DDS 에서의 타이머 쓰레드의 설계 및 구현

강민균*, 차다함*, 김용연*, 최훈*
*충남대학교 컴퓨터공학과
e-mail : nssang@cnu.ac.kr, hc@cnu.ac.kr

Design and Implementation of Timer Thread on DDS

Min-Gyun Kang*, Da-Ham Cha*, Yong-Yeon Kim*, Hoon Choi*
*Dept. of Computer Engineering, Chung Nam University, Korea

요 약

데이터 중심 발간-구독 방식의 DDS 에서 특정 시간 및 일정 시간 간격으로 발생하는 이벤트를 처리하기 위한 타이머 모델을 쓰레드풀 기반으로 구현하였으며, 기존 병행처리 모델과 문제점을 제시하고, 제안한 모델의 구조 및 동작 방식에 대하여 제시하였다. 타이머 서비스를 지원하기 위해 사용된 각 클래스의 기능 및 구현을 통해서 연성 실시간 이벤트 처리가 가능한 타이머 모델을 어떻게 구현하였는지 제시한다.

1. 서론

DDS(Data Distribution Service) 는 데이터 중심 발간-구독 (publish-subscribe) 방식을 채택한 표준 통신 미들웨어 규격으로서, 메시지의 송신과 수신 시에 특정 시점이나 특정 간격으로 처리되거나 호출되어야 하는 이벤트가 빈번히 발생한다. 기존의 병행처리 모델은 이들을 특정 시간 지연 내에 처리하기에 부족함이 있다. 타이머 서비스는 특정 시점이나 특정 시간 간격으로 발생이 가능하며, 연성 실시간(soft realtime) 이벤트 처리가 가능해야 한다.

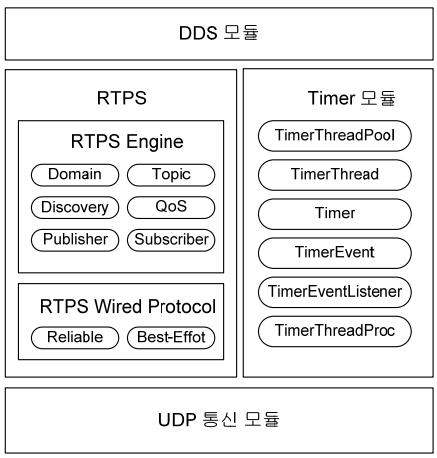
본 논문에서는 기존의 병행 처리 모델의 단점을 극복하고, 연성 실시간 시스템의 이벤트를 처리하는 타이머 모델을 제시하고자 한다.

논문의 구성은 다음과 같다. 2 절에서 DDS 에서의 타이머 모듈의 역할에 대해서 알아보고, 기존의 병행처리 모델과 문제점을 통해 이 논문에서 제안하는 쓰레드풀(pool) 기반의 타이머 쓰레드에 대해서 서술하고, 3 절에서는 타이머 쓰레드의 설계와 구현을 통해서 이 논문에서 제안하는 타이머 쓰레드의 동작 방식을 알아본다. 마지막으로 4 절에서는 이벤트의 발생량에 따른 동적인 쓰레드 생성에 대해 언급하고 결론을 맺는다.

2. 타이머 모듈의 역할

(그림 1)은 DDS를 간략하게 나타낸 구조이다. Timer 모듈은 해당 DDS 모듈의 하부 구조로서 RTPS 모듈과의 연동 과정을 통해 UDP통신 모듈을 이용하여 통신하게 된다. RTPS 모듈은 메시지를 주고 받는 이웃 참여자가 통신에 참여하고 있는 지를 확인하기 위하여 주기적으로 HEARTBEAT 메시지를 전송한다. 내부적으로 heartbeatPeriod 속성을 통해 주기적으로 전송해야할 타이머를 설정하게 된다. 메시지 송신자가 너무 빠르게 메시지를 전송하는 것으로부터 수신자를 보호하기 위해 minimumSeparation 속성을 위해서도 타이머를 설정한다. 이외에도 많은 DDS와 RTPS의 속성들이 일정간격으로 호출되기 위해 타이머를 설정하게 된다.

언급한 속성들을 가장 간단하게 해결하는 방법은 필요한 곳마다 쓰레드를 생성하여 지연시간 만큼 sleep을 하는 것이다. 그러나 이런 방법은 쓰레드의 생성과 파괴가 빈번하여 비효율적인 방법이다.



