가상 쓰레드를 이용한 TMO기반 미들웨어의 설계

이은파° 김문회
건국대학교 컴퓨터공학과
{eplee°, mhkim}@konkuk.ac.kr

Implementation of TMO Middleware based on Virtual Threads

Eun-Pa Lee° Moon-Hae Kim
Computer Science and Engineering Graduate School, Konkuk University

요 약

TMOSM/Linux v2.0[1] 은 리눅스/임베디드 리눅스에서 분산 실시간 객체 모델인 TMO로 작성된 프로그램을 실행시키는 미들웨어이다. 최근 임베디드 환경의 급속한 발전과 더불어 다양한 실시간 기반 제어 및 개발 방법론에 대한 요구사항 역시 증가하고 있지만, 임베디드 리눅스를 비롯한 여러 임베디드 OS에서는 이들을 모두 충족시키는데 여러 단점들이 존재하고 있다. 따라서 TMO기반 미들웨어는 임베디드 OS의 단점들을 보완함은 물론 TMO의 장점을 최대한 끌어낼 수 있는 형태로 구축되어야 할 것이다. 본 논문은 리눅스/임베디드 리눅스에서 TMOSM/Linux v2.0이 TMO 기반 미들웨어로서 그 역할을 최대한 발휘할 수 있도록 구성한 방법 중 가상 쓰레드를 이용한 설계와 그 결과를 보인다.

1. 서 론

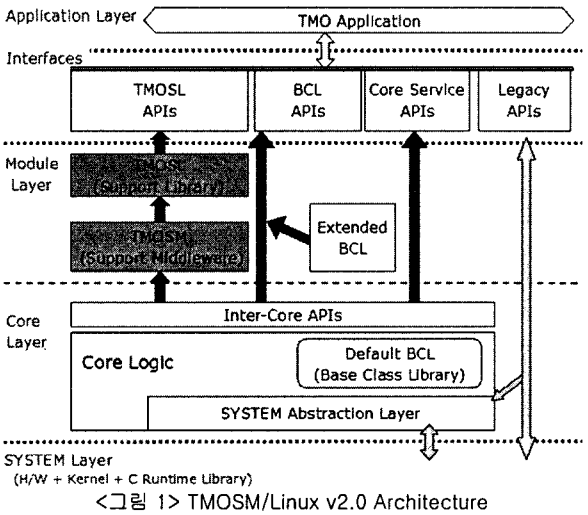
TMOSM/Linux는 Linux 버전의 TMO 기반 미들웨어를 지칭하며 본 논문에 소개되는 v2.0[3]은 2004년 말에 개발되었다. TMO는 기존 객체 모델을 실시간 분산 컴퓨팅을 위해 확장한 모델이며, 기존의 프로그래머들이 약간의 노력만으로도 분산 실시간 응용을 설계하고 구현할 수 있도록 방향을 제시한다.

TMOSM/Linux v2.0은 이전에 개발된 POSIX thread library를 사용했던 TMOSM/Linux v1.0 (TMOSM/KELIX 또는 KELIX/RT로 불리던 것을 최종 확정한 이름이다.)에서 나타났던 여러 가지 문제점들을 원천적으로 해결하고자 하는 취지에서 기본 Architecture부터 새롭게 구축된 것이다. TMOSM/Linux v2.0은 다음과 같이 정의한다.

" *TMOSM/Linux는 TMO Model을 지원하는 미들웨어 및 이에 최적화 시킨 실시간 지원 런타임 라이브러리를 총칭한다. 이는 순수 TMO형태의 실행환경 지원시 발생하는 외적 문제점들을 고려해 실시간 지원, 분산 처리 부분에 내부적으로 개선 가능성을 내포한 구조 및 라이브러리의 군집이다.*"

기존 TMO를 지원하는 미들웨어는 TMOSL (TMO Support Library)에서 정의하는 인터페이스만을 통해 요청된 것들을 서비스하며 실시간 동작을 제어하는 형태를 갖추고 있었다. 즉, TMO 외적인 요소들은 기존의 C/C++ 라이브러리들을 사용한다는 의미다. TMOSM/Linux v2.0은 바로 TMO 외적인 요소들 중 시간 결정성(Time determinism)을 위배할 확률이 높은 요소들이나 개선된 분산 처리 기능을 포함한 별도의 라이브러리(BCL : Base Class Library)를 가지는 프레임워크 구조다.

