내장형 자바가상기계를 위한 다중 고정크기 메모리 할당 기법†

김세영°, 지정훈, 양희재
경성대학교 컴퓨터공학과
sykim@conet.ks.ac.kr, jhji@conet.ks.ac.kr, hjyang@star.ks.ac.kr

Multiple Fixed-Size Memory Allocation Scheme for Embedded Java Virtual Machine

Seyoung Kim°, Junghoon Ji and Heejae Yang
Dept. of Computer Engineering, Kyungsung University

요 약

내장형 실시간 시스템에서는 메모리 관리시스템의 구현에 있어 메모리 단편화와 시간 결정성(determinism)의 문제를 해결하기 위한 방법 중의 하나로 고정크기의 메모리를 할당하는 기법이 사용되어진다. 내장형 자바가상기계에서도 객체를 관리하는 메모리 구조인 힙에 이를 적용하여 활용할 수 있으며 실제 구현으로는 simpleRTJ가 있다. 고정크기의 메모리 할당기법은 구현이 간단하기 때문에 시스템이 단순해지고 실행에서의 오버헤드도 작아지는 장점이 있다. 하지만 고정크기의 객체할당 방식은 가장 큰 객체의 크기를 이용하여 모든 객체를 할당하기 때문에 내부단편화를 발생시키는 단점이 있다. 본 논문에서는 내부 단편화를 최소화 하면서 고정크기 할단 기법의 장점을 최대한 이용할 수 있도록 하기위해 다수의 고정크기를 이용하여 객체를 할당하는 기법에 관해 설명하며 관련 실험을 통해 내부단편화 문제를 얼마나 해결할 수 있는지에 관해 기술한다.

1 서론

자바가상기계는 스택기반으로 동작하는 가상의 컴퓨터로서 클래스파일을 읽어 자바 프로그램을 실행하게 된다. 자바의 특징으로는 플랫폼 독립적이며 객체 지향을 지원하고 우수한 보안기능을 제공하고 있다. 이러한 특징으로 인하여 디지털 TV나 셋톱박스, 로봇제어와 같은 실시간성을 요구하는 내장형 시스템에 자바를 적용하려는 연구가 활발히 진행 중이다.

내장형 시스템에 자바를 적용하는데 있어 중요한 논점 중에 하나는 메모리 관리이다. 제한된 메모리 크기와 낮은 프로세서 성능은 복잡한 메모리 할당과 해제 알고리즘을 그대로 적용할 수 없게 하는 요인이 되며 이에 대한 연구도 많이 이루어진 상태이다.[1]

내장형 자바가상머신을 위한 메모리 할당기법 중에서도 고정크기의 메모리 할당기법은 알고리즘 및 구현이 단순하며 단편화(fragmentation)의 문제도 해결될 수 있으며 실시간 시스템의 중요 요소인 시간 결정성을 어느 정도 보장하고 있기 때문에 내장형 자바가상기계의 메모리 할당 방법의 좋은 대안이 되고 있다. 고정크기 할당 방법은 프로그램이 실행되기 전에 가장 큰 객체의 크기를 알아야 하기 때문에 내장형 자바가상기계의 구현 중에서도 정적으로 클래스 파일을 적제 하는 시스템에 의해 사용되며 이러한 구현으로는 simpleRTJ[2]이 있다.

하지만 모든 객체들이 가장 큰 객체를 기준으로 하는 고정된 하나의 크기로 할당되기 때문에 프로그래머가 자바코드를 작성할 때 클래스 설계가 좋지 않을 경우 심각한 내부단편화가 발생하는 문제가 발생할 수 있다.

본 논문에서는 이러한 고정크기 할당기법의 내부단편화 문제를 해결하기 위한 방안으로 기존의 하나의 고정크기 할당기법을 수정하여 다수의 고정크기를 정의하고 객체를 할당하여 하나의 고정크기 할당방식의 장점을 최대한 유지하면서 내부단편화를 최소한으로 줄이고자 하였다.

본 논문의 구성은 2장에서 관련연구로 가변크기와 고정크기 할당 방법이 각각 적용된 시스템에 관해 알아보고, 3장에서 다수의 고정크기를 이용한 할당기법에 대해 알아보고, 4장에서 그 성능에 대한 실험결과에 관해 분석한다. 마지막으로 5장에서 결과 및 향후과제를 기술한다.

† 이 논문은 2003년도 정보통신부 지원 정보통신기초기술연구 지원 사업에 의해 연구되었음.

2 관련연구

2.1 KVM

CLDC[3]는 네트워크 연결 능력이 있고, 적은 자원을 가진 다양한 기기들에게 적합한 자바 플랫폼을 정의하고 있으며 KVM은 CLDC에서 채택하고 있는 기본 자바가상기계이다.

