수정한 새로운 커널의 형태로 개발된 TMO-Linux[3], TMO-MicroC/OS-II, RT-eCos[4,5] 등이 있다. 본 논문에서는 공개 소스 커널 eCos 3.0을 기반으

· 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다. (UD060048AD)

† 학생회원 : 한국외국어대학교 컴퓨터 및 정보통신공학과
kokolisy@nate.com
†† 종신회원 : 한국외국어대학교 컴퓨터공학과 교수
jgkim@hufs.ac.kr

논문접수 : 2010년 4월 7일
심사완료 : 2010년 4월 26일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제7호(2010.7)

로 TMO 지원을 위해 수정 개발된 RT-eCos 3.0[5]이 사용 되었다.

RT-eCos 3.0은 TMO의 시간 구동 및 메시지 구동 태스크 모델을 지원하기 위해 공개 소스 운영체제인 eCos 3.0을 기반으로 수정 개발된 실시간 임베디드 커널로서 EDF, LLF, FIFO 등의 다양한 데드라인 기반 실시간 스케줄러를 제공하고 TMOSL(TMO Support Library)을 이용한 TMO 기반의 프로그래밍 기능을 제공한다(그림 2).

RMMC(Real-time Multicast & Memory replication Channel)는 UCI TMO 제공 미들웨어의 표준 TMO 분산 IPC에서 제공하는 publisher/subscriber 모델의 추상화된 채널로 TMO 메소드 간의 다양하고 편리한 분산 통신을 지원한다. RMMC는 [6]에서 RMMC의 state message와 event message 중 event message 만을 제공하도록 부분 구현된바 있으나 본 논문에서는 저수준 channel IPC와 RMMC 상위 계층의 계층적 구조로 리모델링하여 이를 전반적으로 모두 구현하였다.

기존의 U.C.I TMO 지원 구조 및 RMMC는 커널 상의 미들웨어 형태로 개발되어 다른 Windows나 Linux 등 다양한 OS환경에서 쉽게 사용할 수 있도록 개발된 반면, RT-eCos 3.0은 TMO 지원과 RMMC는 커널 내부를 수정하여 만들어졌으므로 RMMC에 의한 수신 메소드의 구동에서 보다 정확한 데드라인 스케줄링의 시간 정확성을 가진다. 또한 개발된 RMMC는 2개 계층으로 구성되어 하위 계층만으로도 간단한 분산 IPC가 가능하도록 설계되었다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 실시간 객체 모델 TMO, TMO의 기본 태스크 모델 지원 기능이 탑재된 RT-eCos 3.0에 대한 소개 및 RMMC에 대한 소개, 3장에서는 RT-eCos 기반 RMMC 및 클럭 동기화를 기반으로 하는 글로벌타임(global time)의 설계 및 구현에 대하여 기술한다. 그리고 4장에서는 결론 및 향후 연구 방향에 대해 논의한다.

2. 관련연구

2.1 TMO 모델

TMO(Time-triggered Message-triggered Object)[1]는 객체지향, 실시간 및 분산 컴퓨팅을 한 객체 내에서 모두 제공하는 모델로 보장성 컴퓨팅 패러다임을 지향한다. TMO는 경성 실시간 응용 프로그램뿐만 아니라 병렬 컴퓨팅 응용 프로그램에서도 사용할 수 있는 유연한 구조를 가졌으며 시스템 설계 시의 정시 보장성을 추구한다.

TMO의 특징을 요약하면 다음과 같다.

- TMO는 ODS(Object Data Store)와 이들을 공유하는 메소드인 객체내의 스레드 군으로 구성된다.
- TMO의 메소드는 그 특성에 따라, 사전에 주어지는 시간 조건(ACC: Autonomous Activation Condition)에 의해 구동되는 SpM(Spontaneous Method)들과, 분산 환경에서 다른 클라이언트 TMO가 보내는 메시지에 의해 구동되는 SvM(Service Method)들로 분류된다. SpM은 실행 주기와 수행 데드라인이 주어지며 SvM은 수행 데드라인을 가지며, 두 가지 형태의 메소드는 모두 객체 내의 스레드들이다.
- SpM과 SvM이 객체 내의 공유 데이터에 동시에 접근하여 충돌이 발생할 경우 SpM은 SvM 보다 높은 우선순위를 가지는데, 이것은 설계 시 시간 보장 개념을 도입하기 위해 SpM과 SvM의 시간 선점을 계층화 한 것으로 BCC(Basic Concurrency Constraints)라 한다.

SpM의 작동 주기는 설계 시에 지정 되어야 하는데, 이러한 조건을 AAC(Autonomous Activation Condition)라고 하고, 다음과 같은 형태로 정의할 수 있다.

```
For t = from an 10am to 10:50am every 3sec
Start-during < t-10msec, t+10msec> finish-by t+1 sec;
```

위의 't'는 시간을 의미하며 'For t = from an 10am to 10:50am'은 SpM의 활동 시간의 시작과 끝을 의미한다. 'every 3 sec'은 SpM의 수행 주기가 3초임을 나타낸 것이다. 'Start-during <t-10msec, t+10msec>'은 SpM이 매 주기마다 수행할 때 시작 시간의 오차 허용 한계가 전후 100분의 1초임을 나타내고, 'finish-by t+1 sec'은 매번 수행 시 데드라인이 1초임을 나타낸다.

SvM의 경우에는 주기와 관련된 시간 정보를 필요로 하지 않기 때문에 AAC의 정보 중 주기, 시작 시간, 종료 시간 등의 정보를 생략하고 데드라인만을 사용한다.

다음 그림 1은 위에서 설명한 TMO 모델의 구조를 나타낸 것이다.

