디자인 패턴 지향 다중 프로세서를 위한 실시간 스케줄링 시뮬레이터

이종현, 조현중
고려대학교 컴퓨터정보학과
e-mail : {momoline, raycho}@korea.ac.kr

A Design Pattern-Oriented Real-Time Scheduling Simulator for Multiprocessors

Chong-Hyeon Lee, Hyeonjoong Cho
Dept. of Computer and Information Science, Korea University

요 약

본 논문은 다중 프로세서에서의 다양한 실시간 스케줄링 기법을 지원하기 위해 객체 지향적으로 설계된 시뮬레이터를 제안한다. 다중 프로세서 기반 실시간 스케줄링의 특징을 반영한 설계 원칙에 따라 디자인 패턴을 적용하여 시뮬레이터의 재사용성과 확장성을 높였다.

1. 서론

최근 임베디드 시스템에서는 성능 향상과 비용 효율을 높이기 위해 다중 프로세서 아키텍처의 도입이 각광을 받고 있다. 이러한 동향에 발맞추어 다중 프로세서 플랫폼의 보급이 확대되었고, 특히 다중 프로세서에서의 실시간 스케줄링은 매우 중요한 연구 과제 중 하나가 되었다. Liu [1]는 1969 년 처음 다중 자원 스케줄링 문제를 정의한 이후로 많은 실시간 스케쥴링 기법들이 제안되어 오면서 이들 기법 간의 성능을 상세히 비교하기 위한 시뮬레이터는 필수 불가결하게 되었다. 본 논문은 2 장에서 실시간 스케줄링 기법들을 간략히 소개하고, 3 장에서 디자인 패턴의 필요성과 함께 디자인 패턴 기반 설계 원칙을 수립하고 적용된 디자인 패턴에 대해 자세히 기술한다. 그리고 4 장에서 결론을 맺는다.

2. 실시간 스케줄링

CPU 자원의 총 사용량이 한계치를 넘지 않은 상태에서 모든 태스크들의 데드라인을 충족시킬 때, 이를 최적 실시간 스케줄링이라 한다. 단일 프로세서일 경우 earliest-deadline-first (EDF)[2] 알고리즘에 의해 최적의 스케줄이 가능하지만, 다중 프로세서 기반일 경우 EDF 는 더 이상 최적해를 구하지 못한다.

Proportional fair(Pfair)[3]는 태스크들이 각 태스크의 Utilization Demand 에 비례하여 균형적으로 실행되도록 하는 기법으로 최적 다중 프로세서 스케줄링을 제공한다. 하지만 스케줄 결정을 위해 단위 시간 (time quantum) 마다 스케쥴러를 실행하는 방식을 사용하기 때문에 높은 스케줄링 오버헤드를 동반한다[4].

Zhu[5]의 boundary fair(BF) 기법은 타임 퀀텀 단위 대신 태스크의 주기를 스케줄이 필요한 위치로 정해 전체 스케줄링 포인트를 줄였다. BF 와 Pfair 는 같은 복잡도를 갖고 있지만, 태스크의 수가 100 개 미만일 경우에 한해 Pfair 에 비해 월등한 성능을 보인다.

T-N Plane-Based Scheduling[6]은 다중 프로세서의 실시간 스케줄을 시각적으로 표현하기 위해 T-L Plane 을 도입하여 태스크의 상태 변화를 추상화하고, T-N Plane 에 기반하여 BF 에 비해 오버헤드를 줄이기 위한 실시간 스케쥴링 기법을 제안하였다.

3. 디자인 패턴 지향 설계

디자인 패턴은 소프트웨어 개발 시 자주 발생되는 문제점들에 대한 해결책을 패턴으로 일반화하여, 소프트웨어 개발 시 이를 활용함으로써 개발 시간을 단축할 뿐 아니라 반복되는 문제를 억제할 수 있도록 해준다[7]. 디자인 패턴은 추상적인 정보를 기술하기 때문에 일단 한번 변형된 디자인 패턴은 더 이상 분석 및 추적이 어렵다. 따라서 디자인 설계 원칙을 구체화해 패턴의 설계방향을 일관성 있게 조정해 나갈 필요가 있다. 시뮬레이터의 설계원칙은 다음과 같다.