KVM은 새로운 객체가 생성되면 힙 영역에 공간을 할당하고 프로그램에서 더 이상 참조되지 않는 객체는 힙 영역에서 제거하여 그 공간을 다른 객체에 의하여 재사용할 수 있도록 구성되어져 있다.

KVM에서는 힙의 공간 할당 방법으로 가변크기 메모리 할당 방식을 사용한다. 가변크기 메모리 할당 방식은 객체가 요구하는 각각의 서로 다른 크기의 메모리 공간을 할당해 주는 방식이다. 객체의 할당은 힙 공간의 비어있는 블록을 순차적으로 검색하여 요구하는 크기보다 크거나 같은 블록이 발견되면 메모리 할당이 이루어지게 되며 할당할 공간이 부족할 경우 가비지 컬렉션에 의하여 더 이상 사용되어지지 않는 객체의 공간을 수거하여 재사용할 수 있도록 한다.

가변크기 할당방식은 필요로 하는 크기의 메모리만을 할당하기 때문에 적은양의 메모리를 사용하는 내장형 시스템에서 자원의 효율적인 사용을 가능하게 한다. 하지만 할당과 수거 작업을 반복하면서 객체를 할당할 수 없는 빈공간이 생길 수 있

고, 객체 할당을 위한 빈(Free) 영역을 검색하는데 오버헤드가 발생하게 된다.

2.2 simpleRTJ

simpleRTJ는 수십 킬로바이트의 메모리 환경에서 자바가상머신을 동작시키기 위해 여러 가지 방법을 이용하고 있다. 우선 일반적인 자바가상머신에서 사용하는 동적 클래스 로딩방식을 사용하지 않고 ClassLinker라는 프로그램을 이용하여 정적 클래스 로딩 방식을 사용하고 있다. 또한 특정 운영체제에서 동일하게 실행하기위해 쓰레드 스케줄은 일정 시간 간격으로 스케줄 되는 라운드 로빈(Round Robin)방식을 채택 하고 있다. 외부 이벤트 처리는 폴링(polling)방식을 사용하고 있으며 비동기 이벤트 처리에 관해서는 아직 완전히 지원하고 있지 않다.

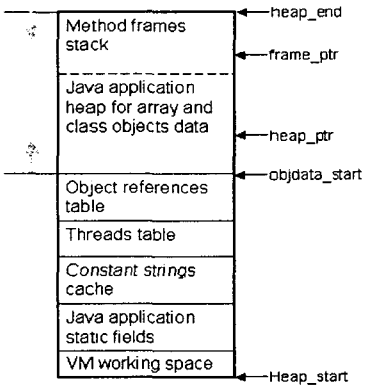

그림 1 simpleRTJ 메모리 구조

메모리 관리는 그림 1과 같은 메모리 구조에 의해 이루어진다. simpleRTJ에 의해 관리되는 메모리 영역은 heap_start(낮은 번지의 메모리 주소) 가 가리키는 메모리 영역에서 시작하여 heap_end(높은 번지의 메모리 주소)가 가리키는 부분 사이이며 동적으로 메모리가 할당/해제되는 부분은 objdata_start에서 heap_end가 가리키는 사이의 영역이 된다. 여기에는 객체와 메소드가 호출되면 생성되는 자바 스택프레임이 포함된다.

객체의 할당은 objdata_start에서 시작하여 heap_end를 향해 할당되며 가장 상위의 할당된 객체의 포인터는 heap_ptr이 가리키게 된다. 또한 자바 스택프레임의 경우는 heap_end에서 시작하여 objdata_start를 향하여 할당되며 frame_ptr은 다음 할당 가능한 비어있는 프레임 공간을 가리키게 된다.

객체 할당 크기는 가장 큰 객체의 크기로 고정되어 할당되며 이러한 방식이 가능한 것은 simpleRTJ가 정적 로딩방식으로 클래스를 로딩하기 때문이다. 자바 스택프레임의 경우도 전체 메소드중 가장 큰 지역변수배열과 오퍼랜드 스택의 크기를 취하여 모든 스택프레임 생성에서 동일한 크기를 할당하도록 되어있다. 스택프레임의 경우 가장 큰 프레임의 크기로 할당되더라도 메소드가 리턴되는 즉시 사라지게 되므로 메모리 낭비는 크지 않다. 하지만 객체의 경우 가바지 콜랙션이 발생하기 전까지는 객체들이 메모리에 유지되기 때문에 클래스의 설계가 나쁠 경우 내부단편화에 의해 메모리의 낭비가 커지게 된다.

