논문 2006-43CI-3-3

임베디드 자바 가상머신에서의 가비지 컬렉션

(Garbage Collection on the Embedded Java Virtual Machine)

이 상 윤*, 김 상 욱**, 최 병 욱**

(Sang-Yun Lee, Sang-Wook Kim, and Byung-Uk Choi)

요 약

자바 언어는 그 객체지향성, 안전성, 유연성으로 인하여 현재 가장 널리 쓰이는 프로그래밍 언어의 하나가 되었으며, 자바 가상머신이 제공해주는 가비지 컬렉터로 인하여 프로그래머는 메모리 관리에 관한 많은 고민이 줄어들었다. 임베디드 환경에서 역시 자바는 강세를 나타내고 있으며 임베디드 환경의 특성을 반영한 가상 머신과 가비지 컬렉션 기법이 요구되고 있다. 본 논문에서는 힙이라고 불리는 메모리 영역을 크게 젊은 세대와 늙은 세대의 두 부분으로 나누어서 관리하며 각 세대는 그 특성과 요구사항에 적합하도록 각기 다른 기법을 적용한 가비지 컬렉터를 제안한다. 더불어 효과적인 가비지의 식별을 위한 쓰기 장벽과 2중 필터링 기법을 제안하고 있으며, 일반적인 방법으로 회수가 불가능한 순환적 구조의 가비지를 검출하여 회수하기 위한 이중 검사 기법을 제안한다. 제안하는 기법은 임베디드 환경의 요구사항인 객체의 빠른 할당, 동작의 실시간성, 모든 가비지의 회수, 단편화 제거, 높은 지역성 등을 모두 만족한다.

Abstract

The Java language has been established as one of the most widely used language due to its object-oriented programming, safety and flexibility and the garbage collection of the virtual machine has relieved programmers of many difficulties related to the memory management. In the embedded environment, Java is also prevalent, the virtual machine and garbage collector that takes into account the embedded environment is required. In this paper we manage the heap memory area by dividing into young generation and old generation, and we propose a garbage collector in which appropriate techniques are applied to each generation to utilize the different characteristics of each generation. Also, we propose the write barrier technique and double filtering technique for efficient garbage recognition, and double check method for determining and reclaiming the garbage with cyclic structure. The proposed method satisfies the embedded environment's requirements of fast object allocation, real time property, recollection of all the garbage, elimination of fragmentations and high locality.

Keywords: Garbage Collection, 가비지 컬렉션, 임베디드 자바, Java, 메모리

I. 서 론

자바(Java) 언어는 그 객체지향성, 안전성, 유연성으로 인하여 현재 가장 널리 쓰이는 프로그래밍 언어의 하나이다. 이러한 자바 언어로 작성된 응용 프로그램이 동작하기 위한 자바 플랫폼은 타깃 시스템의 메모리, CPU 성능, 전력, 응용 프로그램에 적합한 환경 구성을 위해 컨피규레이션(configuration)을 정의한다[4]. 컨피규레이션은 자바 가상머신(Java virtual machine)과 코어 API를 정의하고 있으며, <그림 1>처럼 J2EE/J2SE /J2ME로 구분된다.

이 중 J2ME(Java2 platform, micro edition)는 임베디드 환경을 위한 자바 기술로서 스마트폰, 핸드헬드 디바이스, PDA, 스크린폰, 셋탑 박스, 넷 TV와 같이 네트워크로 연결된 임베디드 혹은 모바일 기기에서 사용되며, 기기의 환경에 따라 다시 CLDC(connected limited

* 정회원, 한국전자통신연구원 임베디드S/W연구단
(Embedded S/W Research Division, Electronics and Telecommunications Research Institute)
** 정회원, 한양대학교 정보통신대학 정보통신학부
(Division of Information and Communications, Hanyang University)
※ 본 연구는 제주대학교를 통한 정보통신부 및 정보통신진흥원의 대학 IT연구센터 지원사업의 부분적인 지원을 받았음 (IITA-2005-C1090-0502-0009)
접수일자: 2005년12월9일, 수정완료일: 2006년5월8일

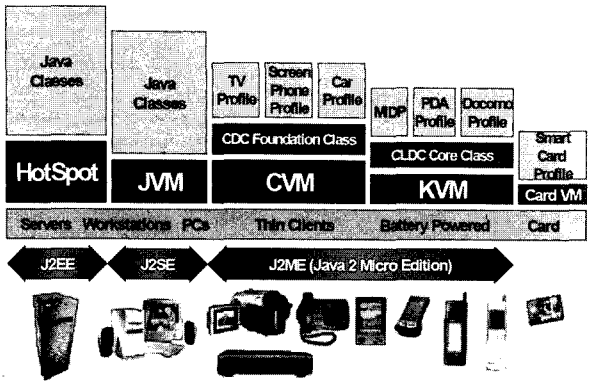


그림 1. 자바 플랫폼
Fig. 1. Java Platform.

device configuration)와 CDC(connected device configuration) 플랫폼으로 구분된다[4]. CLDC 플랫폼은 휴대폰이나 PDA와 같이 메모리나 CPU 성능이 극히 제한적인 기기에 탑재되어 사용되며, CDC 플랫폼은 홈 네트워크, 텔레매틱스, 셋탑 박스 등 좀 더 많은 리소스와 높은 성능을 갖춘 임베디드 시스템에서 주로 사용된다.

자바 언어로 작성된 소스코드는 컴파일되면 자바 바이트코드로 변환된다. 이러한 자바 바이트코드는 자바 가상머신에 의해 해석되고 실행된다. 즉, 자바 가상머신이란 컴파일된 자바 바이트코드와 실제로 프로그램의 명령어를 실행하는 마이크로프로세서 또는 하드웨어 플랫폼 간에 인터페이스 역할을 담당하는 소프트웨어를 의미한다[12]. 자바 가상머신으로 인하여 한 번 작성된 자바 바이트코드는 자바 가상머신이 동작하는 모든 플랫폼에서 동작이 가능해진다. <그림 1>에서 나타나듯이, 자바를 개발한 썬 마이크로시스템즈사는 CDC 플랫폼을 위한 가상 머신인 CVM과 CLDC 플랫폼을 위한 가상 머신인 KVM 등 두 가지 종류의 임베디드 자바 가상머신을 제공하고 있다.