<그림 1>은 앞서 말한 TMOSM/Linux v2.0의 전체적인 구조를 개략적으로 보이고 있다. Core Layer라고 표현된 영역은 TMO를 지원하기 이전에 실시간 지원 기능이 부족한 하부 SYSTEM 영역을 재조직해서 실시간 지원 및 분산처리기능을 제공할 수 있는 다양한 기능이 구현되었고, Module Layer는 Core Layer를 바탕으로 특정 목적을 지니도록 표현하는 부분이다. 예를 들어 TMO Module은 TMO 형태로 제어되게 한다.



<그림 1> TMOSM/Linux v2.0 Architecture

Module은 Core와 독립되어 있으므로 다시 컴파일 하지 않아도 Link-time에 실행파일과 연결만 시켜주면 작동하도록 구성되었다. 이는 Module간에도 같이 적용되며, 예를 들어 종래의 TMO지원 미들웨어의 성격을 가지는 TMOSM은 자신에게 의존하는 TMOSL을 지원하는 서비스들의 집합일 뿐, TMOSL에 의

본 연구는 정보통신부 대학 IT연구센터 육성• 지원사업의 연구결과로 수행되었습니다.

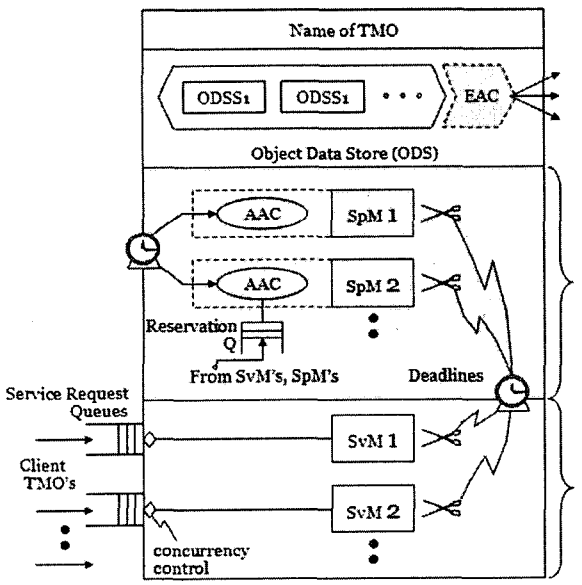
존하는 부분이 없어 서로 다른 버전의 TMOSL (현재 제공되는 것은 v2.1b와 v4.1.8d이다.)을 연결시켜도 동작할 수 있다.

이 구조는 임베디드 환경에서 제한된 리소스 (CPU, Memory, 등등…)를 최대한 활용할 수 있으며, 임베디드 리눅스에서 부족한 시간 결정성을 확보하기 위해서 메모리 관리는 물론 다양한 처리 루틴이 관리되도록 설계할 수 있다. 본 논문에서 언급할 Single Thread 기반 Virtual Multi-Thread를 지원하기 위한 Thread Library (이하 VTMS : Virtual Thread Management System)는 앞서 말한 복잡해질 수 있는 것들을 단순화 시키고 효율성을 높이는데 중심역할을 수행한다.

본 논문은 2장에서 관련연구로서 TMO 모델이 가지는 특징과 User-space관리 구조에 대해 설명하고, 3장에서 VTMS의 제어 및 이를 지원하기 위한 스케줄링 방법에 관련해 설명하며, 4장에서는 분산환경에서의 테스트 및 결과를 보이고, 5장에서는 결론 및 향후 과제를 기술한다.