그림 1은 TMO 모델의 구조를 보여주는 것으로, TMO는 객체 자료 저장소인 ODS(Object Data Store)와 TMO 내의 행동 양식을 나타내는 SpM(Spontaneous Method)과 SvM(Service Method)의 멤버 스레드를 가

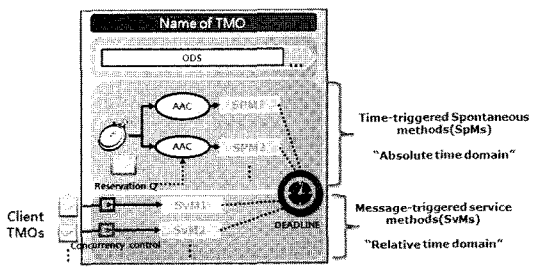

그림 1 TMO 모델의 구조

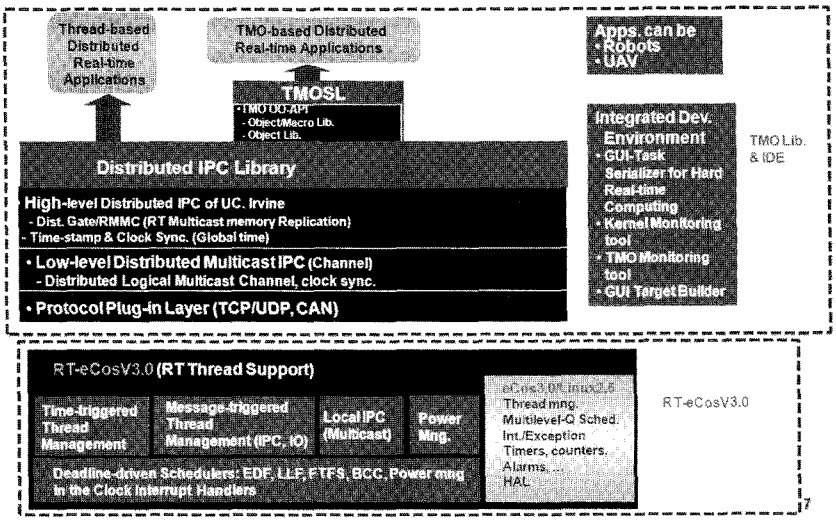

그림 2 RT-eCos3.0 구조

진다. TMO 객체는 여러 개의 SpM과 SvM을 가질 수 있는데, SpM은 시간 조건에 의해 구동되고, SvM은 분산 환경에서 클라이언트가 보내오는 event 메시지, 즉 IPC에 의해서 구동되는 메소드이다. 각 형태의 메소드는 구동되면 데드라인 스케줄링의 적용을 받는다.

2.2 RT-eCos 3.0

RT-eCos 3.0은 TMO의 시간 구동 및 메시지 구동 태스크 모델을 지원하기 위해 공개 소스 운영체제인 eCos 3.0을 기반으로 수정 개발된 실시간 임베디드 커널이다. RT-eCos 3.0은 TMO의 태스크 모델 지원을 위해 다음과 같은 기능을 제공한다.

- CPU의 성능 및 클럭 구성에 따라 최저 30us에서 10ms까지의 정밀도를 갖는 시간 구동 스레드(SpM : Spontaneous Method)와 메시지 구동 스레드(SvM : Service Method)의 마감시간 구동 실시간 스케줄링 기능을 제공한다.
- 논리적인 멀티캐스트, IPC 서브시스템 및 로컬 메시지를 통한 SvM의 구동 기능을 제공한다.
- EDF, LLF, FIFO 등의 다양한 데드라인 기반 실시간 스케줄러를 제공한다.
- TMOSL(TMO Support Library)을 이용한 TMO C++ 객체 기반의 프로그래밍 기능을 제공한다.

그림 2는 RT-eCos3.0의 구조를 나타낸 것으로 기반 커널 부분에는 시간 및 메시지 구동 메소드의 처리와 이를 위한 국부 IPC 및 데드라인 스케줄러가 포함되고 상위 계층에서는 분산 IPC의 각 계층과 모니터링 도구 등 개발 환경요소가 포함되어 있다.

2.3 RMMC(Real-time Multicast & Memory replication Channel)

RMMC는 UCI의 TMO 미들웨어에서 제공하는 추상화된 개념의 표준 분산 IPC이다. RMMC는 여러 노드간의 TMO 메소드간 글로벌 타임 기반 통신을 제공한다. RMMC의 특징과 제공하는 기능은 다음과 같다.

- RMMC는 개념적인 채널을 의미하며 RMMC를 사용하기 위해서 RMMCGate 객체를 생성해야 한다.
- RMMCGate를 통해서 보낼 수 있는 메시지의 종류는 FIFO 메시지 전달 방식의 event 메시지와 공유메모리 성격인 Last-is-Best 방식의 state 메시지의 두 가지가 있다. 이 중에서 state 메시지는 같은 RMMC를 사용하는 분산 노드의 저장소에 있는 내용을 최근 내용으로 동시에 업데이트 시키는 실시간 데이터 저장소이다. state 메시지는 변화하는 동적인 상태 정보를 관찰하는데 사용되는 분산된 중복 메모리 기반 메시지를 뜻한다. event 메시지의 경우는 메시지가 큐잉되는 방식이며, 일반적인 Gate 객체에서 사용하는 메시지를 주고받는 방식과 비슷하다.
- RMMC에는 메시지 배달의 정시성을 보장하기 위하여 ORT(Official Release Time)가 적용되었다. ORT는 노드가 메시지를 미리 수신하여도 ORT로 정해진 시간에 배달을 하는 것으로서 메시지 배달의 예측성 증가에 의한 실시간성 향상에 기여하는 개념이다. ORT를 적용하는 예는 다음의 두 가지가 있다.
- RMMCGate 객체의 state 메시지와 event 메시지를 보낼 때 메시지에 ORT를 설정하면 메시지를 수신하는 TMO 시스템 내부에서 관리되고, 정해진 시간에 메시지를 전송하게 된다.
- RMMCGate의 state 메시지와 event 메시지를 받는 메소드는 메시지를 받는 시간을 ORT로 설정할 수 있다.

