리눅스 환경에서의 실시간 객체모델 수행 플랫폼의 설계 및 구현

조수형°, 조상영, 김정국
한국외국어대학교 컴퓨터공학과
shcho@san.hufs.ac.kr

Design and Implementation of Real-Time Object Model Execution Platform on Linux System

Soo-Hyung Cho°, Sang-Young Cho, Jungguk Kim
Dept. of Computer Science and Engineering, Hankuk University of Foreign Studies

요약

정시 서비스를 제공하는 실시간 시스템을 설계할 때 일반성을 지니도록 하기 위하여 실시간 객체인 TMO(Time-triggered Message-triggered Object) 모델이 연구되었으며 이러한 객체를 수행하기 위한 객체모델 수행 플랫폼이 다양한 환경에서 개발되었다. 본 논문에서는 최근들어 많은 주목을 받고 있는 리눅스 환경에서 설계 및 구현된 미들웨어 형태의 실시간 객체모델 수행 플랫폼 LTMOS(Linux TMO System)에 대해 기술한다. 응용 프로그램 개발자가 실시간 시스템을 TMO 모델로 디자인하면 LTMOS 는 TMO 의 메쏘드를 리눅스의 쓰레드로 매핑하여 리눅스 커널의 실시간 스케줄링을 받도록 한다.

1. 서론

갈수록 증가하고 있는 화상 회의, VOD(Video On Demand), 게임과 같은 각종 멀티미디어 서비스나 미사일 방어 시스템, 원자력 발전소 제어 시스템 및 선박 항해 시스템 등의 다양한 분야에서 시간적 제한이 따르는 실시간 시스템을 요구한다. 실시간 작업을 시뮬레이션 하기 위해서 각각의 환경에 맞는 독자적인 시스템 보다는 일반성을 지닌 시스템을 개발할 필요가 있으며 일반성을 지니도록 하기 위해서는 시스템 설계자가 시스템을 설계할 때 정시 보장 서비스를 제공해야 한다[2]. 또 매우 크고 복잡한 시스템의 설계를 위해서는 시스템이나 응용 프로그램 환경에 대해 일정하면서 통합적인 표현 방법이 필요하다.[3] 이러한 요구로 인해 실시간 객체인 TMO 모델이 연구되어져 왔으며 다양한 환경에서 객체모델 수행 플랫폼에 대한 연구가 이루어 지고 있다[1][2]. TMO 객체모델 수행 엔진인 DREAM(Distributed Real-time Ever Available Microcomputing) 커널이 UCI 의 DREAM Lab.에서 구현되었으며 이는 문자 기반의 경성 실시간 시스템으로 설계되었다[2]. 반면에 TMO 객체모델 수행 엔진인 WTMOS(Windows TMO System)는 Win32 환경에서 미들웨어 플랫폼 형태로 구현되었으며 이는 그래픽 기반의 연성 실시간 시스템으로 설계되었다[1].

최근 들어 많은 주목을 받고 있는 유닉스 클론인 리눅스 시스템 또한 실시간 작업에 대한 스케줄링과 고해상도 타이머, 멀티 쓰레드, 분산 IPC 등에 대해 지원하고 있고 실시간 분야에서 '실시간-리눅스'와 같은 연구가 많이 이루어 지고 있다[5].

본 논문에서는 이러한 리눅스 환경에서 TMO 객체모델 수행 플랫폼의 설계 및 구현 방법에 대해 제시한다. 본 논문의 구성은 2 장에서 TMO 모델에 대해 살펴보고 3 장에서는 LTMOS 의 설계 및 구현에 대해 기술하며, 마지막 5 장에서는 결론을 내리고 LTMOS 의 연구 방향에 대해 제시한다.

2. TMO 모델

실시간 객체 모델 형태인 TMO 는 연성 실시간 및 경성 실시간 시스템에서 주기적인 시간 및 임의의 메시지에 대해 반응하는 객체이며 TMOS 는 이러한 객체들을 실시간 스케줄링하고 분산 IPC 와 클럭 동기화를 지원하는 플랫폼이다. TMO 는 경성/연성 실시간 응용뿐만 아니라 일반적인 분산 병행 프로그램 응용에도 사용할 수 있는 객체 모델이다[1].

TMO 의 특징은 다음과 같다.

