자바 병행 프로그램의 모니터링 시스템*

문세원○ 창병모
숙명여자대학교 컴퓨터과학과
{wonsein○, chang}@sookmyung.ac.kr

A Monitoring System for Concurrent Java Programs

Se-won Moon○ Byeong-Mo Chang
Dept. of Computer Science, Sookmyung Women's Univ.

요 약

자바 언어는 병행 프로그래밍을 위해 쓰레드를 제공한다. 자바 병행 프로그램은 레이스 컨디션이나 데드락에 의하여 사용자가 원하지 않은 값을 출력하거나 예기치 못한 오류를 발생시킬 수 있다. 이러한 문제는 프로그램의 전반적인 신뢰성 및 안정성에 악영향을 미칠 수 있다. 본 연구에서는 실행 중에 실시간으로 쓰레드와 동기화 객체의 발생과 처리 과정을 보여줄 수 있는 모니터링 시스템을 코드 인라인 기법을 기반으로 설계 개발하였다. 이 시스템은 사용자 옵션에 따라 관심 있는 쓰레드나 동기화 객체만을 추적할 수 있으며 실행 후에 쓰레드와 동기화 객체 관련 요약 프로파일 정보를 제공한다.

1. 서 론

자바는 병행 프로그래밍을 위해 쓰레드를 제공한다[6]. 자바 병행 프로그램은 여러 쓰레드의 병행 수행, 레이스 컨디션, 데드락 등에 의하여 사용자가 원하지 않은 값을 출력하거나 예외, 오류 등을 발생시킬 수 있다[3]. 이러한 문제는 프로그램의 전반적인 신뢰성 및 안정성에 악영향을 미칠 수 있으나 일반적으로 이해하기 매우 어렵다. 이러한 병행 프로그램의 효과적인 이해 및 디버깅을 위해 쓰레드의 생성 및 처리 과정을 시각적으로 보여줄 수 있는 모니터링 시스템이 개발되었다[2].

본 연구에서는 이러한 문제점을 해결하기 위해 실행 중에 실시간으로 사용자가 정의한 쓰레드 및 동기화 객체의 생성 및 처리 과정 등을 보여줄 수 있는 동적 모니터링 시스템을 설계 개발한다. 이 시스템은 [2]에서 개발된 기존 시스템과 달리 사용자 옵션에 따라 사용자가 관심 있는 쓰레드와 동기화 객체만을 실행 중에 추적해 볼 수 있으며 실행 후에 요약 프로파일 정보를 제공한다. 이렇게 하여 사용자는 관심 있는 쓰레드와 동기화 객체들만의 처리 과정을 살펴봄으로써 프로그램을 효과적으로 이해하고 디버깅할 수 있을 뿐만 아니라 전체 정보 모니터링으로 인한 실행 시간 부담을 크게 줄일 수 있을 것이다. 이러한 과정은 결과적으로 프로그램의 신뢰성 및 안정성 향상에 기여할 수 있을 것이다.

본 연구는 또한 동적 분석에 의한 실행 시간 부담을 줄이기 위해, 실행 시간에 부담을 주는 JVMPI(JVM Profiler Interface)를 사용하지 않고 코드 인라인 기법을 기반으로 설계하고 구현하였다.

본 논문의 구성은 다음과 같다. 2장은 시스템 설계에 관한 내용으로 시스템의 개요와 시스템 구조에 관해 설명한다. 3장에서는 구현을 위한 배경 연구와 현재까지의 구현 내용 및 실험 결과를 설명한다. 4장에서는 결론 및 향후 연구에 대한 소개한다.

