simpleRTJ 자바가상기계의 메모리 관리 기법

양희재

경성대학교

Memory Management Scheme of the simpleRTJ Java Virtual Machine

Heejae Yang

Kyungsung University

E-mail : hjyang@star.ks.ac.kr

요 약

효율적 메모리 관리는 자바가상기계의 핵심 조건 중 하나이다. 자바에서는 새로운 인스턴스가 생성되거나 메소드가 호출될 때마다 메모리의 할당이 이루어진다. 반면 더 이상 사용되지 않는 인스턴스를 위한 메모리는 자동적으로 회수되며, 호출된 메소드가 복귀될 때마다 메모리도 회수된다. 본 논문에서는 특히 simpleRTJ 자바가상기계에서 적용된 메모리 관리기법에 대해 연구하였다. simpleRTJ는 모든 인스턴스의 크기를 동일하게, 또한 메소드 호출 시 생성되는 스택 프레임의 크기를 모두 동일하게 통일한다는 특징을 갖는다. 우리는 simpleRTJ에서 적용된 이 기법에 대해 상세히 고찰해 보며 이 기법의 성능에 대해서 정성적 분석을 하였다.

ABSTRACT

Efficient memory management is one of the most crucial requirement of Java virtual machine. In Java, memory is allocated everytime when a new instance of class is created or when a method is called. The allocated memory is freed when the instance is no longer used, or when the called method is returned. In this paper we have examined the memory management scheme applied to the simpleRTJ Java virtual machine. The simpleRTJ has such a distinguished characteristic in its memory management scheme that the size of all instances are forced to be the same and the size of stack frames of all methods be the same, respectively. We present in this paper the scheme thoroughly and analyze its anticipated performance qualitatively.

키워드

자바, 자바가상기계, 동적 메모리 관리, 쓰레기 수집기

Ⅰ. 서 론

자바가상기계는 자바 클래스 파일을 읽고 파일 속에 포함된 바이트코드를 실행하는 스택 기반의 가상 컴퓨터를 말한다. 자바가상기계의 내부 구조는 클래스 파일을 읽어들이는 클래스 적재기, 바이트코드를 실행하는 실행엔진, 메모리를 할당하고 또 해제시키는 메모리 관리기 등으로 이루어진다 [1].

메모리 관리기는 자바와 같은 객체지향형 언어의 지원을 위해 매우 중요한 부분이다. 새로운 객체가 생성될 때마다, 또는 객체 내의 메소드들이 호출될 때마다 메모리의 할당이 필요하게 된다. 반면 생성된 객체가 더 이상 필요없게 된다든지, 또는 호출된 메소드가 종료될 때마다 메모리의 해제가 이루어지게 된다. 자바에서는 더 이상 사용되지 않는 객체에게 할당된 메모리는 자동적으로 메모리 해제가 이루어지도록 하고 있다.

메모리 관리기의 구현에서 가장 중요한 것은 효율적인 메모리의 할당 및 해제 알고리즘의 개발이다. 요구되어지는 메모리를 신속히 할당해주고, 또한 더 이상 사용되지 않는 메모리를 신속히 해제하여 회수하는 기능은 효율적 자바가상기계의 구현을 위해 매우 중요하다.

이에 따라 다양한 형태의 메모리 할당 및 해제 알고리즘이 개발되었다[2]. 본 논문에서는 특별히

내장형 시스템을 위한 자바가상기계에서의 메모리 관리기에 대해 연구하였다.

내장형 시스템은 메모리의 양에 큰 제한을 받으므로 효과적인 메모리 할당 및 해제가 더욱 중요하다. 동시에 내장형 시스템에서는 프로세서의 성능도 높지 못하므로 복잡한 메모리 할당 및 해제 알고리즘의 적용은 비현실적이다.

따라서 많은 내장형 자바 시스템들이 각기 저마다의 고유한 메모리 관리 기법을 가지고 있다. 본 논문에서는 특히 simpleRTJ라는 내장형 자바가상기계에서 적용된 메모리 관리 기법에 대해 연구를 해 보았다. simpleRTJ[3]는 RTJ Computing 사에서 개발된 내장형 자바 시스템으로서, 수십 KB 정도의 매우 적은 런타임 메모리를 사용하며 8/16/32 비트 프로세서 환경에 모두 이식되어 사용되고 있다.