자바 가상머신이 가지는 중요한 모듈 중 하나인 가비지 컬렉터(garbage collector)는 응용 프로그램이 더 이상 사용하지 않는 메모리상의 객체를 찾아서 회수하는 역할을 담당한다. 이 결과, 가비지 컬렉터는 프로그래머에게 메모리 관리에 관한 많은 고민을 줄여주었으며, 응용 프로그램이 보다 안정적으로 동작할 수 있게 해준다. 현재, 한국전자통신연구원에서는 임베디드 자바 가상머신을 개발하고 있으며 그 구조는 <그림 2>와 같다. 본 논문에서는 임베디드 자바 가상머신을 위한 효과적인 가비지 컬렉터를 제안한다.

임베디드 환경은 다음과 같은 특성을 갖는다. 첫째, 대부분의 임베디드 기기들은 메모리 용량이 작다. 둘째, 배터리로 동작되는 경우가 많으므로 전력 소비가 중요한 최적화 척도이다. 셋째, 임베디드 기기들은 일반 데스크탑 기기와 비교하여 CPU의 성능이 떨어진다. 이러한 고유의 특성들을 고려한 임베디드 자바 가상머신은 제한된 메모리의 임베디드 시스템에서 오랜 시간동안 동작하도록 요구됨에 따라 자바 객체를 효율적으로 생성하고 관리하는 것이 매우 중요하다. 즉, 동적 메모리 할당이나 가비지 컬렉션이 자바 가상머신의 성능에 매우 중요한 영향을 미친다[3, 7].

본 연구에서는 이러한 특성을 고려하여 다음과 같은 요구사항을 만족하는 임베디드 가비지 컬렉터를 제안한다. 첫째, 메모리 할당 시간이 짧아야 한다. 둘째, 가비지 컬렉션의 수행이 실시간성을 보장하여야 한다. 셋째, 모든 가비지는 가급적 빠른 시간 안에 모두 회수되어야 한다. 넷째, 제한적인 메모리를 효과적으로 사용하기 위하여 메모리 단편화가 제거되어야 한다. 다섯째, 객체 참조에 대한 지역성이 높아야 한다.

제안하는 기법은 힙(heap)이라 불리는 주어진 메모리 영역을 크게 젊은 세대(young generation)와 늙은 세대(old generation)의 두 부분으로 나누어서 관리하는 세대 가비지 컬렉션(generational garbage collection)을 기반으로 한다[6, 8]. 이는 객체의 대부분이 생성된 지 오래지 않아서 소멸된다는 특징을 반영한 것이다[1, 2, 9]. 객체는 상대적으로 크기가 작은 젊은 세대에 할당되어 대부분이 젊은 세대에서의 가비지 컬렉션을 통하여 소멸되고, 여전히 사용 중인 객체만이 늙은 세대로 이동하게 된다. 젊은 세대와 늙은 세대는 각 세대의 특성을 반영하기위해 서로 다른 가비지 컬렉션 기법을 사용하게 된다. 젊은 세대에서는 객체의 할당과 소멸이 빈번하게 발생하므로, 본 연구에서는 가비지 컬렉션의 동작에 부하가 적은 세미 스페이스 복사 컬렉터(semi-space copying collector)를 채택한다[11]. 반면, 젊은 세대와 비교하여 그 크기가 큰 늙은 세대에서는 실시간성을 보장하기 위해 보다 작은 영역으로 분할하여 관리하는 점진적인 가비지 컬렉션 기법(incremental garbage collection)을 채택한다. 이와 같이 힙을 여러 개의 작은 영역으로 분할하여 가비지 컬렉션을 수행하기 위해서는 먼저 해당 영역에서 여전히 사용 중인 객체들을 효과적으로 식별하는 방법이 필요하다. 본 논문에서는 이러한 효과적인 객체 식별 문제를 해결하기 위해 임베디드 환경에 적합한 효과적인 쓰기 장벽(write barrier)을 제안한다[10]. 또한, 일반적인 방법으로 회수가 불가능한

분할된 다수의 영역에 넓게 걸쳐서 존재하는 순환적 구조(cycle structure)의 가비지를 검출하고 회수하는 기법을 제안한다. 제안하는 가비지 컬렉션 기법은 앞서 논의한 임베디드 환경에서 요구되는 빠른 할당, 동작의 실시간성, 모든 가비지의 회수, 단편화 제거, 높은 지역성 등을 모두 만족한다.

본 논문의 구성은 다음과 같다. 제 II장에서는 한국전자통신연구원에서 개발 중인 임베디드 자바 가상머신의 구조와 기능을 설명한다. 제 III장에서는 임베디드 자바 가상머신에서 효과적으로 동작하는 가비지 컬렉터를 제안하고, 그 구조와 작동 방식을 설명한다. 제 IV장과 V장에서는 제안하는 가비지 컬렉터가 가질 수 있는 문제점을 정의하고, 이를 해결하기 위한 쓰기 장벽과 순환적 구조를 가지는 가비지의 검출 및 회수 기법에 대하여 논의한다. 마지막으로, 제 VI장에서는 본 논문을 요약하고 결론을 내린다.

II. 임베디드 자바 가상머신

본 장에서는 한국전자통신연구원에서 개발 중인 임베디드 자바 가상머신의 구조<그림 2>와 각 모듈이 가지는 기능을 설명한다.

자바 가상머신의 가장 핵심이 되는 실행 엔진(execution engine)은 컴파일된 자바 바이트코드를 해석하여 실행시키는 일을 담당한다. 실행 엔진의 구현 기법으로는 인터프리터, 자바 프로세서, JIT(just-in-time) 컴파일러, AOT(ahead-of-time) 컴파일러 기법 등이 있는데 개발 중인 임베디드 자바 가상머신에서는 인터프리터와 JIT 기법을 모두 이용한다.

동적 클래스 로더와 검증기(dynamic class loader & verifier)는 클래스 파일을 메모리로 읽어서 형식을 검사한 뒤 자바 가상머신의 내부에서 사용하는 구조에 따라 그 형태를 변환시킨다. 변환된 클래스의 정보는 클래스와 메소드 영역(class & method area)에 저장되어 실행 엔진에 의해 읽혀진다.