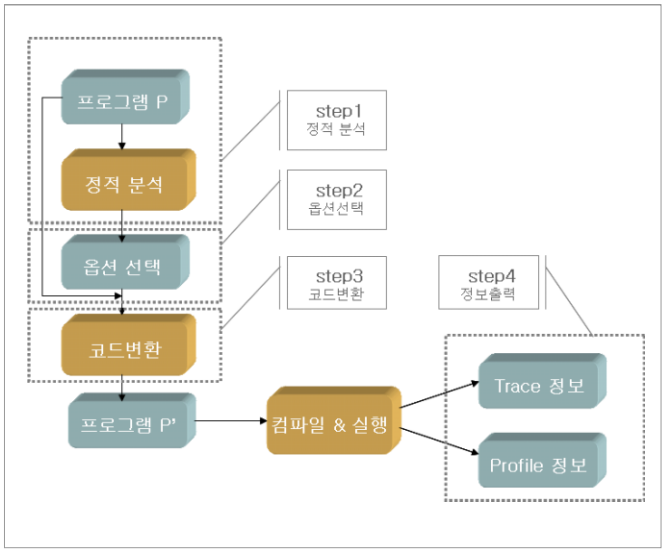
2. 시스템 설계

본 연구에서 개발할 시스템은 사용자가 자바 병행 프로그램의 모니터링을 위한 옵션을 선택할 수 있으며 이를 바탕으로 사용자는 관심 있는 쓰레드와 동기화 객체만을 선택하여 그 발생 및 처리 과정을 실시간으로 모니터링 할 수 있도록 설계하였다. 또한 실행 후에는 실행 중에 발생한 쓰레드와 동기화 객체들에 대한 요약 정보인 프로파일을 제공하여 사용자가 병행 프로그램의 처리 과정을 요약해서 살펴볼 수 있도록 하였다.

시스템의 전체적인 구조는 [그림 1]과 같으며 본 시스템은 크게 네 단계로 구성된다.

(1) 첫 번째 단계는 분석하고자 하는 프로그램을 정적 분석하여 쓰레드와 동기화 객체 등의 정보를 추출하는 단계로 이 정보를 이용하여 사용자에게 옵션 정보를 제

* 본 연구는 한국과학재단 특정기초과제 "정적 분석을 이용한 모바일 자바 프로그램의 효율적인 자원 사용을 위한 환경 연구"(R01-2002-000-003630-0)의 지원에 의해 수행되었음.



[그림 1] 시스템 구조

공한다.

(2) 두 번째 단계는 전 단계에서의 정적 분석 정보를 이용하여 옵션 선택 항목을 제공하면 사용자가 관심 있는 쓰레드 혹은 동기화 객체들을 선택하는 단계이다.

(3) 세 번째 단계는 사용자가 선택한 옵션의 값과 입력 프로그램을 가지고 해당 쓰레드와 동기화 객체들을 모니터링 할 수 있는 코드를 삽입하여 코드를 변환하는 단계이다.

(4) 네 번째 단계는 전 단계에서 변환된 프로그램을 컴파일 하고 실행시켜 모니터링 정보를 제공하는 단계로 변환된 프로그램이 실행되면서 프로그램의 실행 중에 사용자가 원하는 쓰레드 및 동기화 객체의 처리 과정을 실시간으로 볼 수 있다. 또한 프로그램 실행 후에 쓰레드와 동기화 객체에 대한 요약 정보인 프로파일을 제공한다.

3. 구현 및 실험

3.1 Barat을 이용한 코드 인라인

본 시스템은 실행 시간 부담을 줄이기 위해 코드 인라인 방법을 기반으로 구현한다. 코드 인라인 기법이란 실행될 프로그램 내에 모니터링 기능을 하는 코드를 삽입하고 변환된 프로그램을 실행하면서 삽입된 코드에 의해 실행 과정을 모니터링 하는 것이다[5]. 이 방법은 입력 프로그램과 모니터링 하고자 하는 성질 혹은 정책 등을 입력으로 받아 원래 프로그램에 모니터링 할 수 있는 코드가 삽입된 프로그램을 생성한다.

코드 인라인 방법은 각 응용에 따라서 필요한 부분만을 효과적으로 모니터링 할 수 있는 방법으로 그 효율성으로 인해 최근 들어 주목받고 있다. 본 연구에서는 이 기법을 기반으로 하여 사용자가 좀 더 효과적으로 실행 중에 쓰레드 및 동기화 객체의 생성 및 처리 과정을 모니터링 할 수 있도록 시스템을 개발한다.

본 연구에서는 쓰레드 관련 정보 추출과 코드 인라인을 위해 자바 컴파일러 전단부인 Barat을 이용하였다. Barat은 Java 프로그램의 정적 분석기 구현을 지원하는 컴파일러 전단부로 자바 소스 파일이나 클래스 파일을 입력받아 이름과 타입 정보를 포함하는 AST(Abstract Syntax Tree)를 구성한다[4].