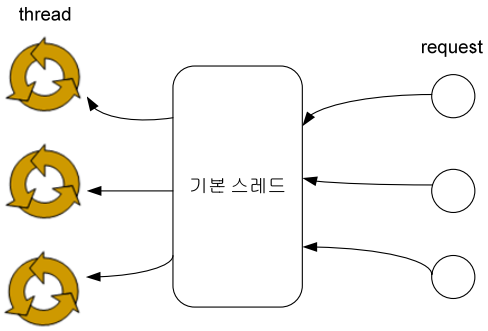
(그림 1) DDS 모듈 구조

2.1 기존의 병행처리 모델과 문제점

2.1.1 Thread-Per-Request 모델

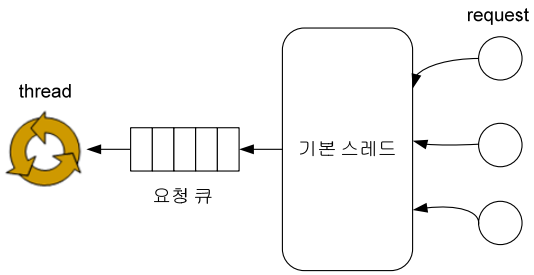
Thread-Per-Request 모델은 (그림 2)와 같이 발생하는 모든 요청에 대해 새로운 쓰레드를 생성하여 처리하고 소멸시키는 방식이다. 구조가 간단하여 구현

이 용이하며, 모든 요청에 대해 쓰레드를 생성하므로 대기 상태 없이 바로 처리된다는 장점을 가지고 있다. 그러나, 동시에 많은 요청이 발생할 경우 그에 비례하여 많은 수의 쓰레드가 생성되어 시스템의 자원 소비량이 증가하여 부하가 발생하는 단점이 있다.



(그림 2) Thread-Per-Request 모델

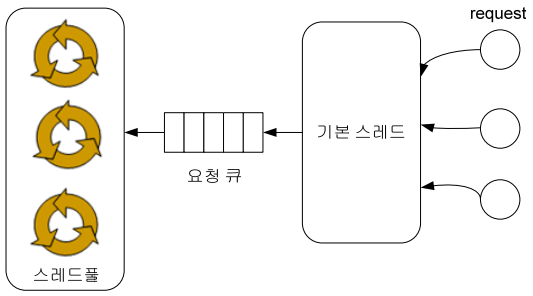
2.1.2 One Thread-Request Queue 모델



(그림 3) One Thread-Request Queue 모델

(그림 3)과 같이 One Thread-Request Queue 모델은 발생하는 모든 요청에 대해 요청 큐(queue)에 저장하면, 대기 중인 쓰레드가 요청 큐의 메시지를 처리하는 방식이다. Thread-Per-Request 모델과 같이 쓰레드의 생성과 소멸이 요청마다 발생하는 것이 아니라 한번 생성된 후 소멸되지 않고, 요청 큐의 모든 메시지를 처리하게 된다. 동시에 많은 요청이 들어와도 시스템 자원 소비량은 같다는 장점이 있다. 그러나, 독립된 쓰레드를 통해 즉시 실행하는 것이 아니기 때문에 동시에 많은 요청이 있을 경우 요청 큐의 메시지를 순차적으로 처리해야 하므로 지연이 발생할 수 있는 단점이 있다.[5]

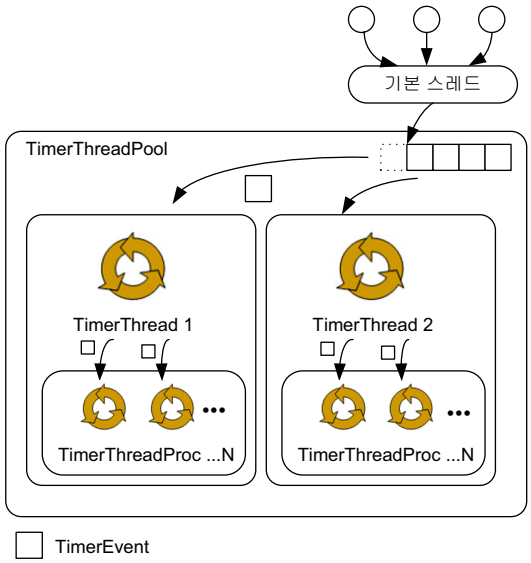
2.1.3 ThreadPool 모델



(그림 4) ThreadPool 모델

(그림 4)와 같이 ThreadPool 모델은 실행 초기에 일정 수의 쓰레드를 생성하여, 최대 그 수 만큼의 쓰레드가 요청을 처리하는 방식이다. Thread-Per-Request 모델에서 발생하는 쓰레드 생성에 따른 부하 문제를 해결하기 위한 모델로서, 요청이 많이 발생하여도 쓰레드 생성과 파괴에 따른 부하가 발생하지 않는 장점이 있다.