예외 처리기(exception handler)는 실행 시에 발생할 수 있는 오류에 대한 처리를 담당하며, 보안 관리자(security manager)는 자바 프로그램이 존재할 수 있는 범위를 정의하고 오용될 여지가 있는 자원에 대한 접근을 금지시키는 역할을 수행한다. 클래스 파일 검증 과정은 클래스 파일이 올바른 클래스인지 확인하는데, 이 검증 과정도 보안 관리자의 일부이다.

쓰레드 시스템(thread system)은 자바 객체간의 동기화와 한 개 이상의 쓰레드가 동시에 수행되도록 지원한다. JNI(Java native interface)는 자바 코드에서 C, C++ 또는 어셈블리어 등으로 작성된 프로그램이나 라이브러리 등을 사용할 수 있도록 해준다. 본 논문에서 다루고자 하는 가비지 컬렉터는 더 이상 사용되지 않는 객체가 점유하고 있는 메모리 공간을 다른 객체가 사용할 수 있도록 확보하는 일을 담당한다.

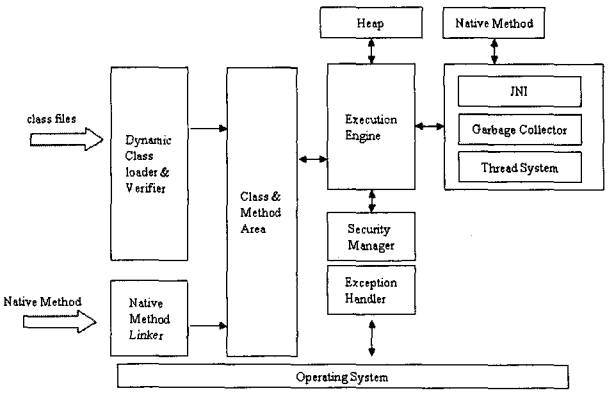


그림 2. 임베디드 자바 가상 머신
Fig. 2. Embedded Java Virtual Machine.

III. 제안하는 가비지 컬렉터

본 장에서는 제안하는 가비지 컬렉터의 구조와 작동 방식을 설명한다. 먼저, 제 3.1절에서는 제안하는 기법의 기반이 되는 세대 가비지 컬렉터에 관하여 논의한다. 제 3.2절에서는 세대 가비지 컬렉터의 젊은 세대에 적용되는 세미 스페이스 가비지 컬렉터에 관하여 설명한 다. 제 3.3절에서는 세대 가비지 컬렉터의 늙은 세대에 적용되는 점진적 가비지 컬렉터에 관하여 논의한다. 제 3.4절에서는 제안하는 기법이 가지는 장점을 설명하고 문제점을 지적한다.

3.1 세대 가비지 컬렉션

본 논문에서 제안하는 가비지 컬렉터는 <그림 3>에 나타난 바와 같이 힙을 젊은 세대와 늙은 세대의 비대칭적인 두 개의 세대로 분할하여 관리하는 세대 가비지 컬렉션에 기반하고 있다[6, 8]. 이러한 세대 가비지 컬렉션은 대다수의 객체가 할당된 후 짧은 시간 동안만 사용된다는 특성으로 인해 많은 수의 가비지 객체가 젊은 세대에서 회수된다[1, 2, 9]. 따라서 상대적으로 크기가 작은 젊은 세대에서의 가비지 컬렉션이 보다 빈번하게 발

생하여 소수의 객체만이 늙은 세대로 이동하므로 가비지 컬렉션에 의한 지연시간이 적고 전체 프로그램의 실행시간이 감소한다는 장점을 갖는다[5].

본 논문에서 제안하는 가비지 컬렉터는 는 <그림 3>에 나타난 바와 같이 힙을 젊은 세대와 늙은 세대의 비대칭적인 두 개의 세대로 분할한다. 새로운 객체는 상대적으로 크기가 작은 젊은 세대에 먼저 할당된다. 새로운 객체의 할당으로 인하여 젊은 세대가 가득 채워져서 메모리가 부족해지면 젊은 세대에서 가비지 컬렉션을 수행하여 메모리 확보를 시도한다.

응용 프로그램에서 사용 중인 객체는 가비지 컬렉션의 대상에서 제외되고, 해당 객체의 나이는 하나 증가한다. 객체의 나이가 미리 지정된 특정 나이에 도달하면, 그 객체는 늙은 세대로 전이된다. 이러한 작업의 반복으로 인하여 늙은 세대가 가득 차면 더 이상의 젊은 세대로부터 전이된 객체를 수용할 수 없게 된다. 이 시점에서는 늙은 세대에 대하여 가비지 컬렉션을 수행한다.

메모리상에 존재하는 객체 중에서 응용 프로그램이 사용하는 객체와 그렇지 않은 객체를 판별하기는 쉽지 않다. 따라서 본 논문에서는 응용 프로그램에서 사용 중인 객체의 정량적 식별을 위해서 도달 가능(reachable)하다는 개념을 이용한다. 본 논문에서는 도달 가능한 객체를 살아있는(live) 객체라 정의한다. 이러한 도달 가능한 객체의 식별은 루트 셋(root set)이라고 하는 영역에 상주되어 있는 객체들의 참조 값을 추적해가면서 이루어진다.

루트 셋은 수행 중인 프로그램에서 참조하고 있는 살아있는 객체들에 대한 참조 값들의 집합이며, 일반적으로 스택(stack)이나 전역 변수 내에 저장된다. 따라서 가비지라는 것은 메모리에 상주하고 있는 객체 중에서 루트 셋으로부터 도달이 불가능한 객체들을 의미하며, 죽은(dead) 객체라고도 표현된다.

이러한 세대 가비지 컬렉션은 대다수의 객체가 할당된 후 짧은 시간 동안만 사용된다는 특성으로 인해 많은 수의 가비지 객체가 젊은 세대에서 회수된다[1, 2, 9]. 따라서 상대적으로 크기가 작은 젊은 세대에서의 가비지 컬렉션이 보다 빈번하게 발생하여 소수의 객체만이 늙은 세대로 이동하므로 가비지 컬렉션에 의한 지연시간이 적고 전체 프로그램의 실행시간이 감소한다는 장점을 갖는다[5].

