직렬화 디스패처 워커모델 기법

임상우
고려대학교 컴퓨터정보통신대학원 소프트웨어 공학과

e-mail : lswnim@korea.ac.kr

Serialization Dispatcher Worker Model

Sang-woo Lim
Dept. of Software Science, Korea University

요 약

클럭 경쟁에서 코어 경쟁으로 전환된 요즘. 병렬 프로그래밍은 중요 하다. 동기화 개체를 사용하면 병목 현상이 발생하며, 1:1 Thread 모델은 자원의 낭비와 문맥전환 비용이 발생한다. Thread 풀 모델은 직렬화에 약점을 가지는게 되는데.

본 논문에서는 다중 개체 대응에 적합한 병렬 프로그래밍 모델을 제시한다.

1. 서론

최근 컴퓨팅 분야의 화두는 멀티코어이다.

서버 컴퓨팅 분야에서는 데카, 도데카 코어 CPU 가 대세로 자리 잡았으며, 일반 PC 유저들에게도 쿼드코어가 보편적인 용어로 자리 잡았을 뿐만 아니라. 모바일 기기 역시 멀티코어를 채용하고 있는 추세이다.

하지만 프로그래밍의 현실은 여전히 싱글코어 방식을 답보 하고 있거나, 멀티스레딩의 활용도가 현저히 떨어지는것이 사실이다.

본 논문에서는 병렬 프로그램 모델을 알아보고, 다중개체 처리에 적합한 모델을 제시한다.

본 논문은 다음과 같이 구성된다.

2 장에서 일반적 으로 사용되는 병렬처리 모델에 대해 알아 보고,

3 장에서는 직렬화된 디스패쳐 워커 모델에 대해 알아본다.

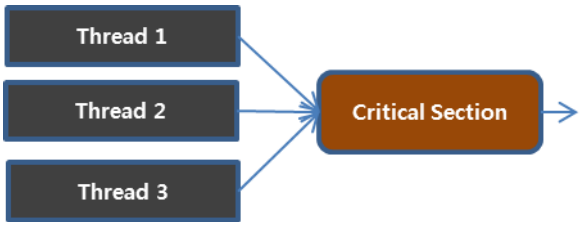
4 장에서는 성능평가의 결과에 대하여 설명하고

5 장에서 결론을 맺는다

2. 병렬 처리 모델

2.1 동기화 개체 사용 방식

Critical-Section, Semaphore 등의 동기화 개체를 사용하여 멀티 Thread 문제를 해결하는 방식이다. Lock, Unlock 인터페이스를 호출 하는 것으로 동기화가 가능 하기 때문에 사용이 매우 편리하다. 하지만 해당 동기화 영역이 자주 사용된다면 Race Condition 이 발생하여 멀티 Thread 의 장점이 퇴색 되는 경우가 많다. 또한 데드락이 발생되어 시스템이 멈추는 경우가 있는 것이 문제이며, recursive call 을 지원하지 않는 동기화 개체의 경우 self dead lock 상태가 발생하기 때문에 인터페이스를 2 중으로 만들어야 하는 문제가 발생 할 여지가 있다.

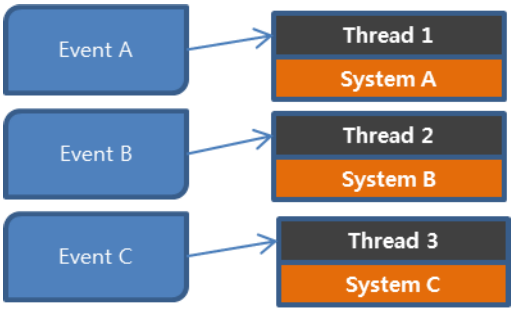


(그림 1) Race Condition

2.2 시스템별 Thread 할당 방식

위 2.1 방식이 병목현상으로 인해 시스템을 기능 별 혹은 사용자 임의의 기준으로 분류 하여, 분류된 시스템 별로 Thread 를 할당 하는 방식이다. 이 방식은 시스템간 이벤트 발생 비중을 균일 하게 맞추지 못하면 Thread 자원이 낭비되는 현상이 발생하게 되는 문제가 있으며, 각 시스템별 이벤트 loop 를 따로 작성해야 하는 번거로움이 따른다. 프로젝트의 유지보수 시점에 시스템을 추가할 경우 Thread 추가에 대한 부담이 발생 하게 되며, Context Switching 의 비용을 재계산 해야 하는 경우가 생긴다.

또한 시스템이 처리해야할 개체 수가 늘어날 경우 시스템별 스레드 수의 불균형 현상이 나타날 수 있기 때문에 대규모 사용자 처리에 부적합 한 모델이다. 이 문제를 해결 하기위해 Thread Pool 시스템과 같은 가변성 시스템을 도입 하고 있지만 o/s resource 의 낭비를 초래 하게 된다.