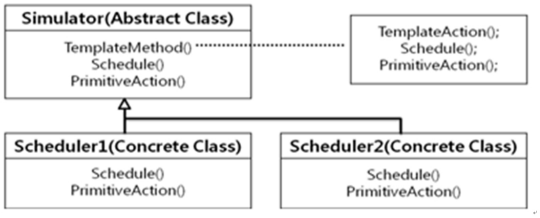
(1) 다양한 실시간 스케줄러를 쉽게 수용할 수 있어야 한다.
(2) 태스크와 프로세서의 개수에 유연해야 한다.
(3) 멀티프로세서 기반이므로 전통적인 스케쥴링 이벤트 이외의 이벤트도 고려해야 한다[6].
(4) Hollywood Principle[8]에 의거하여 의존성 부패를 방지한다.

표 1 을 보면 시뮬레이터를 개발하기 위해 사용된 디자인 패턴의 쓰임을 알 수 있다. 이 중 중요한 패턴 몇 가지를 설명한다.

<표 1> 시뮬레이터에 적용된 디자인 패턴

Factory Method	팩토리 메소드 패턴은 스케줄러를 생성하기 위한 인터페이스를 정의하는데, 어떤 스케줄러 객체를 생성할지 스케줄러 서브클래스가 결정하도록 함
Strategy	스케줄링 알고리즘을 각각 캡슐화 하여 클라이언트와는 독립적으로 알고리즘을 변경, 교환 할 수 있게 함
Observer	시뮬레이터의 상태가 바뀌면 옵저버인 스케줄러도 즉시 갱신됨
Template Method	스케줄링 알고리즘들의 공통 골격을 정의해 두고 서브클래스인 스케줄러들이 특정 단계만을 재정의해서 사용

Template Method 패턴은 알고리즘의 골격을 템플릿 메소드에서 정의하고 알고리즘의 여러 단계 중 일부만을 서브 클래스에서 구현하게 해주는 패턴이다. 그림 1 에서와 같이 템플릿 메소드가 스케줄링 알고리즘의 틀을 만들기 위해 사용되었다. 틀의 일련 단계 가운데 추상 메소드 Schedule() 이 정의되며, 이 메소드는 스케줄러마다의 특성을 반영하여 재정의된다. 이는 시뮬레이터의 기본 틀을 공유하고 각 스케줄링들의 고유 특징만을 구현할 수 있게 하기 위함이다.

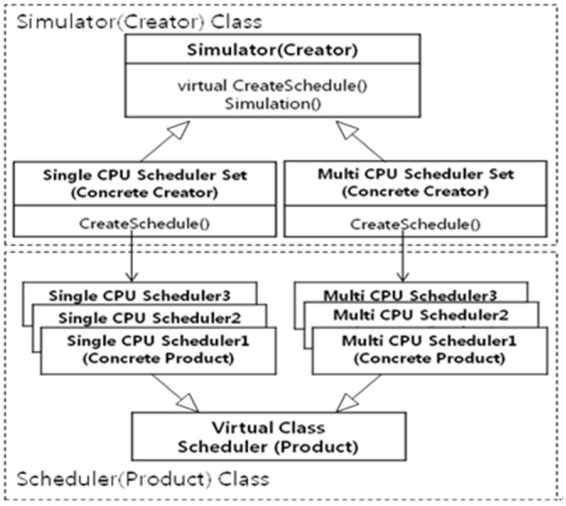


(그림 1) Template Method 패턴의 적용

스케줄러는 특성이 반영되어야 하는 부분만으로 구현되므로 다른 스케줄러를 쉽게 추가할 수 있고 시뮬레이터를 변경할 때도 코드의 중복이 없기 때문에 시뮬레이터 변경을 최소화할 수 있다.