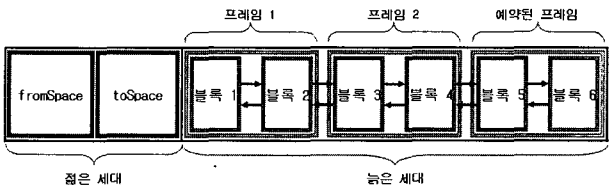


그림 3. 힙의 구조
Fig. 3. The architecture of heap.

3.2 젊은 세대에서의 가비지 컬렉션

세대 가비지 컬렉션에서는 가비지 컬렉션의 지연 시간을 감소시키기 위해서 젊은 세대의 크기를 늙은 세대의 크기보다 작게 설정한다. 세대 가비지 컬렉션에서 가장 좋은 효율을 보이는 젊은 세대와 늙은 세대의 크기 비율에 관한 연구가 있었지만, 주어진 환경에 따라 다른 결론을 보임에 따라 젊은 세대의 크기는 해당 응용 환경을 대상으로 하는 실험을 통해 얻을 수밖에 없다[2]. 본 연구에서는 젊은 세대 가비지 컬렉션 기법으로 구현이 쉽고 동작에 부하가 적은 세미 스페이스 복사 가비지 컬렉션을 채택한다[11].

<그림 3>에 나타난 바와 같이 젊은 세대는 fromSpace와 toSpace로 공간이 이등분 되어 있다. 그림상의 fromSpace에서 객체의 할당이 일어난다고 했을 때, toSpace는 사용되지 않는 빈 영역이 된다. 객체의 할당으로 인해 fromSpace가 가득 차게 되면 가비지 컬렉션이 수행된다.

컬렉션이 시작되면 fromSpace에 있는 객체들이 루트 셋으로부터 도달 가능한지 확인하여 도달 가능한 객체들은 toSpace로 복사한다. 이 때, 도달 가능한 객체들을 탐색하는 방식은 너비 우선 탐색이며, 이를 위하여 toSpace 공간을 큐로 활용한다. 젊은 세대의 가비지 컬렉터는 먼저 루트 셋에서 찾은 fromSpace 상의 도달 가능한 객체들을 toSpace로 복사한다. 그 후, toSpace에 복사된 객체들 중에서 다른 객체를 참조하는 객체들을 찾아서 자식 객체들을 toSpace로 복사를 하고 자식 객체가 없을 때까지 이를 반복한다. 모든 도달 가능한 객체들이 toSpace로 복사되었으면 fromSpace를 비우고 가비지 컬렉션을 완료한다. 가비지 컬렉션이 완료되면 fromSpace와 toSpace의 역할이 바뀌게 되어 toSpace가 이 후 객체의 할당을 담당한다.

세미 스페이스 복사 컬렉터는 객체의 연속적인 할당을 보장한다. 즉, 주어진 공간의 한 쪽 끝에서부터 객체들이 빈틈없이 채워지므로 단편화 현상이 없다. 또한, 함께 할당된 객체들은 함께 사용될 가능성이 높으므로 객체의 연속적인 할당은 참조의 지역성(locality of

reference)을 높일 수 있다[2]. 세미 스페이스 복사 컬렉터는 그 구현이 용이하고, 젊은 세대의 도달 가능한 객체만을 복사하면 되므로 동작에 부하가 적다는 추가적인 장점을 갖는다[5].

3.3 늙은 세대에서의 가비지 컬렉션

늙은 세대는 영역의 크기가 매우 크기 때문에 전 영역에 걸쳐 가비지 컬렉터를 수행하면 지연 시간이 길어 실시간 요구사항을 만족시킬 수 없다. 따라서 본 논문에서는 늙은 세대의 영역을 분할하여 복사 가비지 컬렉션을 부분적으로 수행하되, 이를 점진적으로 수행함으로써 실시간 요구사항을 충족시키고 더불어 복사 가비지 컬렉터의 장점을 모두 수용할 수 있는 기법을 제안한다.

<그림 3>에 나타난 바와 같이 늙은 세대는 균등한 크기의 블록이라는 단위로 분할되어 있고, 이 블록들은 이중 연결 리스트로 상호 연결되어 있다. 블록은 객체 할당의 단위가 된다. 이 블록들을 여러 개 묶어서 프레임을 형성하는데 이 프레임은 가비지 컬렉션의 단위가 된다. 힙이 초기화되는 시점에서 블록의 크기와 프레임의 크기는 환경 변수를 통해 결정할 수 있고, 이로써 가비지 컬렉터의 수행으로 인한 지연 시간의 조절이 가능하다. 각 블록은 할당이 발생한 순서대로 나이를 가진다. 즉, <그림 3>에서 블록1에서부터 블록4번순으로 할당이 발생한다면 가장 나이가 많은 블록은 블록1이 된다. 전체 프레임 중 한 프레임은 객체 할당을 위하여 사용하지 않고 가비지 컬렉션 수행을 위한 예비 공간으로 보존된다.

젊은 세대의 가비지 컬렉터는 지정된 나이가 된 객체를 늙은 세대로 할당시킨다. 늙은 세대의 할당은 블록의 한 쪽 끝에서 연속적으로 할당된다. 할당 요청된 객체의 크기가 현재 할당 중인 블록의 남은 여유 공간보다 큰 경우에는 다음 번호의 새로운 블록에 할당된다. 응용 프로그램이 시작해서 늙은 세대의 모든 블록에 객체가 가득 차기 전에는 가비지 컬렉터가 동작하지 않는다.

늙은 세대의 가비지 컬렉션은 젊은 세대로부터 일정한 나이에 도달한 객체를 늙은 세대로 이동할 때 그 공간이 부족하면 수행된다. 컬렉션의 대상으로 선정되는 프레임은 가장 나이가 많은 프레임이다. 그 이유는 나이가 가장 많은 프레임이 가장 많은 가비지 객체를 포함하고 있을 가능성이 높기 때문이다[15,16,17]. 가비지 컬렉터는 대상이 되는 프레임에서 루트 셋(root set)으로부터 도달 가능한 객체를 미리 예약된 빈 프레임으로 복사시킨다. 루트 셋은 수행 중인 프로그램에서 참조하고 있는 살아 있는 객체들에 대한 참조 값들의 집합이며, 일반적으로 스택(stack)이나 전역 변수 내에 저장된다. 모든 도달 가능한 객체들을 옮긴 후 프레임은 비워지고 다음 가비지 컬렉션을 위한 예비 공간으로 사용된다. 복사가 완료된 프레임의 블록의 나이는 가장 어린 나이로 설정되어 젊은 세대로부터의 다음 할당을 기다린다. 한 프레임의 가비지 컬렉션이 완료되면 멈춰있던 응용 프로그램의 수행이 재개된다.