본 논문에서 특히 simpleRTJ 자바가상기계에 대해 연구하게 된 것은 이것이 가지고 있는 독특한 메모리 관리 기법 때문이다. simpleRTJ 에서 적용된 메모리 관리 기법에서 가장 중요한 부분은 모든 객체, 즉 인스턴스의 크기를 동일하게 만들고, 또한 메소드 호출 시 생성되는 메소드 프레임의 크기를 서로 동일하게 만든 것이다. 다시 말하면 특정 응용 프로그램에서 사용되는 클래스 인스턴스 중 가장 큰 크기를 갖는 인스턴스의 크기로 인스턴스 크기를 통일하였으며, 가장 큰 크기를 갖는 메소드 프레임의 크기로 메소드 프레임의 크기를 통일한 것이다.

이렇게 크기를 일치시키는 것으로 얻을 수 있는 장점은 크기가 일정하므로 메모리의 할당 및 해제가 쉽고 효과적으로 이루어질 수 있다는 것이다. 반면 단점으로는 적은 크기의 인스턴스나 메소드 프레임의 경우 자신에게 할당된 메모리를 모두 사용하지는 못하므로 메모리의 낭비가 있을 수 있다. 본 논문에서는 simpleRTJ 에서 적용된 메모리 관리 기법에 대해 보다 면밀히 분석해 보고 인스턴스나 프레임 크기를 한가지로 통일시키는 방식의 세부구조 및 기대효과를 알아본다.

본 논문의 구성은 다음과 같다. 2장에서는 일반적 자바가상기계에서 메모리 사용이 어떻게 이루어지는지에 대해 알아보며, 3장에서는 simpleRTJ에서 적용된 메모리 관리 기법을 알아본다. 4장에서는 이 기법의 장단점과 함께 기대되어지는 성능에 대해 정성적으로 분석해 보며, 5장에서 결론을 맺는다.

II. 자바가상기계의 메모리 사용

자바가상기계에서 메모리의 사용은 매우 중요한 부분을 차지한다. 그림 1은 자바가상기계 내부의 데이터 영역, 즉 메모리 사용에 대한 개념을 보여주고 있다.

그림에서 알 수 있듯이 메모리의 사용은 크게 네가지로 나뉘어진다. 먼저 클래스 영역이 있는데, 이곳에는 클래스들의 코드와 각종 상수들이 놓인다. 코드와 상수들은 프로그램 실행 시 변함이 없으므로 동적 메모리 관리와 무관하다. 내장형 시스템에서는 ROM에 탑재되어질 수도 있다.

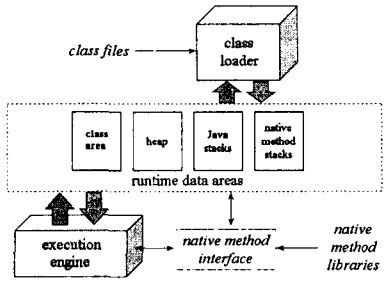

그림 1. 자바가상기계의 메모리 사용

둘째로 클래스의 인스턴스들을 저장하기 위한 힙(heap) 영역이 있다. 각각의 인스턴스들은 클래스 선언에서 정의된 필드들을 저장하기 위해 메모리 슬롯을 필요로 하는데, 각 슬롯들은 각기 하나의 필드를 저장한다. 상속성의 원리에 따라서 자신의 필드 외에 자신의 상위 클래스들이 가지는 필드들을 저장할 메모리 슬롯도 확보해야 한다. 프로그램 실행 시 새로운 인스턴스들이 계속해서 만들어짐에 따라 힙 영역이 계속 증가되어지게 되며, 더 이상 사용되지 않는 인스턴스들이 갖고 있는 슬롯들은 쓰레기 수집기(garbage collector) 에 의해 회수되어진다.