2. 관련 연구

2-1. TMO 모델

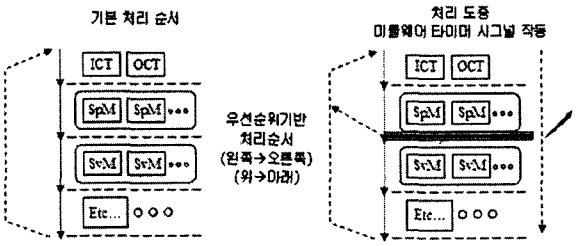


<그림 2> TMO 모델

분산 실시간 객체 모델 TMO(Time-triggered Message-triggered Object)[1]는 분산 실시간 시스템의 시간 보장성 컴퓨팅 설계를 지원하기 위한 목적으로 제안되었다. TMO의 구조는 <그림 2>와 같으며 이는 일반 객체와는 그 구조에 있어서 몇 가지 다른 점을 가진다. TMO는 크게 ODS(Object Data Store), 시간 구동 메소드 (Time - triggered Method: Spontaneous Method: SpM) 그룹, 그리고 메시지 구동 메소드 (Message - triggered Method: Service Method: SvM) 그룹의 세 요소들로 구성된다. TMO기반 미들웨어[2]의 입장에서 이러한 요소들은 <그림 1>에서의 SYSTEM 요소들을 이용해 표현할 수 있어야 하며, 크게 Thread들의 관리, 시간 및 우선순위 기반 스케줄링, 임계영역처리(IO 처리 포함), 투명한 네트워크 프로토콜, 노드간 시간 동기화 등등의 문제를 언급할 수 있다.

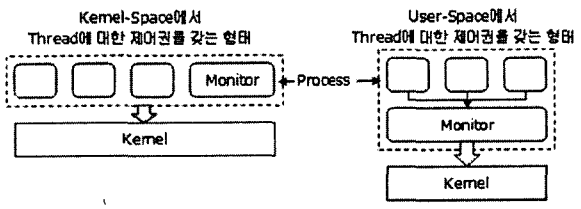
2-2. User-space 관리 구조

TMO기반 미들웨어[2]가 가져야 할 특히 중요한 요소 중 하나는 적시에 동작하고 데드라인 이내에 끝낼 수 있는 메커니즘이다. 실시간 프로그램은 디자인 시점에서 TMO를 기반으로 구성할 때, 여러 가지 시간 조건 및 처리 시간들을 고려해 설계하게 될 것이다. 따라서 주어진 데드라인 값은 다른 객체들과의 관계 및 하드웨어의 성능을 충분히 감안한 수치가 된다. 이를 위해서는 미들웨어만의 스케줄러가 요구되며, 이 스케줄러의 제어 방식은 실시간 우선순위에 의거해 <그림 3>과 같이 단순화 시킬 수 있을 것이다.



<그림 3> TMO Scheduler

미들웨어 사이에서 주고 받는 메시지들의 분석 및 대상 객체에 메시지를 전달하는 ICT, 특정 메시지를 보내고자 하는 객체로 내보내는 OCT, 기타 TMOSL에서 보이는 IO Thread나 그 외의 일반적인 Thread들은 서로 독립적인 실행 주체로서 대부분의 경우 원하는 동작을 보일 것이다. 문제는 그렇지 않는 경우인데, 이를 테면 이들간의 IO에 의한 우선순위 역전 현상이나 실행 객체들을 포함한 미들웨어 내부 객체들이 공유하는 자료에 대한 경쟁 상태, 미들웨어 외적인 OS와 같은 공간에서의 경쟁 상태 등등 예측하기 힘든 영역에 대한 문제는 시간 기반 제어에 큰 장애가 된다. 이를 해결하기 위한 방법으로 많은 해결책이 제시되었고, 그 중 TMOSM/Linux v2.0에서는 언급된 문제들을 원천적으로 최대한 피해갈 수 있는 방법을 고안했다. <그림 4>는 프로세스 레벨에서 Thread의 상태를 알고 관리할 수 있는 모델이며, 앞서 언급된 문제들을 해결하는데 매우 간단한 접근방법을 제시한다.