힙이 초기화되는 시점에서 블록의 크기와 프레임의 크기는 환경 변수를 통해 결정할 수 있고, 이로써 가비지 컬렉션의 지연 시간은 조절이 가능하다. 제안하는 기법에서 객체는 블록에 연속적으로 할당된다. 만약, 할당 요청된 객체의 크기가 현재 할당 중인 블록의 남은 여유 공간보다 큰 경우에는 새로운 블록에 할당된다. 따라서 블록 간에 부분적인 단편화 현상이 발생하나 그 크기가 크지 않다. 가비지 컬렉션은 가장 나이가 많은 프레임으로부터 수행되는데 이는 해당 블록이 포함하는 객체들에게 소멸하기 위한 충분한 시간을 할당해 줌으로서 한 번의 가비지 컬렉션으로 가능한 많은 가비지를 회수하기 위함이다. 늙은 세대에서의 가비지 컬렉션에서 가비지 컬렉션의 대상이 되는 프레임을 젊은 세대의 fromSpace, 예약된 프레임을 toSpace로 볼 수 있으므로 세미 스페이스 복사 컬렉션과 동일한 개념으로 동작한다고 볼 수 있다. 따라서, 세미 스페이스 복사 컬렉션이 갖는 장점을 모두 수용한다.

3.4 논의 사항

제안하는 기법에서 젊은 세대와 늙은 세대 모두에서 객체들은 블록에 연속적으로 할당된다. 이는 한 객체가 할당되면 다음번 객체의 할당 시작 시점 표시를 위해 그 객체의 할당 시작 지점에 그 크기만큼 더해주는 연산만이 요구되므로 객체의 할당 위치 선정을 위한 별도의 연산이 없어 빠른 할당이 가능하다. 함께 할당된 객체들은 함께 엑세스될 가능성이 높으므로, 이러한 객체의 연속적인 할당은 참조의 지역성을 높인다[2]. 젊은 세대에서는 단편화 현상이 없이 객체가 할당되나, 늙은 세대에서는 할당 요청된 객체의 크기가 현재 할당 중인 블록의 남은 여유 공간보다 큰 경우에는 새로운 블록에 할당된다. 따라서 블록 간에 부분적인 단편화 현상이 발생하나 그 크기는 크지 않다. 가장 중요한 실시간 요구 조건에 대하여 젊은 세대는 그 크기가 크지 않고, 또

한 많은 수의 객체가 젊은 세대 내에서 소멸되기 때문에 가비지 컬렉션의 지연 시간이 실시간성을 만족시킨다. 상대적으로 크기가 큰 늙은 세대는 보다 작은 프레임 단위로 분할하여 가비지 컬렉션을 수행하기 때문에 실시간성을 보장할 수 있다. 이 때, 블록의 크기와 프레임의 크기는 힙이 초기화되는 시점에서 환경 변수를 통해 결정한다. 또한, 가장 오래된 프레임부터 가비지 컬렉션을 수행하기 때문에 수행 대상인 프레임 내에서 많은 객체가 소멸하며, 이동 되는 객체의 수가 적기 때문에 가비지 컬렉션의 수행 속도가 빠르다.

앞서 제안한 구조에서 가비지 컬렉션의 대상이 되는 하나의 프레임에 포함되는 살아있는 객체들을 효율적으로 판별하는 것이 매우 중요하다. 기본적인 방법으로서 루트 셋에서 시작하여 전체 힙에 상주하는 모든 객체들을 추적하는 방식을 고려할 수 있으나, 이는 매우 큰 소요시간을 요구한다. 본 논문에서는 제 Ⅳ장에서 이를 해결하기 위한 효과적인 쓰기 장벽에 관하여 제안한다. 또한, 제안하는 기법에서 가비지 컬렉션은 하나의 프레임만을 대상으로 하기 때문에 여러 개의 프레임에 걸쳐서 존재하는 순환적 구조의 가비지는 정상적인 방법으로 회수할 수 없다. 제 Ⅴ장에서는 이러한 순환적 구조의 가비지를 효율적으로 검출하여 회수하는 기법을 제안한다.

Ⅳ. 쓰기 장벽

본 장에서는 도달 가능한 객체의 빠른 식별을 위한 새로운 쓰기 장벽(write barrier)을 제안한다. 먼저, 제 4.1절에서는 쓰기 장벽의 정의와 쓰기 장벽이 필요한 이유에 관하여 설명한다. 제 4.2절에서는 제안하는 쓰기 장벽의 작동 방식과 이를 기반으로 도달 가능한 객체를 식별하는 방식에 대하여 설명한다. 제 4.3절에서는 제안하는 쓰기 장벽의 특징에 관하여 논의한다.

4.1 정의

쓰기 장벽이란 메모리상의 특정 영역에 쓰기 연산을 수행할 때, 이를 감지하여 특정 메모리 관리 연산을 수행시키는 일련의 메커니즘이다[10]. 쓰기 장벽이 필요한 이유는 본 연구에서 제안하는 가비지 컬렉션이 동작할 때 힙의 전체 영역을 대상으로 하지 않고 젊은 세대와 늙은 세대의 한 프레임이라는 일부 영역만을 대상으로 하기 때문이다. 즉, 힙의 한 영역만을 가비지 컬렉션하기 위해서는 그 부분에 속해있는 모든 도달 가능한 객체들만을 정확히 식별해낼 수 있어야 한다. 이러한 객체를 식별하기 위하여 가비지 컬렉션이 수행될 때마다 힙 상의 모든 객체를 추적하는 일은 상당한 부하를 유발시킨다. 따라서 한 영역의 객체가 다른 영역의 객체를 참조하는 경우가 발생하면 이러한 객체에 대해서는 쓰기 장벽을 통해 별도의 표시를 함으로써 가비지 컬렉션시의 부하를 줄일 수 있다.