세 번째는 자바 스택 영역이다. 자바에서는 메소드가 호출될 때마다 스택 프레임(stack frame)이라는 데이터 공간이 만들어진다. 스택 프레임은 연산의 대상이 되는 오퍼랜드(operand)들이 놓여지게 되는 오퍼랜드 스택, 파라미터 전달이나 지역변수 저장을 위한 목적으로 사용되는 지역변수 배열(local variable array), 그리고 현재 실행 중인 명령을 지정하는 포인터 등으로 구성된다. 하나의 메소드가 호출될 때 마다 새로운 스택 프레임이 만들어지며, 이 메소드가 종료되고 원래 이 메소드를 호출했던 메소드로 복귀할 때 마다 프레임은 사라지게 된다.

마지막 네 번째는 네이티브 메소드 스택으로서 C 등으로 작성된 네이티브 메소드가 실행될 때 필요로 하는 스택 영역이다.

첫 번째의 클래스 영역은 클래스 적재 후 일정 크기로 고정되어 있으며, 네 번째의 네이티브 메소드 스택은 일반 컴퓨터의 스택 영역과 크게 다르지 않다. 따라서 본 논문에서는 특히 두 번째와 세 번째의 힙 영역 및 스택 프레임을 위한 메모리 사용에 대해서 연구해 보았다.

III. simpleRTJ 메모리 관리 기법

3.1 메모리의 배치

그림 2는 simpleRTJ에서 전체 메모리를 어떻게

사용하고 있는지를 보여준다. 그림에서 heap_start 와 heap_end 포인터 사이의 메모리가 전체 가용 메모리이며, 이 값은 초기에 정해진 값으로 고정된다.

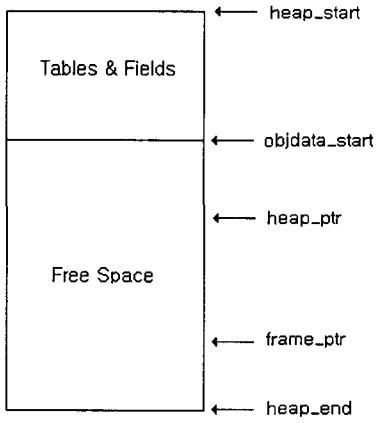

그림 2. simpleRTJ의 메모리 배치

전체 가용 메모리는 다시 두 부분으로 나뉘는데, 첫 부분은 각종 테이블과 정적 필드(static field) 들을 저장하는 목적으로 사용되며, 나머지 부분(free space)이 실제로 인스턴스들을 저장하기 위한 영역과 스택 프레임들을 저장하기 위한 영역으로 사용된다. 그림에서 objdata_start 포인터가 free space의 시작 번지를 가리키고 있다.

simpleRTJ 에서는 free space의 앞 쪽 부분은 인스턴스들을 위한 힙 영역으로 사용하고, 뒤 쪽 부분은 스택 프레임 영역으로 사용한다. 즉 그림에서 heap_ptr의 초기값은 objdata_start와 같고, 새로운 인스턴스가 만들어질 때마다 heap_ptr 값이 점차 증가된다. 따라서 heap_ptr은 힙 영역이 현재 차지하고 있는 최고점을 나타낸다.

반면 frame_ptr의 초기값은 heap_end 와 같다. 메소드 호출에 따라 새로운 스택 프레임이 만들어질 때마다 frame_ptr 값은 점차 감소된다. 따라서 frame_ptr은 스택 프레임 영역이 현재 차지하고 있는 최저점을 나타낸다.

프로그램이 진행됨에 따라 힙 영역과 스택 프레임 영역은 점차 확장되게 되며, 이 두 영역이 어느 한도 이상으로 접근하거나 또는 겹치게 되면 자동적으로 쓰레기 수집 과정이 일어나서 더 이상 사용되지 않는 메모리를 해제시키게 된다.

3.2 힙 영역의 할당

먼저 힙 영역의 할당에 대해 알아보자. 힙 영역은 각 클래스의 인스턴스를 저장하는 목적으로 사용되며, 주요 정보는 필드들이다. 어떤 클래스의 인스턴스가 만들어지면 상속성의 정의에 따라 그것의 상위 클래스들이 가지고 있는 필드들에 대해서도 힙 메모리가 할당되어져야 한다.

각 인스턴스는 헤더 부분과 데이터 부분으로 구성되는데, 헤더 부분은 이 인스턴스의 크기, 열쇠(lock)의 개수, 상태를 나타내는 플레그, 이 인스턴스가 속한 쓰레드의 id 등을 나타내며, 10바이트의 고정 크기를 갖는다. 데이터 부분은 실제 필드들이 저장되는 메모리 슬롯들로 이루어진다.