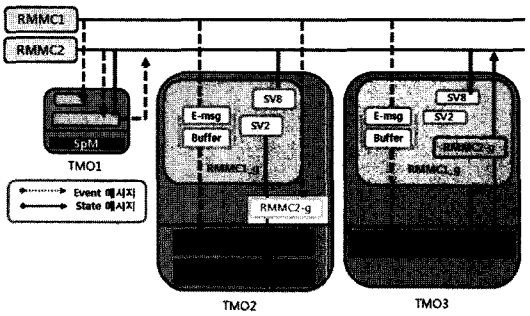

그림 3 RMMCGate 동작

ORT의 설계 및 구현은 3.2.5절에서 상세히 기술한다.

위의 그림 3은 RMMC의 state 메시지와 event 메시지의 동작과정을 도시한 것이다.

그림 3에서와 같이 RMMCGate 객체를 이용하면 같은 RMMC를 사용하는 모든 노드에 메시지를 네크워크에 투명하게 전달할 수 있다. 전달할 수 있는 메시지의 종류는 큐잉 방식의 event 메시지와 Last-is-Best 방식의 state 메시지가 있다.

3. RMMC 설계 및 구현

RMMC는 RT-eCos 3.0에서 제공하는 하위계층 채널 기반 분산 IPC와 상위 RMMC 계층의 2개 계층 구조로 설계 구현되었다. 다음은 계층별 구현에 대한 설명이다.

3.1 RT-eCos 3.0 TMO 커널의 하위 계층 분산 IPC 모델 구현

RT-eCos 3.0과 같은 TMO 기본 엔진은 분산 IPC 도구로 논리적으로 멀티캐스팅 되는 채널을 제공한다. 이는 하나의 TMO가 특정 채널로 메시지를 송신할 경우, 이 채널에 대기하고 있는 모든 TMO는 해당 메시지를 수신할 수 있음을 의미한다. 채널 식별자는 분산 시스템 상의 모든 노드들에게 있어 동일하게 적용되는 값이며, 각 TMO가 원격 TMO들에게 메시지를 송신할 때 그 위치에 대한 정보 없이 채널만을 이용한다. 분산 IPC 환경에서 여러 TMO들이 서로 메시지를 주고받기 위해서는 각각의 객체들이 사용할 채널을 할당하는 것이 필요하며, 메시지를 주고받을 TMO들은 같은 채널을 사용해야 한다. 그림 4는 TMO 기반 엔진의 분산 IPC 모델을 나타낸다.

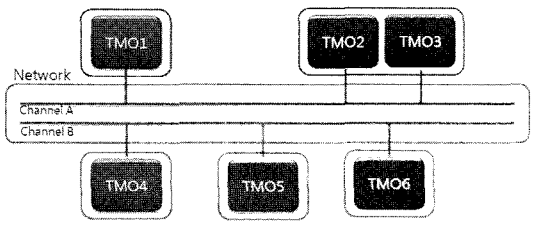

그림 4 TMO엔진의 분산 IPC 모델

채널 A를 할당 받은 TMO1, TMO2, TMO3의 TMO 객체는 채널 A를 통해 같은 메시지를 받을 수 있으며, 한 객체는 다른 모든 객체에게 메시지를 보낼 수 있다. TMO 객체 2와 3은 같은 노드에 있는 다른 객체로써 각 객체 별로 채널을 할당 받기 때문에 같은 노드에 있는 객체 일지라도 서로 같은 채널을 할당 받을 수도, 다른 채널을 할당 받을 수도 있음을 보여준다.

각 TMO는 채널을 읽기, 쓰기, 읽기/쓰기 모드로 사용할 수 있고, TMO 엔진에서 분산 IPC는 로컬이나 혹은 원격에 있는 TMO에게 채널을 통해 메시지를 전달할 수 있으며, 읽기 모드로 채널을 사용하고 있는 TMO는 메시지를 받아 대기 중인 SvM을 동작시킨다.

이러한 하위 계층 채널 기반 분산 IPC의 구동을 위해 각 노드에는 메시지를 수신 배포하는 IMMT(Incoming Message Management Task)와 채널별 자료구조가 다음과 같이 설계되었다.

IMMT는 내부에 외부 노드로부터 수신되는 TCP 메시지를 수신하여 event 메시지는 채널 별 메시지 큐인 IMQ(Incoming Message Queue)에 저장하고 채널에서 대기하고 있는 SvM을 깨워주는 역할을 한다. State 메시지는 채널별 공용 저장 버퍼인 Channel_Data_Message에 저장되고 read_data_message 함수에 의해 저장된 메시지를 읽어 올 수 있다. 이때 IMQ의 각 셀과 Channel_Data_Message 버퍼의 크기는 alloc_channel 함수에서 주어진 event 메시지 크기와 state 메시지 크기 인자 값으로 설정한다. 다음 그림 5는 IMMT의 동작과정을 나타낸 것이다.

그림 5에서 IMMT는 수신된 메시지의 종류를 판단하여, 각 메시지 종류에 따른 행동을 하도록 구현되었다.

하위계층 분산 IPC에서 제공하는 인터페이스를 간략히 소개하면 다음과 같다.