본 연구에서 제안하는 가비지 컬렉터에서 쓰기 장벽은 두 가지 경우에서 필요하다. 첫 번째 경우는 젊은 세대에서의 가비지 컬렉션 시 늙은 세대에 있는 객체가 젊은 세대에 있는 객체를 참조하는 것을 파악하기 위한 것이다. 이는 다음과 같은 두 가지 상황에서 발생한다. 그 하나는 늙은 세대의 한 객체가 응용 프로그램에 의해서 젊은 세대의 객체를 참조하는 상황이다. 또 다른 하나는 젊은 세대의 한 객체가 젊은 세대의 다른 객체를 참조하고 있는데 젊은 세대에서의 가비지 컬렉션 수행 후 해당 객체의 나이가 지정된 나이에 도달하여 늙은 세대로 이동하면서 발생하는 상황이다. 쓰기 장벽이 필요한 두 번째 경우는 늙은 세대에서의 가비지 컬렉션을 수행할 때 컬렉션의 대상이 되는 프레임의 객체를 다른 프레임의 객체가 참조하는 것을 파악하기 위한 것이다. 이는 다음의 세 가지 상황에서 발생한다. 첫 번째는 늙은 세대의 한 객체가 응용 프로그램에 의해서 다른 늙은 세대의 객체를 참조할 때 발생한다. 두 번째는 늙은 세대의 객체가 젊은 세대의 객체로 참조를 가지는 상황에서 대상이 되는 젊은 세대의 객체가 젊은 세대의 가비지 컬렉션을 마친 후, 지정된 나이에 도달하여 늙은 세대로 이동되는 경우에 발생한다. 세 번째는 늙은 세대의 가비지 컬렉션으로 도달 가능한 객체가 다른 블록으로 복사되면서 발생한다.

4.2 해결 방법

본 연구에서는 가비지 컬렉션 시 도달 가능한 객체를 빠르게 검출하기 위해서 서로 다른 세대나 서로 다른 블록 내 객체들 간의 참조가 발생하면 쓰기 장벽을 이용하여 이러한 정보를 프로그램이 실행하는 동안 별도의 영역에 기록해둔다. 이 후 가비지 컬렉터는 이 별도의 영역만을 검사하여 가비지 컬렉션을 하고자 하는 대상 영역 내 모든 도달 가능한 객체를 식별할 수 있다.

제안하는 기법은 한 영역의 도달 가능한 객체를 식별하기 위해 이 단계 필터링을 사용한다. 제 1차 필터는 가비지 컬렉션의 대상이 각 블록에 대하여 그 블록내의 객체들이 참조하는 블록들을 식별한다. 제 1차 필터링

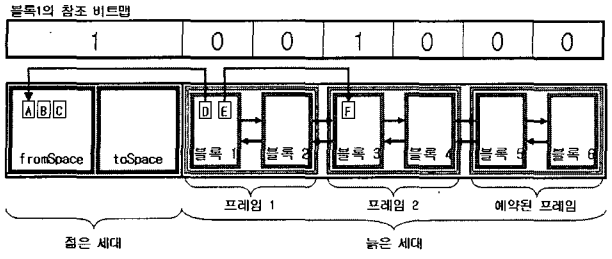


그림 4. 쓰기 장벽과 객체 참조 비트맵
Fig. 4. Write Barrier and object reference bitmap.

을 위하여 각 블록의 헤더는 <그림 4>와 같이 참조 비트맵을 가진다. 비트맵 내의 각 비트는 힙 내의 블록과 1대1 대응되며, 따라서 각 블록 내의 비트맵은 블록 수 만큼의 비트수를 갖는다. 프로그램의 실행 도중이나 가비지 컬렉션이 종료한 후, 한 블록A내의 한 객체가 다른 세대 혹은 다른 블록B 내의 객체를 참조하게 되면 블록A의 참조 비트맵은 참조가 되는 세대 또는 블록B와 대응되는 비트를 1로 설정한다. 가비지 컬렉터는 블록 헤더의 참조 비트맵을 검사함으로서 현재의 블록을 참조하는 객체를 가지는 모든 블록들을 검출해낼 수 있다. <그림 4>의 예에서, 블록1은 블록3과 젊은 세대 내의 객체들을 참조함을 볼 수 있다. 참조 비트맵은 한 블록A에서 다른 블록B로의 참조가 발생하면, 블록A내의 비트맵에서 블록B에 해당하는 비트가 즉시 1로 설정된다. 그러나 이러한 참조가 끊어지는 경우에 해당되는 비트를 0으로 리셋하지는 않는다. 이는 한 블록 내에 서로 다른 두개 이상의 객체가 하나의 세대 또는 블록을 참조 하는 경우가 발생 가능하기 때문이다. 즉, 응용 프로그램에 의해서는 참조 비트맵은 1에서 0으로 변경될 수 없다. 한 블록의 참조 비트맵은 그 블록을 대상으로 가비지 컬렉션이 수행될 때 갱신된다. 즉, 그 블록에서 도달 가능한 객체가 이동될 때, 그 객체가 참조하는 블록의 위치 값으로 갱신되는 것이다.

제 1차 필터에 의해 추적해야 할 블록이 결정되더라도 하나의 블록에는 많은 수의 객체가 존재할 수 있으므로 이 객체들을 모두 추적하는 것은 역시 상당한 시간이 소요된다. 따라서 블록 내에서 다른 블록을 참조하는 객체를 구체적으로 식별하기 위해 제안된 기법에서는 *<그림 5>와 같은 1제 2차 필터를 둔다. 제 2차 필터는 힙을 페이지라는 물리적 단위로 다시 세분화한 후 각 페이지마다 참조 상태를 저장하여 참조 상태에 따라 객체 추적을 달리하는 방법이다. 각 페이지의 참

* 비슷한 방법이 CVM[14] 에서도 사용되고 있다.