2.1.4 쓰레드풀 기반 타이머 모델



(그림 5) 쓰레드풀 기반 타이머 모델

(그림 5) 와 같이 쓰레드풀 기반 타이머 모델은 우리가 제안하는 모델로서 요청 큐에서 메시지를 가져오는 TimerThread 와 가져온 메시지를 타이머 이벤트에 설정된 지연 시간 이후 요청을 처리하는 TimerThreadProc 쓰레드로 구성 된다.

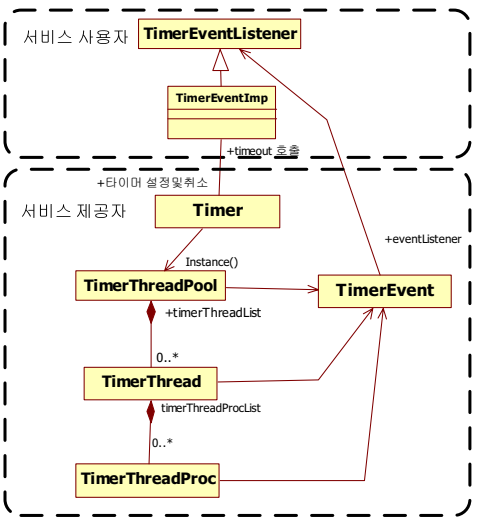
일정 시간 이후에 요청을 처리해야 하는 타이머 서비스에서 ThreadPool 모델에서는 많은 요청이 있을 경우 고정된 쓰레드를 사용하기 때문에 시간 지연이 발생할 수 있다. 정해진 시간에 요청을 처리해야 하는 타이머 서비스 모델에서는 적합하지 않기에 (그림 5]와 같은 모델을 제안하였다. TimerThread 쓰레드는 일반 쓰레드풀 모델과 같이 요청큐에서 메시지를 바

로 가져오고 요청을 처리하는 TimerThreadProc 쓰레드에 넘겨준다. TimerThreadProc 쓰레드는 TimerThread 쓰레드 마다 2 개가 기본적으로 생성되어 있고, 1~2 초 안에 호출되는 메시지를 번갈아 처리한다. 2 초 이상의 상대적으로 긴 지연 시간 이후의 작업은 새로운 TimerThreadProc 을 생성하고, 지연 시간 만큼의 sleep 후에 호출 후에 소멸된다. 제안한 모델과 같이 이중 구조의 쓰레드풀을 이용하여 쓰레드 생성과 소멸에 따른 부하를 해결하고, 많은 요청 처리에 의해 발생할 수 있는 지연 시간을 줄인다.

3. 타이머 서비스 설계 및 구현

타이머 서비스는 특정 시점이나 특정 시간 간격마다 등록된 객체를 호출해주는 서비스이다. 연성 실시간(soft realtime)이벤트를 처리하기 위한 서비스이므로 밀리 초까지의 정교한 서비스를 요구하지 않으며 약간의 시간 지연이 있어도 크게 문제 되지 않는다. 타이머는 추가 및 삭제가 용이하고, 특정 시점이나 특정 간격, 반복 횟수를 지정할 수 있다. 타이머 서비스를 받으려는 객체 내에서 여러 개의 타이머 이벤트 생성이 가능하며 객체내의 타이머 이벤트 발생 시간도 독립적으로 발생한다.

3.1 타이머 서비스 설계



(그림 6) 타이머 서비스 구조도

(그림 6)은 타이머 서비스 구조도를 나타낸 것으로서 크게 타이머 서비스를 사용하는 서비스 사용자와 타이머 서비스를 제공하는 서비스 제공자로 나누어진다. 사용자 입장에서는 TimerEventListener 인터페이스를 상속받은 클래스의 timeout 함수를 구현하고, Timer 객체를 이용하여 특정 시점 혹은 특정 간격으로 이벤트 객체를 등록한다. Timer 객체를 통해 등록한 시점이나 간격이 발생하면 서비스 제공자는 TimerEvent 에 등록한 TimerEventListener 객체의 timeout 을 호출하여 준다.