일반적인 자바가상기계와 구분되는 simpleRTJ의 가장 큰 특징은 이 데이터 부분의 크기가 일정하다는 것이다. simpleRTJ는 정적 클래스 적재만을 지원하는데, 따라서 인스턴스들이 필요로 하는 필드의 개수는 컴파일 시간에 미리 알 수 있다. simpleRTJ 에서는 이 인스턴스들 중에서 가장 많은 필드(상위 클래스의 필드도 포함)를 필요로 하는 인스턴스가 요구하는 메모리 양만큼을 모든 인스턴스들에게 동일하게 할당해 주고 있다.

새로운 인스턴스가 생성될 때는 그림 2의 objdata_start 로부터 시작하여 힙 영역 중에서 현재 사용되고 있지 않으며 그 크기가 원하는 힙의 크기와 같거나 큰 것이 있다면 그 영역을 할당해 준다(배열의 경우는 인스턴스와 달리 각기 다른 크기를 가지므로 힙 영역 내에서 크기 정보를 조회할 필요가 있다).

만일 frame_ptr이 가리키는 위치까지 검색을 해도 원하는 힙 영역이 발견되지 않는다면 쓰레기 수집기를 작동시켜 여유 메모리를 얻고자 시도한다. 그렇게 해도 원하는 영역이 발견되지 않는다면 MemErrNoHeap 예외 사건을 발생한다.

3.3 스택 프레임 영역의 할당

simpleRTJ에서 스택 프레임의 구조는 그림 3과 같은 이중 연결 리스트 형식을 취하고 있다. 가장 오른 쪽의 스택 프레임이 활성 스택 프레임에 해당되며, 새로운 메소드가 호출될 때마다 또다른 스택 프레임이 만들어져서 리스트의 제일 오른 쪽에 붙여진다.

그림 3에서 prev와 next는 이중 연결 리스트를

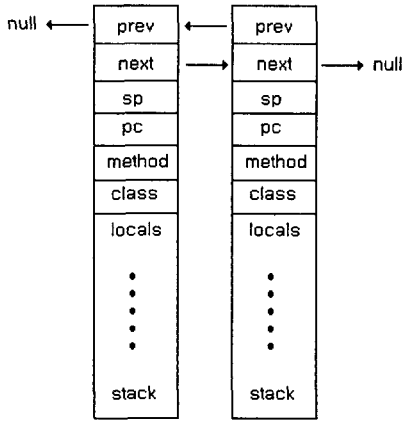

그림 3. 스택 프레임의 구조

이루는 포인터들이며, sp와 pc는 각각 스택 포인터와 프로그램 카운터에 해당된다. method는 이 스택 프레임을 사용하는 메소드를 가리키며, class는 이 메소드를 포함하는 클래스를 가리킨다. 이상의 내용이 스택 프레임의 헤더 부분에 해당되며, 이 크기는 24바이트로 일정하다.

스택 프레임의 나머지 부분은 데이터 부분으로서 이 부분이 지역변수배열 및 오퍼랜드 스택의 용도로 사용된다. 일반적인 자바가상기계와 구분되는 simpleRTJ의 가장 큰 특징은 이 데이터 부분의 크기가 일정하다는 것이다. 앞에서도 언급한 바와 같이 simpleRTJ는 정적 클래스 적재만을 지원하므로 지역변수배열 및 오퍼랜드 스택의 크기를 컴파일 시간에 미리 알 수 있으며, simpleRTJ에서는 지역변수배열과 오퍼랜드 스택의 크기의 합이 가장 큰 메소드가 요구하는 메모리 양 만큼을 모든 메소드 호출 시 동일하게 할당해 주고 있다. 지역변수배열은 할당된 데이터 부분 중 위에서부터 아래로 인덱스 번호가 부여되며, 오퍼랜드 스택은 일반적인 스택과 마찬가지로 아래에서부터 위로 인덱스 번호가 부여된다.