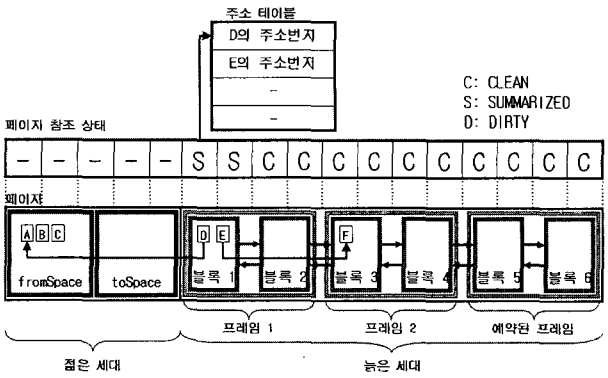


그림 5. 쓰기 장벽과 페이지 참조 상태
Fig. 5. Write Barrier and the status of page referencing.

조 상태 정보는 CLEAN, DIRTY, SUMMARIZE* 중의 하나이다. 참조 상태가 CLEAN 이면 그 페이지 내에서는 다른 세대 혹은 블록을 참조하는 객체가 전혀 없음을 뜻함, 그 페이지는 가비지 컬렉션 시 내부의 객체들을 추적하지 않아도 된다. DIRTY 이면 해당 페이지 내에 다른 세대 또는 블록을 참조하는 객체가 많음을 뜻하며 해당 페이지의 모든 객체들에 대하여 참조를 추적해야 한다. 참조 상태가 SUMMARIZE 이면 다른 세대 또는 블록으로의 참조가 있지만 특정 수 N개 이하임을 뜻한다. 여기서 N은 가상머신이 초기화 될 때, 설정이 가능한 수이며, 본 연구에서는 4라는 값을 기본값으로 사용한다. 즉, 다른 세대 또는 블록을 참조하는 객체의 수가 많지 않으므로, 해당 페이지의 객체를 전부 추적하지 않도록 다른 세대 또는 블록을 참조하는 객체의 주소를 별도의 주소 테이블에 저장한다. 따라서 참조 상태가 SUMMARIZE이면 이 주소 테이블에 기록된 객체만을 추적해보면 된다.

따라서 가비지 컬렉터는 제 1차 필터를 통해 해당 세대 또는 프레임이 어떤 블록에서 참조 되는지를 검사한 후, 참조가 있는 블록에 대해서는 제 2차 필터를 통해 정확히 어떤 객체가 도달 가능한지 식별할 수 있다. 따라서 가비지 컬렉션시의 도달 가능한 객체들을 매우 효과적으로 식별할 수 있다.

4.3 논의 사항

제안하는 기법은 가비지 컬렉션의 대상이 되는 블록의 도달 가능한 객체를 찾기 위하여 전체 힙에 존재하는 객체 추적의 부하를 응용 프로그램의 실행 중에 분산시킴으로서 가비지 컬렉션의 지연 시간을 줄인다. 이 단계 필터링에 필요한 정보를 저장하기 위한 추가적인 공간이 소요된다. 제 1차 필터를 위해서는 (전체 블록의

수+1)2/8 바이트가 필요하며, 제 2차 필터를 위해서는 (페이지의 수 * 페이지 참조 상태 정보의 크기) + (SUMMARIZED 페이지를 위한 주소 테이블의 크기) 만큼의 공간이 필요하다. 제안하는 기법은 메모리가 제한적인 임베디드 환경을 대상으로 하므로 전체 블록과 페이지의 수가 많지 않다. 따라서 추가 요구 공간은 제한적인 메모리 내에서도 충분히 수용할 수 있을 정도로 작다. 예를 들어, 힙의 크기가 5MB로 주어진 기기에서 젊은 세대의 크기를 1MB, 늙은 세대의 한 블록의 크기를 128KB로 설정하면 늙은 세대의 전체 블록의 수는 32개가 된다. 따라서 제 1차 필터를 위해서는 137바이트의 공간이 필요하다. 제 2차 필터를 위해 5MB의 힙을 512바이트의 페이지로 분할한다고 가정하였을 때, 전체 페이지의 수는 8192개이다. 각 페이지의 참조 상태는 1바이트에 저장될 수 있으며, SUMMARIZED 페이지를 위한 주소 테이블의 크기는 각 페이지 당 4바이트로 전체 주소 테이블의 크기는 32KB이다. 구현상의 추가 공간을 고려할 때 전체 필터링을 위해 필요한 공간은 48KB이다.

V. 순환적 구조의 가비지에 대한 검출과 회수

본 장에서는 점진적 가비지 컬렉션에서 정상적인 방법으로 회수가 불가능한 순환적 구조의 가비지를 효과적으로 검출하여 회수하는 기법을 제안한다. 먼저, 제 1절에서는 순환적 구조의 가비지를 정의하고, 이로 인한 문제점을 논의한다. 제 2절에서는 순환적 구조의 가비지를 효과적으로 검출하고 회수하는 기법을 제안한다. 제 3절에서는 제안하는 기법의 특징에 관하여 논의한다.

1 정의

순환적 구조(cycle structure)의 가비지란 참조를 통해 순환적 구조를 가지는 객체들이 루트 셋으로부터의 참조가 없어지면서 가비지가 되는 것을 의미한다[5]. 여기서 순환적 구조란 한 객체에서 그 객체가 참조하는 다른 객체를 재귀적으로 추적해가면 자기 자신으로 돌아오는 구조를 말한다. <그림 6>은 순환적 연결 리스트 D, E, F의 예를 나타낸 것이다.

이러한 순환적 구조의 가비지가 <그림 6>에서와 같이 두 개 이상의 여러 프레임에 걸쳐서 존재할 경우 이러한 가비지는 회수가 불가능해진다. 그 이유는 제안하는 기법에서는 가비지 컬렉터가 하나의 프레임만을 대상으로 하며, 도달 가능한 객체의 식별은 앞 장에서 설명한 쓰기 장벽을 통한 기록에 의존하기 때문이다. 즉, 가비지 컬렉션 대상인 프레임의 외부 블록 내의 가비지로부터 참조되어지는 객체는 여전히 도달가능하다고 판단되어 살아있는 객체로 간주되는 문제가 발생한다.