<그림 4> User-space에서의 Thread 관리 모델

Kernel을 비롯한 미들웨어 영역 외부에서 지원되는 Thread 라이브러리를 사용할 경우, 이것이 어떻게 동작하는지, 특정 상태를 어떻게 감지할 수 있는가는 복잡하거나 어려워 앞서 말한 문제들을 쉽게 피해갈 수 없게 된다. TMOSM/Linux v2.0은 이러한 점에서 미들웨어 내부에 Thread 관리 라이브러리인 VTMS를 만들었으며, <그림 4>의 오른쪽과 같은 구조로 구성되어 있다. 기타 다른 미들웨어 기능들도 관리되는 구조를 위해 마찬가지 구성을 취하고 있다.

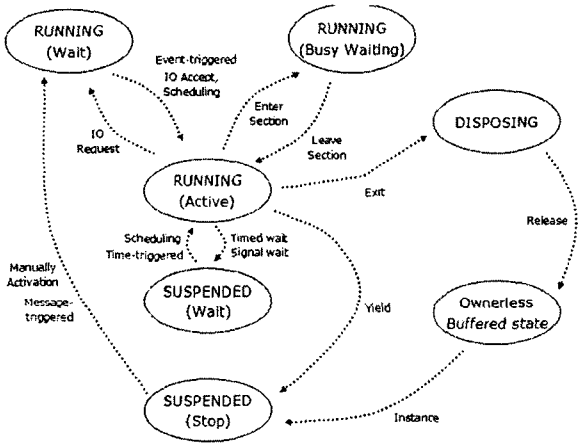
3. VTMS (Virtual Thread Management System)

TMOSM/Linux v2.0의 Core Layer는 다양한 성격을 가진

Module로 조립되어 있다. VTMS는 그 중 하나이며, GNU Pth[4]의 기술적 메커니즘을 도입했다. 이것은 비선점 스케줄링 기반으로 하나의 프로세스에서 사용자 레벨의 Context들을 구축해 Virtual Thread를 지원하는 형태로 구성되어 있다. 하지만 이것은 명백히 단점이 존재한다. 우선 IO 요청에 대한 프로세스 Suspend 상태를 방지하기 위해서 non-blocking IO를 사용해야 한다. 사용자에게는 일반적인 프로그래밍을 하더라도 느끼지 못하도록 <그림 1>의 가장 오른쪽 수직화살표와 같은 IO blocking 상황을 발생시키는 인터페이스에 대해 후킹을 수행해야 한다. 특히 스케줄러 역시 동일한 Thread에서 동작하기 때문에 자발적인 양보를 통해 각 Context, 즉 Virtual Thread간의 전환이 이뤄지게 해야 한다.

이런 큰 단점들이 존재하는 Thread Library를 구축한 이유는 이러한 단점들을 실시간 지원 미들웨어에서 오히려 큰 장점으로 만들 수 있었기 때문이다.

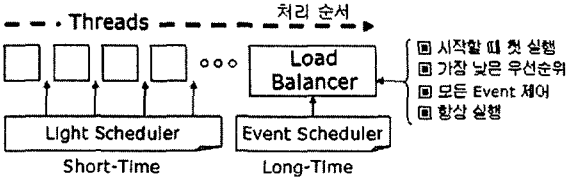
첫째, 결국 하나의 프로세스에서의 작업이기에 Context들 사이의 메모리 동기화 문제는 존재할 수 없다. 둘째, 대부분을, 특히 IO에 관련된 부분을 후킹해 미들웨어 영역에서 순차적이면서도 우선순위 기반 IO 처리를 수행할 수 있어, 우선순위 역전 현상을 없앨 수 있다. 셋째, 단일 Thread이기에 이것만 높은 우선순위로 지정한다면, 필요한 만큼 처리하고 제어권을 반납할 수 있다. 이것은 시간 결정성을 높이며 수치화가 가능해 비선점 스케줄링 알고리즘의 적용이 용이하다. 결국 각 Virtual Thread 당 처리량을 높일 수 있어 데드라인을 여유롭게 만들 수 있다. 넷째, 디자인 타임의 설계, 그리고 약간의 보정만 거치면 사용자 요구사항을 그대로 반영할 수 있는 프로그램의 제작이 가능해진다. 예를 들면 구현된 모든 로직이 각각 최대 1밀리초의 처리량이 넘지 않는다면 시간 정확도를 그 정도 수준까지 제공할 수 있다. 다섯째, 결국 사용해야 하는 미들웨어 영역 외부의 OS나 각 C Runtime Library와 같은 영역을 사용함에 있어도 예기치 못한 제어권을 반환할 확률을 크게 떨어뜨릴 수 있다.