- alloc_channel(channelId, event message size, data message size, read-write flag): 분산 IPC 채널을

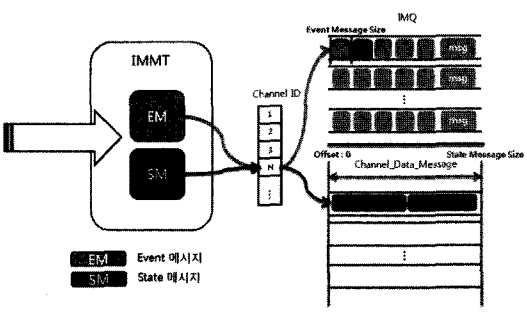

그림 5 IMMT 동작 과정

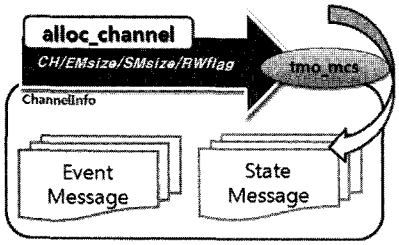

그림 6 alloc_channel 동작 과정

할당한다(그림 6).

- delloc_channel(channelId): 할당된 채널을 해제한다.

하위계층의 event 메시지와 관련된 함수는 다음과 같다.

- send_dist_event_message(Message, Channel, ORT)
- send_event_message(Message, Channel, ORT)
- mt_thread_wait_invocation(Channel, Message, bool_blocked, ORT)

send_event_message 함수는 할당된 분산 IPC 채널을 통해 국부 함수를 이용해 event 메시지를 시스템 내에 채널 별로 설정된 IMQ에 순서대로 저장한다. send_dist_event_message 함수는 내부에 send_event_message 함수를 포함하며, 결국 국부 및 원격 TMO들에게 event 메시지를 송출하고 결과적으로 해당 채널에서 대기하는 SvM을 구동시키게 된다. 메시지를 수신 받은 원격 TMO에서는 IMMT가 메시지를 분석하여 시스템 내부의 send_event_message 함수를 호출하게 된다. 메시지 수신에 의해 SvM이 구동되면 mt_thread_wait_invocation 함수가 호출될 때 까지 데드라인 기반 스케줄링이 적용된다. 다음 그림 7은 event 메시지 관련 인터페이스의 동작과정을 나타낸 것이다.

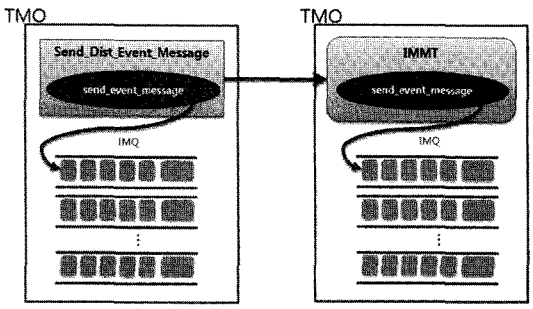

그림 7 Event 메시지 관련 인터페이스 동작과정

하위 계층의 state 메시지와 관련된 함수는 다음과 같다.

- write_dist_data_message(Message, Channel, ORT)
- write_data_message(Message, Channel, ORT)
- read_data_message(Offset, Length, Data, Channel)

write_event_message 함수는 할당된 분산 IPC 채널을 통해 State 메시지를 국부 시스템 내에 채널 별로 설정된 Channel_Data_Message에 중복된 값으로 저장한다. 즉, 가장 마지막에 전송된 state 메시지가 저장 되게 된다. write_dist_data_message 함수는 내부적으로 write_data_message 함수를 포함하며, 결국 국부 및 원격 TMO들에게 state 메시지를 전송하게 된다. 저장된 메시지는 read_data_message 함수의 호출에 의해 해당 채널의 Channel_Data_Message에 가장 최근에 저장된 메시지 값을 읽어올 수 있다. 다음 그림 8은 state 메시지 관련 인터페이스 동작과정이다.

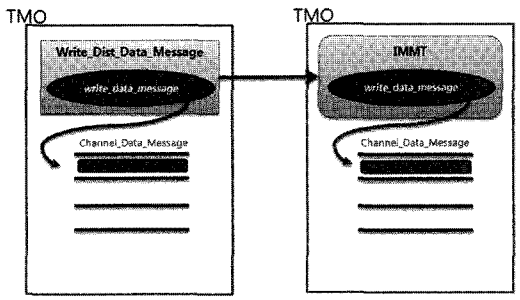

그림 8 State 메시지 관련 인터페이스 동작과정

3.2 상위 계층 RMMC의 설계 및 구현

RMMC는 RT-eCos 3.0 엔진에 구현된 하위 계층 채널 기반 분산 IPC를 최대한 이용하여 상위계층 라이브러리로 구현하였으며 사용자는 eCos에서 제공하는 GUI configtool을 이용하여 RMMC를 이용한 응용을 작성할 수 있도록 eCos configtool을 수정하였다(그림 9).

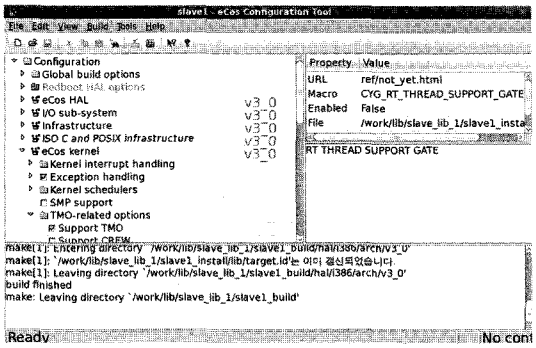

그림 9 eCos configtool

RT-eCos3.0 하위 계층 위에 상위 계층 라이브러리로 추가 구현된 기능은 다음과 같다.