- 설계 시 시간 보장 개념을 제공한다.
- TMO 의 메소드는 특징적인 두 가지 부류로 나누어 지는데, 하나는 시간 조건에 의해 동작하는 SpM (Spontaneous Method)이고, 다른 하나는 분산 IPC 메시지에 의해 동작하는 SvM (Service Method)이다.
- SpM 과 SvM 이 객체내의 Object Data Store Segment 를 동시에 접근할 때의 상호 배재를 위해 SpM 이 SvM 보다 높은 우선순위를 가지는데 이것을 BCC (Basic Concurrency

Constraints)이라 한다.
그림 1 은 TMO 모델을 나타낸다.

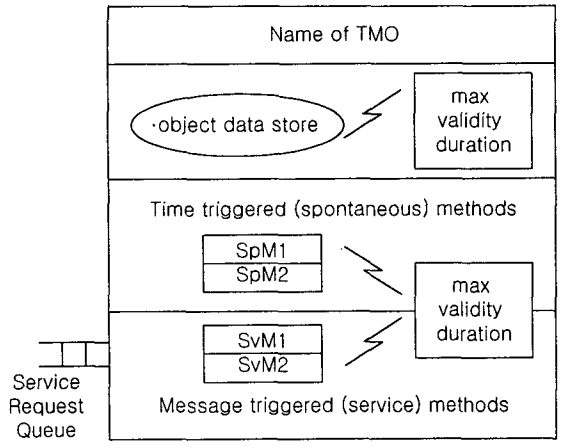
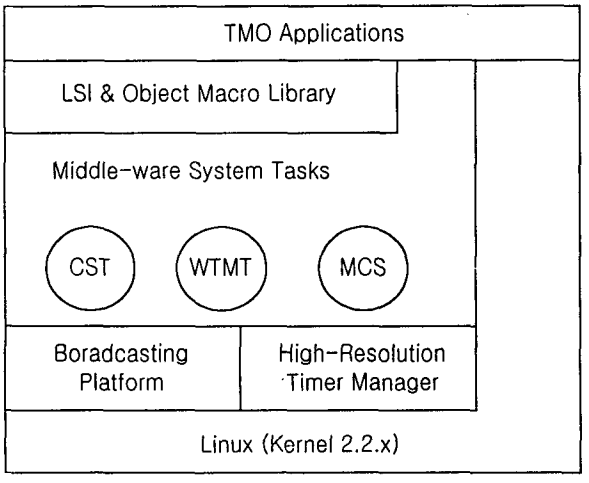


그림 1. TMO의 구조

3. 설계 및 구현

리눅스 환경에서 TMO 객체모델 수행 플랫폼 LTMOS 는 그림 2 와 같이 5 가지 계층 구조를 갖는다.



LSI : LTMOS Service Interface
CST : Clock Synchronization Task
WTMT : Watchdog TMO Management Task
MCS : Message Commnucation System

그림 2. LTMOS의 계층구조

3.1 TMO 메소드의 LTMOS 모델

TMO 의 SpM 과 SvM 은 함수 호출에 의한 수행이 아닌 각 메소드의 시간적인 특성이나 메시지에 의해 구동 되는 능동적인 메소드이다. 리눅스에서 TMO 의 메소드들은 각각 하나의 쓰레드로 구현되어진다. 쓰레드로 구현하기 위하여 TMO 의 매소드를 쓰레드의 매소드로 변환하기위한 매크로 라이브러리를 사용한다.

3.2 LTMOS 에서의 SpM 과 SvM 의 실행 모델

SpM 은 시간적 특성에 따라 주기적으로 수행되는 TMO 의 메소드이다. LTMOS 초기화 시에 생성된 SpM 은 THREAD_PRIORITY_NORMAL 로 생성되고 등록을 마친 후에는 블록(BLOCK)되었다가 LTMOS 의 시스템 쓰레드인 WTMT 의 스케쥴링을 받아 작업을 수행하게 된다. 표 1 은 TMO 및 LTMOS 시스템 쓰레드 우선순위 단계를 나타낸다.