3 다중 고정크기 메모리 할당

메모리 할당에 있어서 simpleRTJ에서와 같은 하나의 고정크기로 객체를 할당하는 방식의 문제점은 내부 단편화에 따른 메모리의 낭비이다. 하지만 내부단편화를 해결하기 위해 KVM에서 사용하는 것과 같은 가변크기 할당방법을 사용할 경우 외부 단편화 현상이 발생하고 할당을 위한 빈 공간을 검색하는데 시간이 소요된다. 그리고 시스템이 복잡해지기 때문에 오버헤드 또한 커지게 된다.

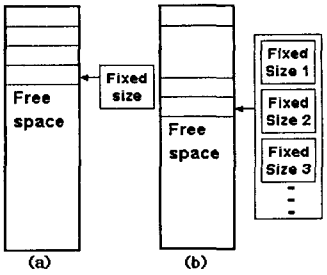

그림 2 고정크기 메모리 할당 구조

단일 고정크기 메모리 할당방식의 장점을 유지하면서 내부단편화를 줄이기 위한 방법으로 하나 이상의 고정크기를 사용하여 객체를 할당하는 다중 크기 메모리 할당 방식이 있다. 그림 2에는 단일 고정크기 메모리 할당 방법(a)과 다중 고정크기 메모리 할당 방법(b)을 보여주고 있다. 다중 고정크기 메모리 할당방식에서는 객체를 할당하기 위해 실제 객체의 크기와 각 고정 크기와 비교하여 가장 근사한 크기를 가지는 고정크기를 객체를 위해 할당하게 된다.

다중 고정크기 메모리 할당방식으로 객체를 할당하게 되면 내부 단편화를 줄일 수는 있지만 단일 고정크기 할당방식이 아니기 때문에 메모리의 빈 공간을 찾는 오버헤드가 발생하며 정의되는 크기의 개수가 증가하면 가변크기 할당방식과 비슷하게 되어 고정크기의 장점이 감소한다. 때문에 최대한 단일 고정크기 할당방식의 장점을 유지하면서 내부단편화를 최소화 하기위해서는 다중 고정크기 의 개수를 정의하는 방법과 고정크기 값을 결정하는 최적화 방법이 요구되며 이러한 조건들의 관계를 알아 보기위해 다음 장에서 그것의 영향을 분석해 본다.

4 성능평가

다중 고정크기 메모리 할당 방식의 성능평가를 위하여 simpleRTJ를 사용하여 실험을 하였다. simpleRTJ의 메모리 할당 방식은 이미지 파일에 포함된 클래스들 중 가장 큰 크기를 가지는 객체의 크기로 메모리를 할당하는 단일 고정크기 메모리 할당 방식을 사용한다. 본 논문에서는 실험을 위하여 단일 고정크기 메모리 할당방식을 사용하는 simpleRTJ의 구조를 변형하여 여러 개의 고정 크기를 가지는 다중 고정크기 메모리 할당 방식의 성능을 실험하였다. 고정 크기 메모리의 개수를 1개에서 15개까지 늘려가면서 가비지 컬렉션의 발생횟수, 가비지 컬렉션이 발생하기 까지 객체의 평균 할당 개수, 평균 내부단편화 크기, 힙 메모리 평균 접근 횟수 등을 측정하였다. 실험에서 측정한 내부단편화 발생량과 객체의 평균 할당 개수를 통해 자바가상기계의 메모리 사용의 효율성을 알아볼 수 있고, 가비지 컬렉션 발생횟수와 객체 할당을 위한 힙 메모리 평균 접근 횟수를 통해 시간 결정성을 보장해야 하는 실시간 내장형 시스템에서 자바가상기계의 성능을 측정할 수 있다.

실험을 위한 자바 테스트 프로그램은 4바이트에서 최고 60바이트의 임의의 크기를 가지는 30,000개의 객체를 생성하였다. 그리고 힙 영역에 객체를 저장하는 메모리 블록의 크기는 가장 작은 객체의 크기와 가장 큰 객체의 크기 사이의 영역을 동일한 크기로 분류하였다.

표 1 다중 고정크기 메모리 할당방식 성능평가

메모리 블록	GC 발생횟수	객체 평균 할당 개수	평균 내부단편화 (Byte)	heap 평균 접근횟수
1	51	588	17,618	1
2	41	731	10,766	9
3	39	769	6,523	16
4	39	769	5,687	19
5	40	750	3,211	24
6	39	769	3,249	27
12	41	731	1,603	39
14	39	769	223	41

표 1은 객체 할당을 위한 다중 고정크기 메모리 할당방법의 성능을 보여준다. 단일 고정크기 메모리 블록을 사용할 때보다 다수의 고정크기 블록을 사용 할 경우, 가비지 컬렉션 발생횟수와 평균 내부단편화 발생량에서 효율적인 것으로 나타났다. 하지만 객체 할당을 위하여 비어있는(Free) 메모리 블록을 찾는 작업에 따른 힙 영역 접근 횟수는 단일 고정크기 할당 방식을 사용할 때 가장 효율적인 것으로 나타났다.