3.2 타이머 서비스 사용자

타이머 서비스와 관련된 클래스는 다음과 같다.

- TimerEventListener 인터페이스

서비스를 제공 받을 객체가 상속받아서 timeout 함수를 구현한다. 인자로 넘겨받는 TimerEvent 객체의 eventID 를 통해 여러 개의 타이머 서비스를 구현할 수 있다.

- Timer 클래스

서비스를 제공 받기위해 Timer 클래스를 통해서 타이머를 등록(setTimer)하거나 취소(killTimer)할 수 있다. eventID 를 통해 한 객체 내에서 여러 개의 타이머 서비스를 받을 수 있다.

3.3 타이머 서비스 제공자

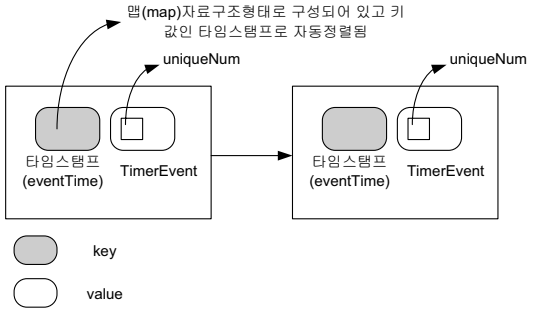
3.3.1 Timer 클래스

사용자가 setTimer 함수를 통해 이벤트 등록을 하게 되면 TimerThreadPool 객체의 addTimerEvent 함수를 호출하여 이벤트를 등록하고, Timer 객체마다 이벤트 등록에 관한 정보들을 관리한다. Timer 객체 내부의 이벤트 등록 정보는 타이머의 수정이나 삭제시에 이용한다. killTimer 함수를 통해 이벤트 삭제를 하게 되면 이벤트 등록시 TimerThreadPool 객체로부터 얻어온 정보를 이용하게 된다. 이벤트 등록과 관련된 정보는 타이머 이벤트 고유의 아이디인 uniqueNum, Timer 객체 내부에서 구분하는데 사용하는 eventID, 이벤트가 발생할 시간정보인 eventTime 이다.

3.3.2 TimerThreadPool 클래스

타이머 서비스의 가장 핵심적인 역할을 하는 클래스로서 하나의 프로그램 내에서 하나의 객체만 생성하도록 클래스를 설계하는 싱글톤 패턴 형태를 갖게 된다. TimerThreadPool 클래스는 TimerEvent 정보를 맵(map) 자료구조 형태로 갖고 있게 된다. 맵의 키 값은 타임스탬프 정보인 eventTime 으로 새로운 이벤트 추가 시 빠른 추가가 가능하다. 생성 초기에 고정된 수의 TimerThread 객체를 생하고 TimerEvent 추가와 생성 작업을 한다. deleteTimerEvent 함수를 타이머 이벤트를 삭제하게 되는데, 2 가지 방법이 되는데, eventTime 은 맵 자료구조의 키 값에 해당하기 때문에 빠른 삭제가 가능하지만 단발성의 이벤트에 해당하는 경우이고, 반복 동작하는 이벤트에서는 사용할 수 없다. 반복 횟수에 의해 타이머 이벤트가 2 번째 발생할 경우 키 값에 해당하는 타임스탬프는 새로운 타임스탬프로 수정하게 되는데, Timer 객체는 초기 생성시의 타임스탬프 값만을 갖고 있게 되므로 타임스탬프 키 값을 통한 삭제가 불가능하다. 그래서 2 번째 방법인

uniqueNum 을 이용한 방법을 이용하게 는데 순차방식으로 모든 자료구조를 조사해야 하는 부하가 존재한다. (그림 7)은 타이머 이벤트의 맵 자료 구조를 보여주고 있는 그림이다.



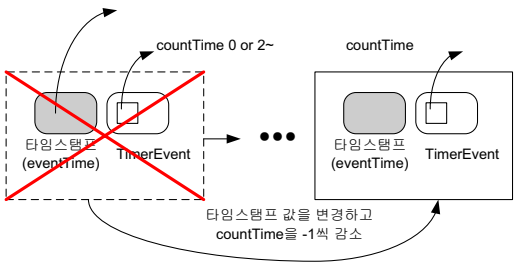
(그림 7) 타이머 이벤트 리스트 자료 구조

3.3.3 TimerEvent 클래스

타이머 이벤트 관련 정보를 가지는 클래스로서 이벤트의 유일한 ID 인 uniqueNum, 객체 내부에서 사용하는 eventID, 일정 시간 후에 발생하기 위해 사용하는 aftertime, 이벤트 발생 주기 시간을 나타내는 periodTime, 이벤트 반복횟수를 나타내는 eventCount, 일정 시간 후 발생할 TimerEventListener 를 가르키는 eventListener 로 구성되어 있다.