이렇게 구성된 AST의 각 노드를 비지터(Visitor)라는 디자인 패턴을 이용한 트리 횡단 루틴을 이용하여 노드를 방문하면서 필요한 연산을 수행할 수 있다. Barat이 기본적으로 제공하는 비지터에는 DesendingVisitor, Out-putVisitor 등이 있다. DesendingVisitor는 AST의 모든 노드들을 깊이 우선 탐색하는 비지터이고, Output-Visitor는 모든 노드를 검색한 후, AST정보를 기반으로 소스코드를 다시 생성해주는 비지터이다. 이러한 비지터들의 메쏘드를 재정의함으로써 AST 노드를 방문 시 원하는 연산을 수행하도록 재구성할 수 있다.

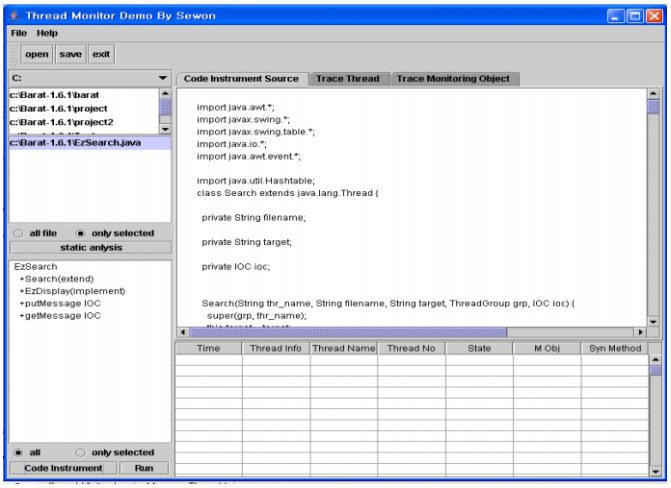
3.2 구 현

본 논문의 구현은 먼저 파일이나 패키지를 선택 한 후, 프로그램 내에 정의되어 있는 쓰레드와 동기화 객체들을 정적 분석한다. 이 단계에서 분석된 정보들은 다음 단계에서 제공되는 옵션 메뉴를 구성하는데 사용된다. 정적 분석기는 Barat에서 기본적으로 제공해주는 DescendingVisitor를 확장하여 구현하였다. [그림 2] 시스템 메인 화면의 “static analysis” 버튼이 이 기능을 수행한다.

전 단계에서 분석한 정적 정보를 기반으로 모니터링 할 정보에 대한 옵션 선택 메뉴를 제공한다. 옵션 선택 메뉴는 사용자에게 관심 있는 쓰레드나 동기화 객체만을 선택할 수 있게 함으로써, 전체 정보 모니터링으로 인한 실행 시간 부담을 크게 줄일 수 있다. [그림 2]의 시스템 메인 화면의 옵션 선택 버튼이 이 기능을 수행한다. 이 버튼을 이용하여 모든 쓰레드 및 동기화 객체들을 선택하거나 사용자가 관심 있는 쓰레드와 동기화 객체 선택할 수 있다.

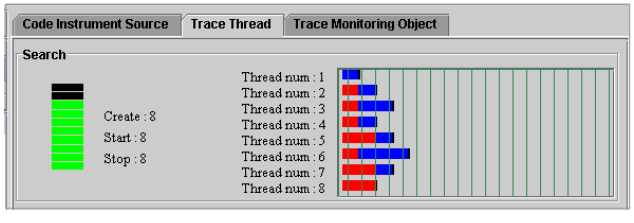
옵션 선택 단계에서 사용자가 선택한 옵션을 기반으로 입력 프로그램을 변환하는 코드 변환기를 구현하였다. 이 변환기는 Barat에서 기본적으로 제공하는 비지터인 OutputVisitor를 확장하여 선택된 쓰레드와 동기화 객체를 모니터링 하는 코드를 삽입하도록 구현하였다. 이 비지터는 두 번째 단계에서 사용자가 선택한 옵션에 따라 해당 쓰레드와 동기화 객체를 만날 때마다 입력 프로

그램에 모니터링 코드를 삽입한다. [그림 2] 시스템 메인 화면의 "Code Instrument" 버튼을 누르면 이 기능이 수행된다. 실행되어 변환된 코드는 오른쪽 화면에서 볼 수 있다.