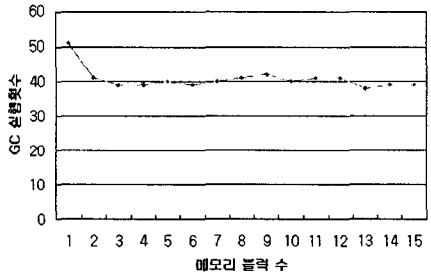

그림 3 블록 수에 따른 GC발생 횟수

그림 3은 다중 고정크기 할당방식에서 메모리 블록 수에 따른 가비지 컬렉션의 실행 횟수를 보여준다. 하나의 메모리 블록을 사용할 때에 비하여 두 개 이상의 메모리 블록을 사용할 때에 가비지 컬렉션이 실행되는 횟수가 많이 줄어든 것을 볼 수가 있다.

가비지 컬렉션 발생 횟수가 줄어든 것은 모든 객체를 가장 큰 객체의 크기로 할당하는 단일 고정크기 할당방법에 비해 다중 고정크기 할당방법은 실제 객체 크기에 근접한 메모리 블록을 할당할 수 있기 때문에 내부 단편화가 줄어들어 동일한 크기의 메모리에 더욱 많은 객체를 할당할 수 있고 메모리를 효율적으로 사용할 수 있다. 메모리 블록의 수가 3개 이상일 때 가비지 컬렉션 발생 횟수가 유사하게 나타나는 것은 블록의 수가 많을수록 내부 단편화는 줄어들지만 그만큼의 외부 단편화가 생기기 때문이다.

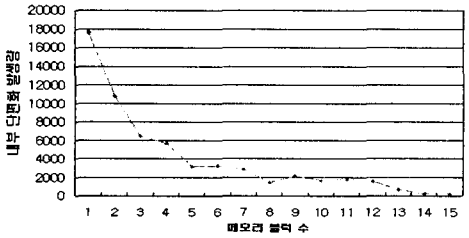

그림 4 내부단편화 발생량

그림 4는 메모리 블록 수의 변화에 따른 내부 단편화 평균크기를 보여준다. 내부 단편화 평균크기는 메모리 블록의 수를 증가시킬수록 감소되었다. 즉 실제 객체의 크기가 가장 근사한 고정 크기로 객체를 할당하기 때문에 메모리 블록 수가 많을수록 내부 단편화 평균 크기는 감소하게 된다.

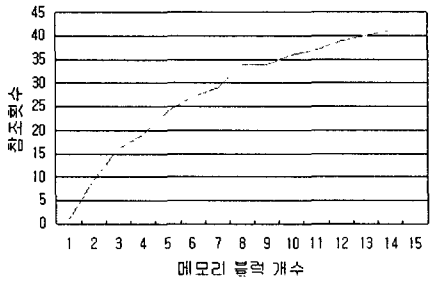

그림 5 블록 수에 따른 메모리 참조횟수

다중 고정크기 메모리 할당기법을 사용함으로 해서 힙의 빈 공간에 객체를 할당하기 위한 메모리의 참조횟수도 증가하게 된다. 그림 5는 이러한 참조횟수의 변화를 메모리 블록의 개수 변화에 따라 나타내고 있다. 메모리 블록의 개수가 증가함에 따라 그래프가 로그함수의 성질을 가짐을 알 수 있다.

실험 결과를 통하여 다중 고정크기 할당방법을 이용할 경우 내부단편화가 감소하여 전체 메모리 효율이 향상되었음을 알 수 있다. 하지만 할당을 위한 메모리 참조가 단일 고정크기에 비해 증가하며 메모리 효율의 증가율도 블록의 개수가 일정 개수 이상에서는 미비하기 때문에 고정크기 메모리 블록의 수를 효율적으로 유지하는 것이 중요함을 알 수 있다.

5 결론

본 논문에서는 내장형 자바가상기계를 위한 고정크기 메모리 할당 방법의 내부단편화 문제를 해결하기 위한 방법으로 다중 고정크기 할당방법에 관해 알아보고 실험을 통하여 효과를 검증해 보았다. 다중 고정크기 할당방법에서는 할당 시점에서 블록의 크기를 계산해야 하기 때문에 이에 따른 오버헤드가 발생하며 메모리 참조횟수가 증가한다. 앞으로 이에 대한 최적화 방법의 연구가 계속되어야 하겠다.

참고문헌

[1] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull; *The Real-Time Specification for Java*, Addison-Wesley, 2000.

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

[3] Sun Microsystems, Connected, Limited Device Configuration Specification, Version 1.0a, May 2000