<그림 5> Virtual Thread Transition

<그림 5>는 VTMS에서 앞서 말한 장점들을 최대한 살리고 TMO의 특성이 반영될 수 있도록 변형한 Virtual Thread가 변하는 상태도를 보이고 있다. 기존과 차이점은 RUNNING의 상태가 세분화 되었다는 점이며, 결국 Thread의 제어 조건들을 TMO의 성격에 맞게 설정할 수 있게 되었다.

VTMS의 첫번째 장점과 시간 기반 비선점 스케줄러를 고려할 때, 스케줄러의 기능이 한곳으로 집중될 필요는 없다. 하지만 두번째 장점을 적용하는 데는 스케줄러의 기능이 집중되는 것이 좋으며, 이 두 가지를 고려해 <그림 6>과 같은 스케줄러가 구현되었다.

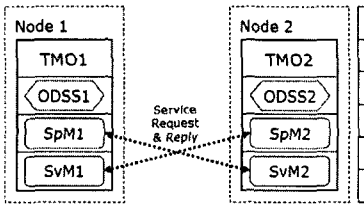


<그림 6> VTMS Scheduler

4. 테스트 및 결과

VTMS의 성능 평가는 물리적으로 독립된 두 노드 사이에서 각각 SpM이 다른 노드의 SvM에 메시지 (1Kbyte)를 주기적(10msec)으로 보내 메시지 수신 결과를 받기까지의 시간으로 결정했다. 이유는 메시지를 받거나 보내는 이들은 SpM들과는 독립적으로 조직되는 미들웨어 Thread가 담당하기 때문이다. (Thread들이 OS의 제어를 받는 경우에는 OS의 성능에 따라 그 차이를 보이며, 그 주기도 timer signal에 기반하기에 최소 수 msec 정도의 지연을 받게 될 것이다.)

Node1은 Pentium IV 2.4GHz Hyper-Threading이 지원되고, Node2는 Pentium IV 1.5GHz이다. 원활한 디버깅을 위해 미들웨어의 우선순위를 낮췄다. 각각 1초당 100번의 메시지가 교환되며, 각각 평균을 보였다. 가상 Thread의 개수는 미들웨어 내부 3개를 포함한 각각 5개로 이 테스트에서 관심 사항은 VTMS의 장점으로 거론한 세번째와 네번째의 검증이다.



<그림 7> 테스트 TMO 프로그램 구조

Node1	Node2
00초.000.357	00초.000.331
00초.000.355	00초.000.339
00초.000.345	00초.000.331
00초.000.342	00초.000.333
00초.000.346	00초.000.329

<그림 8> 테스트 결과

5. 결론 및 향후 과제

지금까지 TMOSM/Linux v2.0의 VTMS에 대한 개념과 그 성능을 보였다. 테스트 결과는 400 μ sec 이내의 응답성능에 10 μ sec 단위의 오차를 제외하고는 항상 일정한 수치를 기록함을 보이고 있다. 이 수치는 임베디드 시스템에서는 위 결과보다 확실이 떨어진 수치가 나오지만 성능상의 문제일 뿐 결국 큰 외압이 없는 한 계속해서 보여줌을 확인했으며, 이는 기대했던 세번째와 네번째의 장점을 검증해준다.

하지만 다른 장점들에 대한 검증 방법이나 좀더 근거 있는 테스트 기준을 설정하지 못하고 있으며, 향후 좀더 신뢰성 있는 미들웨어가 되려면 좀더 여러 가지 테스트를 거쳐야 할 것이다.

참고 문헌

[1] Kim, K.H.: APIs for Real-Time Distributed Object Programming. IEEE computer, (2000) 72-80
[2] Kim, K.H., Ishida, M., and Liu, J.: An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation. ISORC, (1999) 54-63
[3] http://rtselab.org
[4] http://www.gnu.org/software/pth/pth.html