3.3.4 TimerThread 클래스

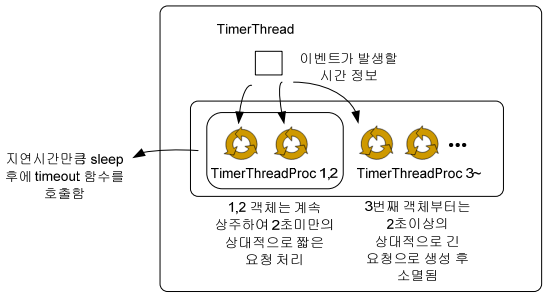
TimerThreadPool 클래스로부터 타이머 이벤트 자료 구조를 인자로 넘겨 받아 생성된다. 계속 해서 타이머 이벤트 자료구조를 검사하여 TimerThreadProc 클래스에 넘겨주는 역할을 한다. (그림 8) 과 같이 TimerThread 클래스는 TimerThreadProc 에 이벤트를 넘겨주기 전에 TimerEvent 의 countTime 을 조사한다. countTime 이 0 일 경우 무한반복이고, 그 외는 반복 횟수를 나타낸다. 그러므로 0 이거나 2 이상일 경우 새로운 타임스탬프로 변경하여 countTime 을 -1 감소 시킨 후 새로 등록 작업을 시도한다.



(그림 8) 타이머 이벤트 리스트 자료 구조

3.3.5 TimerThreadProc 클래스

TimerThreadProc 클래스는 TimerThread 클래스에 의해 생성되며, 지정된 시간 만큼의 sleep 작업 후에 해당 객체를 호출하는 클래스이다. (그림 9)는 TimerThread 에 이벤트 발생 시간 정보 확인후 상대적으로 짧은 시간의 요청은 초기에 생성하여 TimerThread 내의 쓰레드 풀로 있는 TimerThreadProc 1, 2 에 의해 처리되고, 상대적으로 긴 시간의 요청은 요청이 있을 때 마다 생성과 파괴를 반복하는 쓰레드에 의해 처리되는 모습이다.



(그림 9) 요청을 처리하는 TimerThreadProc 클래스

4. 결론 및 향후 연구

본 논문에서 제안하는 쓰레드풀 기반의 타이머 쓰레드는 상대적으로 많지 않은 타이머 이벤트가 발생하는 경우에 적용될 수 있는, 비교적 적은 오버헤드를 유발하는 방식이다. 이벤트 발생 시간과 상관 없이 많은 이벤트 메시지가 발생할 경우 고정적인 개수의 쓰레드를 생성한 TimerThread 에 부하가 발생할 것이다. 이벤트 개수에 따른 TimerThread 의 처리량에 따라 고정적인 쓰레드 수 외에 동적인 쓰레드 생성이 필요할 것이다.

참고문헌

[1] 원영암, 김명선, 최 훈, "DDS 기반 WTIS 설계 및 구현", 한국통신학회 2006 추계학술대회, 논문 초록집 제 32 호, pp.206, 2006. 11.

[2] Gerardo Pardo-Castellote, Bert Farabaugh, Rick Warren, “An Introduction to DDS and Data-Centric Communications”, 2005 Real-Time Innovations, August. 2005.

[3] 박충범, 권기정, 차다함, 최훈, "데이터 분배 서비스 시스템 설계", 2007 한국컴퓨터종합학술대회 논문집, Vol.34, No.1[A], pp.153-154, 2007.06.

[4] 정승욱, 이경호, 김중배, “EJB 2.1 타이머 서비스 설계 및 구현”, 한국정보과학회 학술발표논문지 제 30 권 제 2 호(III), pp.247-249, 2003.10

[5] Programming the Thread Pool in the .NET Framework, http://msdn.microsoft.com/en-us/library/ms973903.aspx.

[6] OMG, “The Common Object Request Broker : Architecture and Specification, Revision 2.0”, OMG TC Document, 1994.