- Gate와 RMMC의 자동 하위 계층 채널 할당
- RT-eCos의 하위 계층 채널과 분산 IPC를 이용한 TMO 표준 분산 IPC 라이브러리의 RMMCGate 클래스

• 메시지의 정시성을 보장하기 위해 제안된 ORT의 구현
• 분산 노드 간의 클럭 동기화에 의한 글로벌 타임의 제공

3.2.1 RMMC의 자동 채널 할당

하위계층 분산 IPC에서는 사용자가 직접 채널을 할당하여야 했다. 그러나 RMMC에서는 RMMCGate 객체를 사용하므로 분산 환경의 각 노드에서 자동으로 각기 다른 채널이 할당되고 이들에 대한 맵핑이 일어나야 한다. 이를 위하여 각 노드별 객체-채널 테이블을 정의하여 채널 맵핑이 수행되도록 하였다.

그림 10은 TMO 표준 인터페이스의 RMMC가 채널 테이블을 참조하여 채널을 요청하고 할당 받는 과정을 도시한 것이다.

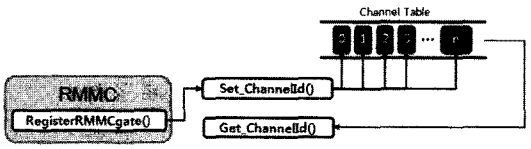

그림 10 자동 채널 할당 과정

그림 10과 같이 각각의 각 노드별 TMO 시스템은 각각 하나씩의 로컬 채널 테이블 리스트를 가지게 된다. 할당된 채널은 RMMCGate 객체를 이용하여 메시지를 보낼 때, Global TMO 이름과 Gate이름을 포함하여 전송하게 되고, 메시지를 수신한 TMO 시스템에서는 Global TMO의 이름으로 온 메시지를 RMMCGate를 통해서 수신했다고 판별하고 Gate이름으로 자신의 채널 테이블을 검색하여 들어온 메시지의 수신 여부를 결정하게 된다.

3.2.2 RMMCGate 클래스의 구현

RMMC는 개념적인 분산 공유 저장소와 메시지를 송수신하는 개념적인 채널이고 이에 접근하기 위해서는 RMMCGate 클래스가 생성되어야 한다. 그림 11은 RMMCGate의 생성과 사용에 관련된 RT-eCos의 하위계층 분산 API와 RMMC의 상관관계와 자동 채널 할당 과정을 계층 별로 도시한 것이다.

그림 11에서 DistIPC 계층까지는 RT-eCos의 하위계층 API들이다. TMOSL 계층과 DistIPC 계층 사이에 채널 계층을 두고 이 계층에서 채널을 자동으로 할당받도록 설계하였다. 사용자는 생성된 RMMCGate 객체의 g_Update()와 Announce() 함수를 통해 다른 TMO 시스템에게 state 메시지와 event 메시지를 전달한다.

3.2.3 Event 메시지의 처리

Event 메시지는 사용자가 만든 RMMCGate 객체의 Announe() 함수를 사용하여 다른 TMO 시스템에게 event 메시지를 전달한다. 이 때 사용되는 하위계층 분

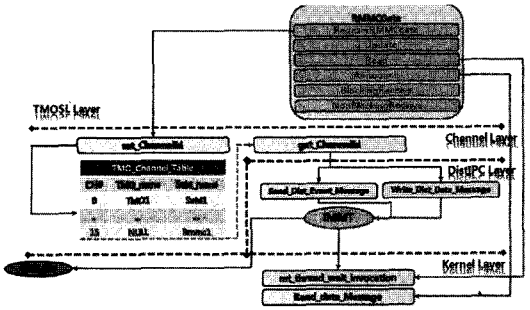

그림 11 RMMCGate 클래스

산 IPC API는 다른 노드의 TMO 시스템에게 메시지를 전달하는 send_dist_event_message() 함수이며, 메시지를 수신하는 TMO 시스템에서는 IMMT(Incoming Message Management Task)에서 메시지를 수신하여 TMO 시스템 내의 채널별 메시지 큐인 IMQ 에 메시지를 저장하고 SvM은 기존 RT-eCos의 커널 API 중 메시지가 들어오기를 기다릴 때 사용하는 mt_thread_wait_invocation 함수에 의해 메시지를 받게 된다. 다음 그림 12는 RMMCGate에서 제공하는 Event 메시지 관련 API와 RT-eCos 커널 API와의 상관관계 및 메시지 전달 과정을 레이어 별로 도시한 것이다.

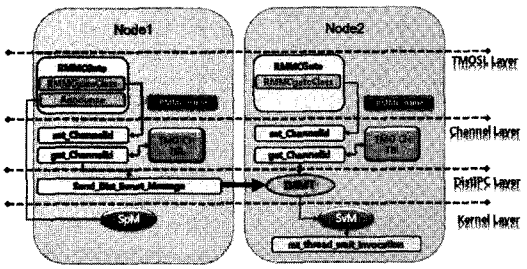

그림 12 RMMCGate 사용 시 event 메시지 전달 과정

그림 12와 같이 RMMCGate의 Announce 함수를 사용하여 event 메시지를 전달하게 되고, 이렇게 보내진 event 메시지는 같은 RMMC를 사용하는 모든 노드에 전달된다.

3.2.4 State 메시지의 처리

State 메시지는 고유한 이름을 가지게 되며 같은 RMMC를 사용하는 각 TMO 시스템에 있는 state 메시지 리스트인 SMV 리스트에 각 state 메시지의 이름과 크기 값의 정보가 리스트에 저장된다. State 메시지는 채널 별 Channel_Data_Message 버퍼에 저장되고, 새로운 state 메시지가 수신되었을 때 state 메시지 이름을 검사하여 저장되어 있는 state 메시지와 같은 이름이면 내용이 갱신된다.