시간적 특성에 의해 구동되는 SpM 과는 달리 분산 메시지에 의해 구동되는 SvM 의 정보는 해당 메시지를 처리하기까지의 데드라인 만으로 이루어져 있다. SvM 은 생성시에 이 정보를 LTMOS 에 등록하고 블록 되어 메시지를 기다리다가 분산 IPC 메시지 처리를 담당하는 IMMT 에 의해 깨어나 WTMT 의 스케쥴링을 받게 된다.

표 1. TMO 및 LTMOS 시스템 쓰레드 우선순위 단계

1 단계	THREAD_PRIORITY_TIME_CRITICAL
2 단계	THREAD_PRIORITY_HIGHEST
3 단계	THREAD_PRIORITY_ABOVE_NORMAL
4 단계	THREAD_PRIORITY_NORMAL
5 단계	THREAD_PRIORITY_BELOW_NORMAL
6 단계	THREAD_PRIORITY_LOWEST

3.2 TMO 메소드 스케쥴링

WTMT 는 TMO 메소드의 미들웨어 시스템의 시스템 쓰레드로 천분의 일초의 정밀도를 갖는 고해상도 클럭 관리기에 의해 주기적으로 깨어나 SpM 의 시간조건에 의한 구동과 SpM 과 SvM 의 TUF(Task Urgent Factor)[1]에 따른 우선순위 조정, 데드라인 감시 및 예외 처리기의 구동을 담당한다. 그림 3 은 WTMT 의 작업을 나타낸다.

```
Wait for periodic invocation;
Update timer-related informations in MCB's;
For each inactive SpM
        If(time to activate) Activate the SpM
        If(time to stop) Stop the SpM
For all active SpM's and SvM's
        If(deadline is passed)
                Invoke the exception handler;
        Else
                Reassign priority due to the TUF;
```

그림 3. WTMT 의 작업

3.3 LTMOS 의 IPC 모델

LTMOS 에서 원격 혹은 지역 TMO 간의 IPC(Inter Process Communication)는 채널 단위의 브로드캐스팅을 사용한다. 이것은 어떤 TMO 가 메시지를 한 채널에 보내면, 이 채널에서 대기하고 있는 모든 TMO 가 이 메시지를 받는 것을 의미한다. 채널은 내부적으로 포트로 매핑되어진다.

3.4 MCS 의 구조

MCS (Message Communication Subsystem)는 분산 IPC 를 담당하며 고해상도 클럭 관리기(High Resolution Clock Manager)에 의해 주기적으로 호출되는 LTMOS 시스템 태스크인 IMMT, OMMT 와 분산 IPC 로 구성된다. LTMOS 의 시스템 쓰레드인 IMMT 는 TMO 를 수행시키는 모든 분산 노드에 브로드캐스팅

되는 메시지를 IMQ에 삽입하고, 대기 상태중의 SvM을 스케줄링 받을 수 있는 상태로 만들어 주는 작업을 한다. OMMT는 로컬 시스템에서 SpM이나 SvM의 메시지 송신 서비스 호출에 의해 XMQ에 버퍼링된 메시지들 중 원격 TMO로 전달되는 것들을 브로드캐스팅 한다.

3.5 IMMT

IMMT는 네트웍을 통하여 전달되는 메시지를 TMO에게 전달하는 LTMOS의 시스템 쓰레드이다. IMMT는 주기적으로 구동 되어 네트웍으로부터 전달된 메시지를 받아 해당 메시지가 전달 되어질 TMO가 있는 지를 검사한다. IMMT는 메시지를 기다리고 있는 SvM이 있을 경우, SvM을 ACTIVE 상태로 만들어 스케줄링 받을 수 있는 상태로 만들어 주는 작업을 한다. 그림 4는 IMMT의 작업을 나타낸다.

```
Wait for Periodic Invocation
While(message exist in Channel)
{
        Get Message;
        If(Used Channel)
                Put messages into IMQ;
        If(!Empty(Waiting Queue))
        {
                Activate SvM
        }
}
```