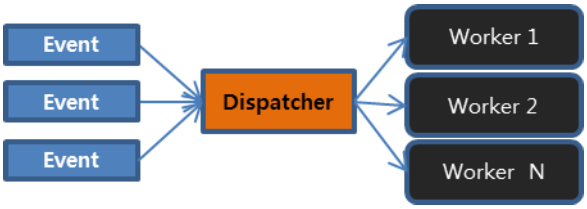


(그림 2) 시스템별 스레드 할당 방식

2.3 Dispatcher-Worker 모델.

Event 요청을 일괄 처리하는 Dispatcher 와 각 이벤트를 처리하는 Worker 로 구성되는 방식이다. 이때 Dispatcher 는 Singleton 으로 구현 하며. Dispatcher 와 Worker 는 각 하나의 Thread 와 Message Queue 를 가지게 된다.

워커 수를 가변적으로 운용 하여 시스템 성능을 최대한으로 끌어낼 수 있기 때문에. 네트워크 패킷 프로세싱등에 주로 사용 되며 대규모 사용자에 대한 응답을 처리하는데 적합하다. 하지만 Event 의 직렬화가 어렵고. 디스패쳐가 시스템 종속적으로 사용되기 때문에 시스템 확장성이 떨어지는 단점이 있다.



(그림 3) Dispatcher-Worker 방식

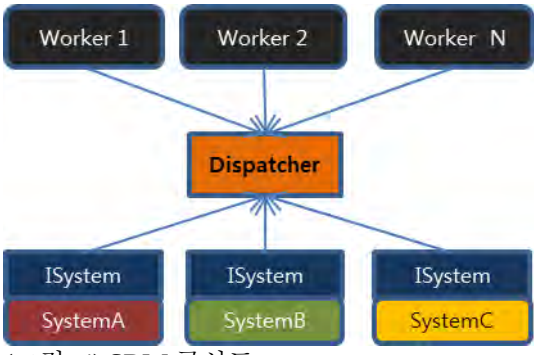
3. Serialization Dispatcher-Worker 모델.(이하 SDW)

3.1 Concept

프로젝트가 대규모화 될수록 병렬처리 모델은 Message 방식으로 귀결되는데 이때 가장 효율 적인 방식은 Dispatcher-Worker 모델이다. 하지만 Message 시스템은 A 개체와 B 개체가 상호작용을 하는 이벤트의 경우 직렬화가 깨지는 단점이 생기게 되며, 시스템마다 Dispatcher-Worker 를배치 하게되면 2.2 의 시스템별 Thread 할당 방식보다 o/s resource 낭비를 가져오게된다. 위와 같은 단점을 극복 할 수 있게 한다

3.2 Serialization Dispatcher-Worker 개요 #1/3

Dispatcher 는 슈퍼 클래스의 지위를 갖게 구성하게 되는데. Dispatcher 는 Worker 를 생성 하는 수량을 동적으로 조절 할 수 있게 하여 시스템 Initialize 시에 worker 들을 대기 시킨다. 이때 Diaptcher 는 Event Processing 을 위해 interface- ISystem 을 상속받은 각종 시스템을 등록 시켜준다.



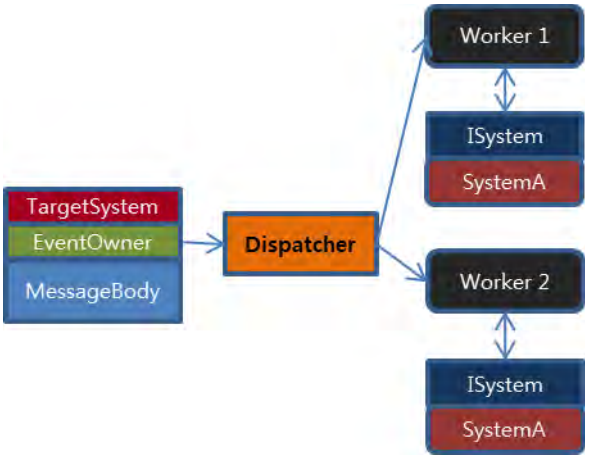
(그림 4) SDW 구성도

3.3 Serialization Dispatcher-Worker 개요 #2/3

Dispatcher 에 입력되는 이벤트 Message 는 일반적인 Message 와 구성이 다르다.

첫번째로 TargetSystem 값은 해당 Message 를 처리할 System 의 고유 ID 가 기록 되어야 하며, Message 는 하나의 TargetSystem 을 가리키도록 한다.

두번째로 EventOwner 값은 이 Message 의 소유권을 가지는 개체의 고유 ID 를 갖게 되며. EventOwner 는 N 개를 소유 할 수 있게 하여. Message 프로세싱에 있어서 다수의 개체에게 Transaction 를 할 수 있게 해준다.