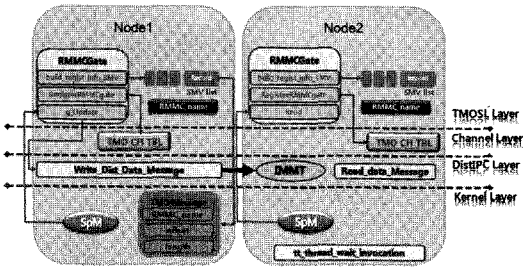

그림 13 RMMCGate의 State 메시지 전달 과정

그림 13은 RMMCGate에서 제공하는 state 메시지 관련 API와 하위 계층 API와의 상관관계 및 메시지 전달 과정을 계층별로 도시한 것이다.

위의 그림 13과 같이 RMMCGate의 g_Update() (Global Update) 함수를 사용하여 state 메시지를 전달하게 되고, 메시지는 같은 RMMC를 사용하는 모든 노드의 Channel_Data_Message에 메시지가 업데이트 된다. 이 때 호출되는 하위 계층 API는 write_dist_data_message 함수이다. 메시지를 수신하는 노드에서는 RMMCGate의 read 함수를 이용하여 원하는 메시지를 읽어오게 된다. 이 때 호출되는 하위 계층 분산 IPC API는 read_data_message 함수이다.

3.2.5 ORT의 설계와 구현

ORT(Official Release Time)은 event 메시지와 state 메시지가 사용되는 메소드에게 정시성을 보장해 주기위한 일종의 약속 시간으로, 메시지 배달 시에 설정되어 보내진다. ORT가 적용된 메시지는 메시지를 수신하는 TMO 시스템의 IMMT에 의해 ORT_Queue에 저장된다. ORT_Queue에 저장된 메시지는 정해진 시간까지 유보하고 추후에 시간에 되면 배달하여야 하므로 RT-eCos 커널 내부의 실시간 스케줄러에서 관리되도록 설계 되었다. 그림 14는 ORT가 적용된 메시지의 처리 과정을 도시한 것이다.

그림 15는 5초 주기의 SpM에 1000 tick의 ORT가 주어졌을 때, ORT_Queue에서 메시지가 추가되는 방법과 추가된 메시지들의 처리 방식을 도시한 것이다.

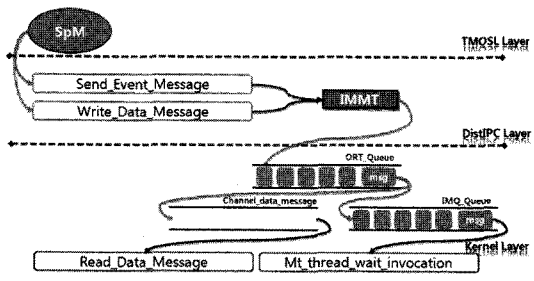

그림 14 ORT 적용 메시지의 처리 과정

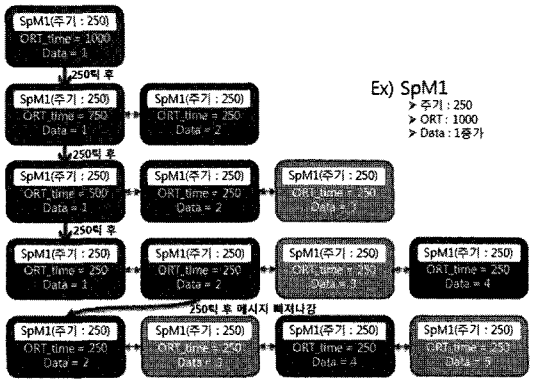

그림 15 ORT_Q에서 ORT 메시지 처리 동작

그림 15에서 250 tick의 주기로 전송되는 메시지에 ORT 값이 1000 tick이 되어 있을 때, 이 메시지가 IMMT에 의해 ORT_Queue에 들어오면, 매 주기마다 ORT 값을 계속 줄여나가면서 ORT 값이 소진되는 순간에 메시지를 ORT_Queue에서 내보내게 된다. 결국 메시지는 ORT가 정의된 시간만큼 ORT_Queue에서 대기 상태로 있게 된다.

위의 예제와 같이 ORT가 설정된 메시지가 ORT_Queue에 들어갈 때 현재 남아있는 ORT를 비교하여 delta-Q 방식으로 구현하여 큐의 가장 첫 메시지의 ORT만 체크해도 관리가 가능하도록 하였다. ORT가 모두 소진된 메시지는 대기하고 있는 SvM이 있을 경우에는 대기하고 있는 SvM의 메시지로 추가되고, 대기하고 있는 SvM이 없거나 RMMCGate의 event 메시지일 경우에는 해당 채널의 IMQ_Queue에 추가되고, state 메시지일 경우에는 Channel_Data_Message 에 추가된다.

3.2.6 RT-eCos 3.0 동적 분산 시스템에서의 글로벌 타임의 구현

글로벌 타임이란 분산 시스템에서 클럭 동기화에 의해 같은 동일 동시의 클럭 틱(tick) 행위를 하게하고, 이를 기반으로 분산 컴퓨팅 참여 노드가 동일한 시간 기반으로 서비스 행위나 메시지의 타임스템프 등을 갖게 하는 것을 말한다. 분산 노드 간의 동일한 시간, 즉, 글로벌 타임은 epoch time(1970.1.1.0시 기준)이나 기타의 기준을 사용한 마이크로 초 단위의 64비트 정수로 표시되는 것이 보통이다. 분산 노드 간의 메시지 통신이나 서비스 요구 시의 글로벌 타임의 구현을 위해서는 일단 각 노드의 tick 간격을 보정하는 클럭 동기화와 이를 기반으로 기준 노드 간 동일한 글로벌 타임을 제공하는 체제가 필요한데, RT-eCos3.0에서는 이를 위해 시간 서버의 역할을 하는 마스터 노드와 나머지 슬레이브 노드들로 분산 시스템을 구현하였다.