그림 4. IMMT의 작업

3.6 OMMT

OMMT는 지역 노드에 TMO들이 저장해 놓은 XMQ의 메시지를 특정한 포트로 전달하는 역할을 하는 LTMOS의 시스템 쓰레드이다. 리눅스 TMOS의 고해상도 클럭 관리기(High Resolution Clock Manager)에 의해서 주기적으로 호출된 OMMT는 로컬 시스템의 XMQ를 검사한다. XMQ에 메시지가 존재할 경우 먼저 해당 메시지가 로컬 시스템에도 전달되어야 하는지를 확인한다. 만약 로컬 시스템에도 메시지를 기다리고 있는 SvM이 존재할 경우 IMMT와 같이 해당 SvM을 ACTIVE 상태로 만들어 준다. 이러한 작업을 수행한 후 OMMT는 메시지를 네트웍에 전달한다. 그림 5는 OMMT의 작업을 나타낸다.

```
Wait for Periodic Invocation
For(all message in XMQ)
{
        Remove message from XMQ;
        If(local channel Used in local)
        {
                Insert message into IMQ;
                Active SvM;
        }
        Broadcast Message;
}
```

그림 5. OMMT의 작업

3.7 브로드캐스팅 플랫폼

LTMOS의 브로드캐스팅 플랫폼은 동일한 네트웍 세그먼트에서 메시지를 모든 컴퓨터에 전달하기 위하여 포트별 데이터그램 방식의 브로드캐스팅 프로토콜을 사용한다. 사용된 브로드케스팅 주소는 Ipv4 주소체계를 따라 255.255.255.255 이며 같은 렌 세그먼트에 있는 컴퓨터들로 브로드케스팅 됨을 확인할 수 있었다. 또 LTMOS가 실시간 시스템이므로 속도를 위해 TCP보다는 UDP를 이용하였다.

3.8 클락 동기화

분산 노드의 클락 동기화는 미들웨어 태스크 중 하나인 CST(Clock Synchronization Task)가 담당하게 된다. 분산 환경하에서 독립적인 하드웨어 자원을 가지고 실행되는 각 노드의 클럭은 동시에 시작되었다 하더라도 하드웨어의 특성과 환경에 따라 조금씩 다른 주기를 가지게 되고 시간이 지날수록 오차를 가지게 된다. 이러한 문제는 분산 시스템 하에서 시간적 조건에 따라 수행되어야 하는 실시간 플랫폼이나 실시간 응용 프로그램에서 여러가지 문제를 발생시키며, 이러한 오차의 누적을 막기 위하여 LTMOS의 시스템 태스크인 CST는 논리적 클락[4]을 사용하여 주기적으로 전체 분산 LTMOS의 클럭을 동기화 시킨다.

4. 결론 및 향후 연구방향

본 논문에서는 리눅스 운영 체제가 갖는 특징을 이용에서 실시간 객체모델을 수행할 수 있는 실시간 객체모델 수행 플랫폼을 미들웨어 형태로 설계 및 구현에 대해 기술하였다.

앞으로 TMO 응용 프로그램 개발자들을 위한 모니터링 및 디버깅 도구가 추가되어져야 할 것이며 TMO의 스케줄링 부분이 본 논문에서는 LTMOS가 간접적으로 제어하여 리눅스 커널에 반영되도록 하였지만 공개되어 있는 리눅스 커널 루틴 중 쓰레드 스케줄링에 해당하는 부분을 TMO 시스템에 적합하도록 직접 수정한다면 LTMOS가 연성 실시간 시스템에서 좀 더 경성 실시간 시스템에 가까워 질 수 있을 것이다. 또 이미 개발되어 있는 다른 TMO 시스템간에 인터페이스를 표준화하여 이기종 컴퓨터 사이에서도 TMO를 동시에 실행시킬 수 있도록 해야 할 것이다.

참 고 문 헌

[1] 이충환, " 실시간 객체모델 수행 플랫폼을 위한 분산 IPC의 설계 및 구현", 석사학위논문, 5. 1998

[2] Jungguk Kim and Kee-Wook Rim, "A Timeliness-Guaranteed Kernel Model – DREAM Kernel – and Implementation Techniques", Proc. RTCSA '95, pp.80-87

[3] K.H.Kim, "Object Structures for Real-Time Systems and Simulators", IEEE Computer, 8. 1997, pp.62-70.

[4] Andrwe S. Tanenbaum, "Distributed Operating System", Prentice Hall, pp.119-133, 1996

[5] Barabanov and Yodaiken , "Real-Time Linux", Linux Journal, Mar. 1996

[6] Xavier Leroy, "LinuxThreads Library", http://pauillac.inria.fr/~xleroy/linuxthreads/