Factory Method 패턴은 서브클래스가 어떤 클래스의 인스턴스를 만들지 결정하도록 정의하는 객체지향 디자인 패턴이다. 이 패턴은 불필요한 의존성을 없애기 위해 사용된다. 그림 2 에서와 같이 패턴 적용 전 CreateSchedule()은 new 를 이용하여 스케줄러 클래스의 인스턴스를 만드는 메소드였다. 이와 같은 구상 클래스 바탕의 코딩은 향후 수정해야 할 가능성이 높아지고 유연성이 떨어지게 된다. 이 문제는 직접 인스턴스를 만들 수 없는 추상클래스, 인터페이스로는 해결할 수 없기 때문에 스케줄러 인스턴스를 생성하는 부분을 캡슐화하여 문제를 해결한다. 클라이언트인 Simulator 에서는 새롭게 생성되는 스케쥴러와는 독립적으로 팩토리 메소드에서 생성되는 스케줄러를 참조해서 인터페이스에서 정의했던 시뮬레이션 하는 동작만 수행한다.

새로운 스케줄러를 추가할 때에는 프로세서 수와 같은 고유의 스타일에 맞게 서브클래스를 정의해야 한다. 같은 특성을 갖는 스케줄러들을 모아 캡슐화하고 팩토리 메소드인 CreateSchedule()을 추상 메소드로 정의한 후 서브클래스가 각 조건에 맞는 인스턴스 생성을 결정하도록 설계 하였다.



(그림 2) Factory Method 패턴의 적용

SchedulerSet 은 CreateSchedule() 을 오버라이트 하지만 Scheduler 는 추상 클래스기 때문에 Simulation()에서 실제로 어떤 구상 클래스에서 작업이 처리되고 있는지 알 수 없다. 즉, Simulator 와 Scheduler 는 서로 독립적으로 설계 되어 있다.

4. 결론

본 논문에서는 다중 프로세서를 위한 실시간 스케줄링 시뮬레이터의 개발에 디자인 패턴을 적용한 예를 소개 하였다. 멀티프로세서를 위한 실시간 스케줄러의 동작을 일반화하고 이들 특성을 반영한 디자인 패턴 기반의 유연한 시뮬레이터는 스케쥴러 개발과 시험, 평가 등에 유용하게 사용될 것이다.

참고문헌

[1] C. L. Liu. “Scheduling Algorithms for Multiprocessors in a Hard Real-time Environment” JPL Space Programs Summary, pp. 28-37, 1969

[2] J. Adnerson, V. Bud, U. C. Devi. “An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems” IEEE ECRTS, pp199-208, July, 2005

[3] S. Baruah, N.Cohen, C. G. Plaxton, D. A. Varvel. “Proportionate Progress: A Notion of Fairness in Resource Allocation” Algorithmica, vol.15, 1996

[4] A. Srinivasan, P. Holman, J. H. Anderson, S. Baruah. “The case for Fair Multiprocessor Scheduling” IEEE Workshop on Parallel and Distributed Real-Time Systems, pp.114.1, April, 2003

[5] D. Zhu, D. Mosse, R. Melhem. “Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary?” IEEE RTSS, 2003

[6] H. Cho, B. Ravindran, E. D. Jensen. “An Optimal Real-Time Scheduling Algorithm for Multiprocessors”,IEEE RTSS, 2006

[7] E. Gamma, R. Helm, R.Johnson, J.Vlissides. “Design Patterns: Elements of Reusable Object-Oriented Software” Addison-Wesley , Reading, Massachusetts, 1995

[8] J. Vlissides, “The Hollywood Principle” C++ Report, vol. 8, Feb. 1996