• 마스터 슬레이브 노드 간의 글로벌 타임

각 분산 노드는 노드별로 일정치 않은 부팅 시작 시간과 부팅 시간을 가지므로 마스터 노드와의 참여 동기화 과정을 통해 분산 시스템에 참여하게 된다. 여기서 마스터 노드의 부팅 완료 즉, 시작 시간을 상대적인 글로벌 타임으로 생각한다. 마스터 노드가 배터리 백업 월 클럭이나 GPS를 갖고 있는 경우는 부팅후의 실제 시간을 글로벌 타임으로 결정한다. 다음 그림 16은 이러한 상황에서 두 개의 슬레이브 노드의 참여 시나리오의 타이밍을 도시한 것이다.

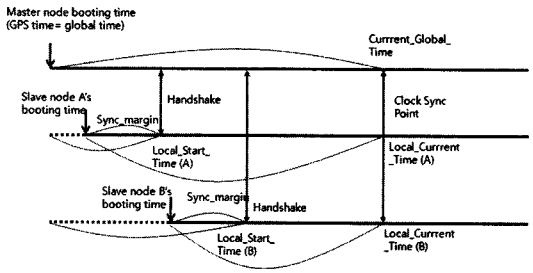

그림 16 클럭 동기화 과정

그림 16에서 노드 A와 노드 B는 자유로운 시간에 부팅되고 동적으로 분산 시스템에 참여하게 된다. 분산 시스템에 참여시에는 Handshake를 호출하여 마스터 노드로부터 글로벌 타임 기준으로 Local_Start_Time을 받아 저장한다.

부팅 후 마스터 노드와 동기화가 될 때까지의 시간을 sync_margin이라 한다. 각 노드는 어떤 시간 위치에서 자신의 클럭 틱만을 알고 있기 때문에, 이 시각의 글로벌타임은 다음 식을 통해 계산된다.

global_time = Local_Start_Time +
Local_Current_Time - Sync_margin

위 식에 의해 계산된 글로벌타임은 모든 노드가 동일해야 하며 분산 IPC에서 메시지 전송 시의 타임스탬프나 서비스 요구 시간 지정 등은 위의 글로벌타임을 사용하게 된다.

• 클럭 동기화

글로벌 타임을 동일하게 유지하기위해서는 주기적인 클럭 보정이 있어야 한다. 클럭 동기화 알고리즘은 여러 가지가 있으나 RT-eCos3.0에서는 마스터 노드가 GPS를 갖고 있다고 가정하므로, 타임 서버를 둔 클럭 동기화를 수행한다. LAN 환경에서 마스터 노드는 주기적으로 UDP를 통해 글로벌 타임을 브로드캐스팅하며 이를 수신한 슬레이브 노드는 자신의 글로벌 타임을 계산하여 그 차이를 clock-gain 변수에 저장한다. clock-gain 계산 시에는 UDP의 평균 전송 지연도 고려한다. 계산된 clock-gain이 양수이면(즉 슬레이브의 시간이 느린 경우) clock-gain이 만회될 때까지 클럭 인터럽트에서의 lost-clock-tick을 한 tick 씩 증가 시켜 주는 방식으로 점진적으로 clock-gain을 만회해 나아간다. 음수일 때에도 역시 점진적으로 1 tick 씩 처리를 스킵하는 방식을 사용한다.

3.2.7 RMMC의 통신 실험

본 논문에서 개발된 RMMC가 RT-eCos3.0 상에서 정상적인 IPC를 지원됨을 확인하고, 역시 개발된 클럭 동기화가 정상적으로 동작되는지 확인 하기위해서 실험을 진행하였다.

실험 환경은 RT-eCos3.0 configtool로 RT-eCos3.0의 옵션을 설정해주기 위해 리눅스 2.6 커널을 가진 Fedora 10을 사용하였고. 분산 노드 통신 실험을 위해 VmWare 6.0 프로그램을 이용한 가상 머신을 사용하였다. Intel(R) Core(TM) i7 CPU와 4G RAM을 장착한 데스크톱 머신환경에서 실험이 진행되었다.

실험은 다음과 같이 진행되었다. RT-eCos3.0 configtool에서 본 논문에서 개발된 RMMCGate와 클럭 동기화 옵션이 적용 되도록 설정한 다음, 설정된 RT-eCos3.0를 2개의 가상 머신에 설치한다. 각각의 머신은 마스터-슬레이브 관계를 가지도록 하였다. 마스터 노드의 테스트 프로그램에서 RMMCGate를 통해 5초 주기로 event 메시지를 전송하고 전송 시간을 기록한다. 슬레이브 노드는 전송된 event 메시지가 수신 되었을 때의 시간을 기록한다. 메시지는 네트워크 상황과 시스템 상황 등에 따라 자연스럽게 지연되어 보내지는데, 이 때 클럭 동기화 옵션을 지정하게 되면 송신 시간과 수신 시간이 글로벌타임 값으로 나오게 된다. 만약 네트워크 지연이 전혀 없는 상황이라면 마스터 노드의 송신시간과 슬레이브 노드의 수신 시간은 동일한 시간을 나타나게 된다. 실험은 네트워크 지연이 발생하는 상황에서 RMMCGate를 이용한 IPC가 이뤄질 때, 마스터와 슬레이브 노드의 글로벌타임 값을 서로 비교하여 시간의 차이를 네트워크 지연으로 판단하여, 이 네트워크 지연이 시스템에 영향을 주지 않는 범위 내에서 발생하는지 확인 한 것이다.