새로운 메소드가 호출될 때는 그림 2의 heap_end로부터 frame_size(고정길이) 만큼 뺀 후 그 프레임이 비어있는지를 확인하고 비어있다면 그 메모리를 할당해 준다. 그렇지 않다면 계속해서 frame_size 만큼 빼가면서 비어있는지의 여부를 알아보며, 만일 이런 선형탐색이 heap_ptr이 가리키는 위치까지 검색을 해도 발견되지 않는다면 쓰레기 수집기를 작동시켜 여유 메모리를 얻고자 시도한다. 그렇게 해도 원하는 영역이 발견되지 않는다면 OutOfMemory 예외 사건을 발생한다.

IV. 성능 예측 및 현재 연구진행 사항

simpleRTJ 자바가상기계의 메모리 관리는 모든 인스턴스의 크기와 모든 스택 프레임의 크기를 각각 한 가지로 통일시킨 것에 있다. 즉 가장 많은 필드(상위 클래스의 필드 포함)를 요구하는 인스턴스의 크기로 모든 인스턴스 크기를 통일하였으며, 메소드들 중에서 지역변수배열 및 오퍼랜드 스택 크기의 합이 가장 큰 메소드가 필요로 하는 스택 프레임의 크기로 모든 스택 프레임의 크기를 통일한 것이다.

이렇게 같은 크기로 통일하여 얻을 수 있는 가장 좋은 점은 원하는 크기의 메모리를 쉽고 빠르게 찾을 수 있다는 점이다. 일반적인 동적 메모리 할당 알고리즘에서는 할당 메모리의 양이 제각기 다르므로 알고리즘의 선택 여하에 따라 심각한 단편화 현상이 발생하였으며, 원하는 크기의 메모리를 발견하는데도 오랜 시간이 소요되었다. simpleRTJ에서는 모든 메모리 할당 단위가 균일하므로 이런 문제점이 발생하지 않게 된다. 현재 이 방법의 수치적 평가를 위해 다양한 프로그램 환경에서 실험을 진행하고 있는 중에 있다.

단점으로는 메모리의 낭비를 들 수 있다. 즉 가장 큰 크기의 인스턴스와 가장 큰 크기의 스택 프레임으로 통일하였기 때문에 대부분의 인스턴스 및 스택 프레임에서 실제로는 사용되지 않는 메모리가 존재하게 된다. simpleRTJ의 API 클래스 라이브러리에 대한 최근의 조사 결과 [4] 지역변수배열의 크기는 평균 1.9이며, 표준 편차는 1.27로 나타났다. 또한 오퍼랜드 스택의 크기는 평균 2.25, 표준 편차는 1.31이었다. 이 자료를 바탕으로 메모리의 낭비가 정량적으로 어느 정도인지를 밝히는 실험을 역시 진행 중에 있다.

V. 결 론

본 논문에서 우리는 simpleRTJ 자바가상기계에서 적용된 메모리 관리 기법에 대해 고찰해 보았다. 내장형 시스템과 같이 프로세서의 성능이 좋지 않은 환경에서 너무 복잡한 메모리 할당 알고리즘의 적용은 높은 추가부담을 끼치므로 좋지 않다. simpleRTJ 에서는 메모리 할당 단위를 일정한 값으로 고정함으로서 보다 고속의 효율적인 할당이 가능하도록 했다. 메모리가 실제로 사용되지 않을 수 있음으로서 낭비가 발생할 수 있지만, 일반적 할당법과 같이 외부 단편화에 의한 낭비가 거의 없게 되므로 메모리 사용도면에서도 좋은 성능을 발휘할 수 있다. 향후에는 실험을 통해 이 기법의 성능에 대한 정량적 분석을 행할 예정이다.

참고문헌

[1] 양희재, 자바가상기계, 한국학술정보, 2001년 3월, ISBN 89-5520-342-4

[2] P. Wilson et al., "Dynamic Storage Allocation: A survey and Critical Review," *Lecture Notes in Computer Sci.*, v.986, 1995

[3] RTJ Computing, *simpleRTJ: A Small Footprint Java VM for Embedded and Consumer Devices*, http://www.rtjcom.com

[4] 양희재, "simpleRTJ 클래스 파일의 형식 분석", 한국해양정보통신학회 2002 추계학술대회, 제6권 2호, 2002. 11., pp.373-377