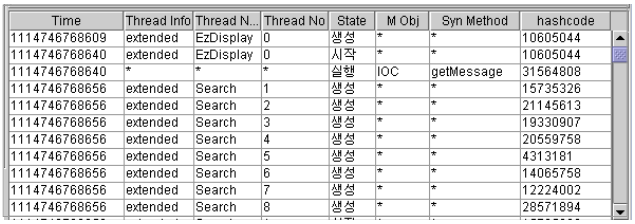


[그림 2] 시스템 메인 화면

[그림 2]의 샘플 프로그램은 인자로 받은 문자열과 매칭되는 파일들을 보여주는 애플릿 프로그램으로 java.lang.Thread를 상속받아 사용자가 구현한 Search 쓰레드 객체가 매핑되는 파일을 테이블 형식으로 보여주기 위해 IOC에 있는 동기화 함수인 getMessage, putMessage 함수를 호출하여 사용하는 프로그램이다.



(a) 쓰레드 정보 화면



Time	Thread Info	Thread N...	Thread No	State	M Obj	Syn Method	hashcode
1114746768609	extended	EzDisplay	0	생성	*	*	10605044
1114746768640	extended	EzDisplay	0	시작	*	*	10605044
1114746768640	*	*	*	실행	IOC	getMessage	31564808
1114746768656	extended	Search	1	생성	*	*	15735326
1114746768656	extended	Search	2	생성	*	*	21145613
1114746768656	extended	Search	3	생성	*	*	19330907
1114746768656	extended	Search	4	생성	*	*	20559758
1114746768656	extended	Search	5	생성	*	*	4313181
1114746768656	extended	Search	6	생성	*	*	14065758
1114746768656	extended	Search	7	생성	*	*	12224002
1114746768656	extended	Search	8	생성	*	*	28571894

(b) 프로파일 화면

[그림 3] 결과 화면

변환된 프로그램을 실행시키면 [그림 3]과 같은 결과 화면을 실시간으로 볼 수 있다. [그림 3]은 [그림 2]에 보여진 샘플 입력 프로그램에 대하여 [그림 3(a)]에서 생성된 쓰레드에 생성과 처리에 관련된 정보가 시각적으로 보여주고, [그림 3(b)]에서 쓰레드와 동기화 객체에 관련된 프로파일을 테이블 형식으로 보여준다.

4. 결론 및 향후 연구

본 연구에서는 병행 프로그램의 이해와 효과적인 디버깅을 위하여 사용자의 옵션 선택과 프로그램 변환을 통한 효율적인 쓰레드 모니터링 시스템을 설계 구현하였다. 특히 이 시스템은 사용자가 원하는 정보만을 얻을 수 있도록 옵션을 선택을 할 수 있으며 이를 이용하여 필요한 코드만을 삽입함으로써 시스템 부하를 크게 줄일 수 있었다.

현재까지 구현은 정적 분석, 프로그램의 변환 및 변환된 프로그램의 자동 실행 부분을 중심으로 구현하였다. 향후 연구에서는 실행 결과인 쓰레드와 동기화 객체의 모니터링 정보를 사용자가 보다 쉽게 이해할 수 있도록 시각화할 것이며 옵션 선택 항목을 보다 세분화하여 사용자가 더 자세히 필요한 정보만을 모니터링 할 수 있도록 할 것이다. 또한 기존의 시스템과 비교 실험을 통하여 개발된 시스템의 효율성을 실험적으로 보일 것이다.

5. 참고 문헌

[1] McGill Univ. Measuring the Dynamic Behaviour of Aspect Programs, OOPSLA '04

[2] Sun Microsystems, Design of JFluid: A Profiling Technology and Tool Based on Dynamic Bytecode Instrumentation, 2003.11

[3] Doug Lea, Concurrent Programming in Java(TM): Design Principles and Pattern (2nd Edition), Addusib Wesley

[4] Boris Bokowski, André Spiegel.Barat - A Front-End for Java Technical Report B 98-09 December 1998

[5] Ulfar Erlingsson, The Inlined Reference Monitor Approach to Security Policy Enforcement, PhD thesis, Department of Computer Science, Cornell University, 2003. Available as Technical Report 2003-1916

[6] http://java.sun.com/j2se/1.4.2/docs