다음 그림 17은 300초 동안 분산 노드에서 RMMC-Gate를 이용하여 SpM이 5초 주기로 event 메시지를 SvM으로 전송하는 경우에 클럭 동기화를 지원한 송수신 노드에서의 측정한 글로벌타임 값을 비교하여 얻은 RMMC 사용 시의 네트워크 지연(delay) 값의 변화를 관찰한 그래프이다.

그림 17에서 네트워크의 상황이나 시스템성능에 따라 나타나는 분산 노드들 간에 발생되는 글로벌타임 값의 차이가 최대 +10 msec(millisecond) 정도의 값을 나타냄을 확인할 수 있다. 이는 본 논문에서 개발된 RMMC-

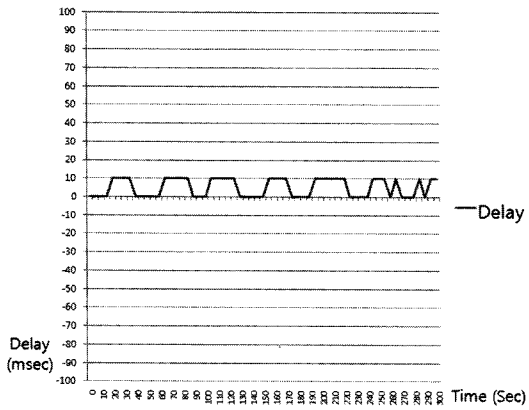

그림 17 네트워크 지연(Delay) 측정

Gate가 TMO 분산 노드에서 정상적인 IPC를 지원됨을 확인하고, 본 논문에서 개발된 클럭 동기화를 통해 마스터-슬레이브 간의 글로벌타임 값이 시스템 오차 허용범위 이내에서 정확하게 유지되고 있음을 확인한 것이다.

4. 결론 및 향후 과제

본 논문에서는 UCI의 TMO 표준 분산 IPC인 RMMC를 RT_eCos 3.0에서 제공하기위해 2개의 계층 구조를 가진 분산 IPC 시스템을 설계 구현하였다. 하위 계층은 channel 기반 분산 IPC 시스템으로 그대로 사용할 수 있으며 추상화 channel 객체에 의한 통신이 편리한 경우에는 상위계층까지 사용할 수 있도록 계층을 분리하여 설계 구현하였다.

개발된 TMO 상위 계층 라이브러리는 1) 기존 RT-eCos의 커널의 스레드 생성, 관리와 관련된 API 를 사용하여 손쉽게 스레드의 생성, 관리가 가능하도록 구현하였으며, 2) 하위 계층의 채널 개념을 TMO 표준 분산 IPC 라이브러리의 RMMCGate로 사상하여 사용자로 하여금 조금 더 직관적이고 손쉽게 응용 프로그램을 작성할 수 있도록 하였으며, 3) 메시지의 정시성을 보장하기 위하여 제안된 ORT 를 RT-eCos 의 커널 내부에 삽입함으로써 보장성 컴퓨팅을 조금 더 용이하게 할 수 있도록 하였다. 기존의 U.C.I TMO 지원 구조 및 RMMC는 커널 상의 미들웨어 형태로 개발되어 다른 Windows나 Linux 등 다양한 OS환경에서 쉽게 사용할 수 있도록 개발된 반면, RT-eCos 3.0은 TMO 지원과 RMMC는 커널 내부를 수정하여 만들어 졌으므로 RMMC에 의한 수신 메소드의 구동에서 보다 정확한 데드라인 스케줄링의 시간 정확성을 가진다. 또한 개발된 RMMC는 2개 계층으로 구성되어 하위 계층만으로도 간단한 분산 IPC가 가능하도록 설계되었다.

향후 연구 과제로는 현재 RT_eCos 3.0에 국한된 TMO RMMC 라이브러리를 TMO-Linux 등 다른 커널에도 이식하는 것이다.

참 고 문 헌

[1] K. H. Kim and H. Kopetz, "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials," *Proc. 18th IEEE Computer Software and Applications Conference*, pp.392-402, November 1994.

[2] K. H. Kim, "Object-oriented real-time distributed programming and support middleware," In *Proceedings of the IEEE 7th International Conference (ICPADS2000)*, pp.10-20, 2000.

[3] S. H. Park, "TMO-Linux : A Linux-Based Real-Time Operating System Supporting Execution of TMOs," Computer Engineering Master's Thesis, Hankuk University of Foreign Studies, June 2005.

[4] Kim, J. G., et al, "TMO-eCos: An eCos-based Real-time Micro Operating system Supporting Execution of a TMO Structured Program," *8th IEEE International Symposium on Object/ Component/ Service-Oriented Real-Time Distributed Computing*, pp.182-189. IEEE Computer Society Press, Seattle, 2005.

[5] Kim, J. H., Kim H. J., Park, J. H., Ju, H. T., Lee, B. E., Kim, S. G., Heu, S, "TMO-eCos2.0 and its Development Environment for Timeliness Guaranteed Computing," *1st Software Technologies for Dependable Distributed Systems*, pp.164-168. IEEE Computer Society Press, Tokyo, 2009.

[6] C. H. Woo, and J. G. Kim, "Design & Implementation of the Standard Interface & ORT on a TMO kernel," *Proc., KCC 2007*, vol.34, no.1(B), 2007.

한 승 연

2009년 2월 한국외국어대학교 정보통신공학과 졸업. 2009년 3월~현재 한국외국어대학교 컴퓨터 및 정보통신공학과 석사과정. 관심분야는 실시간 운영체제, 네트워크

김 정 국

1977년 2월 서울대학교 계산통계학과(이학사). 1979년 2월 KAIST 전산학과(이학석사). 1986년 2월 KAIST 전산학과(공학박사). 1983년 3월~현재 한국외국어대학교 컴퓨터공학과 교수. 관심분야는 분산 실시간 컴퓨팅