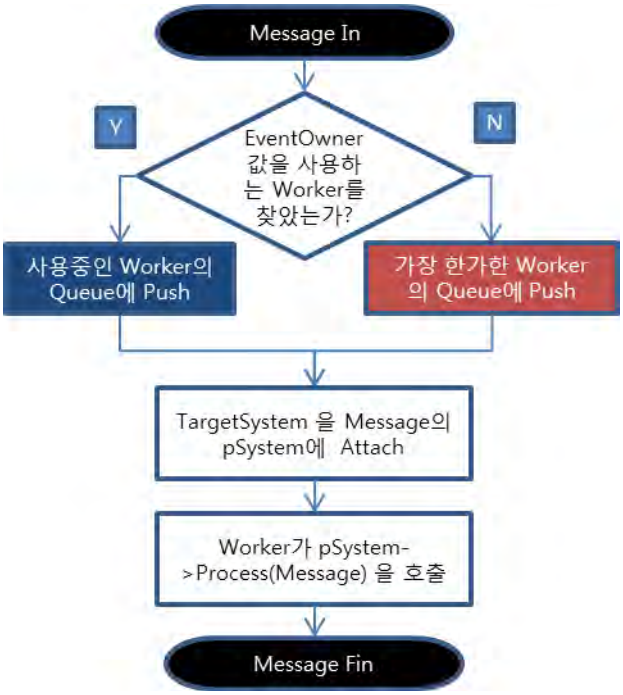
(그림 4) SDW 의 Message 처리 개념도

3.4 Serialization Dispatcher-Worker 개요 #3/3

Message 의 직렬화는 오직 EventOwner 값에 의해 결정 되며. 직렬화가 보장 되기 때문에, 다수의 Worker 가 동시에 같은 System 을 얻어 사용 하게 되더라도 Lock 을 최소화 할 수 있다.

```
Struct Message
{
ID_TYPE TargetSystemID;
Std::vector<Owner_ID> kContOwner;
Isystem *pSystem;
MessageBody kBody;
};
```

(그림 5) Message 의 실제 구현.



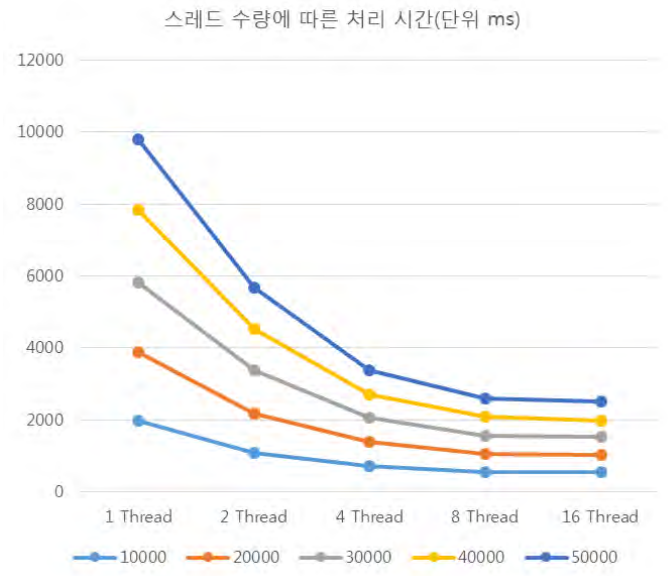
(그림 6) Message Serialization 과 Worker 에 System 을 Attach 하는 과정

4. 성능평가와 결과.

SDW 가 실제 프로젝트에 사용 될 수 있는 수준의 성능을 보여 주는지 테스트를 진행 하였다.

i7 3770K CPU(8Core with hyperthread) 를 기준으로 테스트 하였으며 Windows7 환경 에서 C++ 언어를 채택 하여 구현 하였다.

임의의 Message 를 발생 시킨 후, dispatcher 에 push 하고 worker 를 통과하여 프로세싱에 드는 시간을 측정 하였다. 시간 단위는 ms 를 사용 하였다.



(그림 7)Worker 수량에 따른 Message 처리에 소요되는 시간

Worker 가 1 개 일 경우는 싱글스레드 모델과 성능이 동일 하며. Worker 가 복수로 증가할 경우 성능이 비례하여 빨라짐을 확인 할 수 있다. SDW 의 성능은 cpu core 의 수에 영향을 받게 되는데. Worker 의 수량이 CPU Core 개수에 근접 하거나 높으면 성능 향상이 미미한 수준이 된다.

5. 결론

SDW 을 적용하면 ISystem 을 상속받은 어떠한 시스템이라도 Dispatcher 에 등록 하게되어 시스템 추가 비용이 매우 낮고. Hardware 의 성능을 한계까지 끌어낼 수 있다. 하지만 Message Owner 설정이 적절하지 않을 경우 Message 의 직렬화를 기대할 수 없거나. 모든 Message 가 하나의 worker 에 몰리게되어, 최악의 경우 single thread 시스템 처럼 동작하게 된다.

다수 개체간 이벤트가 많더라도 하나의 Thread 로 Message 가 몰리지 않게 하는 기법, 다수의 Message Owner 를 가진 Message 의 Dispatch 성능 향상등이 추후 연구 과제로 남아있다.