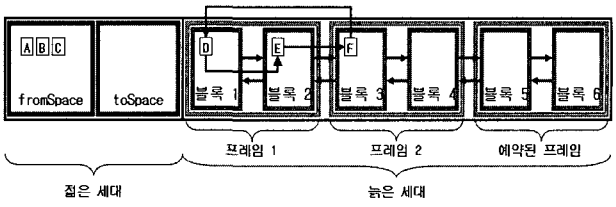


그림 6. 순환적 구조의 가비지
Fig. 6. garbage with cyclic structure.

이와 같은 실질적인 순환적 구조의 가비지는 메모리의 낭비를 유발시킨다. 순환적 구조의 가비지가 다수의 객체를 참조하고 있는 상황에서는 그 자손 객체들 모두 가비지 컬렉션의 대상에서 제외되기 때문에 메모리의 낭비 문제는 더욱 심각해진다. 이러한 가비지의 메모리 점유 문제로 인하여 객체 할당을 위한 공간이 부족해지고, 공간 확보를 위하여 자주 가비지 컬렉터를 동작시켜야 하는 부하가 발생한다. 최악의 겨우, 메모리 부족으로 인하여 시스템이 비정상적으로 종료하게 될 수 있다[5].

2 해결 방법

순환적 구조의 가비지는 힙의 어느 공간에서든 발생할 수 있다. 그러나 젊은 세대에까지 걸쳐서 존재하는 순환적 구조의 가비지는 젊은 세대에서의 가비지 컬렉션을 통해 모두 늙은 세대로 이동된다. 따라서 늙은 세대에서의 가비지 컬렉션에서 순환적 구조의 가비지를 검출하고 회수하여야 한다.

제안하는 기법에서는 먼저 순환적 구조를 가지는 가비지를 식별하기 위해 모든 프레임에 대한 가비지 컬렉션이 완료된 시점에서 주기적으로 전체 힙 영역을 대상으로 모든 도달 가능한 객체들을 추적한다.

이후의 늙은 세대에서의 가비지 컬렉션은 앞서 표기된 이 정보와 다른 프레임으로부터의 참조 정보를 모두 고려하여 가비지를 회수한다. 즉, 실제 루트 셋으로부터 도달 가능한 객체들만을 미리 표기해 두므로 순환적 구조의 가비지를 성공적으로 회수할 수 있게 된다.

전체 힙 상의 도달 가능한 객체를 추적하는 일은 모든 프레임에 대한 가비지 컬렉션이 완료된 시점에서 주기적으로 실행된다.

객체를 추적하면서 도달 가능한 객체는 그 객체가 가지는 헤더 내 하나의 비트에 표기를 한다.

이제 첫 번째 프레임에 대한 가비지 컬렉션이 실행되면 가비지 컬렉터는 쓰기 장벽을 통해 기록된 정보로부터 도달 가능한 객체를 식별하는데, 이 때 그 객체의 헤더를 조사하여 표기가 되어 있는지를 검사한다.

만약 한 객체가 다른 블록의 객체로부터 참조되어지지만 헤더의 표기가 없다면 순환적 구조의 가비지에 해당되므로 예약된 프레임으로 이동이 되지 않는다. 이러한 이중 검사를 통해 도달 가능하다고 식별된 객체는 예약된 프레임으로 이동하면서 헤더의 표기를 초기화한다.

3 논의 사항

순환적 구조의 가비지가 힙 상에 오래도록 존재하는 것은 메모리의 낭비를 유발시킬 뿐만 아니라, 할당 공간 확보를 위한 가비지 컬렉션을 보다 자주 수행시킴으로서 시간적 부하도 유발시킨다. 이러한 순환적 구조의 가비지를 검출하기 위하여 제안하는 기법도 수행상의 부하가 존재한다.

그러나 전체 힙을 대상으로 가비지 컬렉션을 하는 것이 아니라 단지 도달 가능한 객체들만을 식별하는 과정만 수행하므로 그 부하는 상대적으로 작으며, 따라서 제안된 가비지 컬렉터의 실시간성의 보장을 저해하지는 않는다.

이중 검사가 수행되는 프레임이 가지는 객체들의 참조 정보는 이중 검사가 진행 될수록 헤더에 표기 하는 시점과 시간적 차이가 있으므로 변하게 된다. 그러나 표기하는 시점에서 가비지였던 객체가 다시 도달 가능해지는 경우는 없으므로 제안하는 기법의 안전성은 보장된다.

VI. 결 론

본 논문에서는 임베디드 환경의 제약 사항을 고려한 가비지 컬렉터를 제안한다. 본 논문에서 제안하는 가비지 컬렉션 기법은 힙을 크게 젊은 세대와 늙은 세대의 두 부분으로 나누어서 관리하는 세대 가비지 컬렉션에 기반 하고 있으며, 젊은 세대에 세미 스페이스 가비지 컬렉터를, 늙은 세대에 점진적인 가비지 컬렉터라는 각기 다른 기법의 적용한다. 제안하는 기법은 연속적인 할당으로 인하여 빠른 할당과 객체 참조의 지역성을 만족하며, 젊은 세대의 크기 및 블록의 크기와 하나의 프레임이 가지는 블록 수를 변경함으로서 지연 시간이 조절가능하고 따라서 실시간성을 보장한다. 또한, 도달 가능한 객체 식별을 위한 효과적인 쓰기 장벽과 이 단계 필터링 기법을 제안하고 있으며, 순환적 구조의 가비지를 검출하여 회수하기 위한 이중 검사 기법을 제안한다. 이로 인하여 모든 가비지의 정확한 회수가 가능하다. 향후 연구로는 제안하는 기법의 최적의 효율을 위한 각종 파라미터의 설정에 대한 실험을 수행하고, 제안하는 기법과 기존의 기법들에 대한 성능 평가를 수행할 예정이다.

참 고 문 헌

[1] D. A. Barrett and B. G. Zorn, Using lifetime predictors to improve memory allocation performance, In Proceedings of SIGPLAN Conference on Programming Languages Design and Implementation(PLDI), Vol. 24, No. 7, pp. 187-196, June 1993.

[2] S. M. Blackburn, P. Cheng, and K. S. McKinley, Myths and reality: The performance impact of garbage collection, In Proceedings of International Conference on Measurement and Modeling of Computer Systems, pp. 25-36, June 2004.

[3] C. J. Cheney, A non-recursive list compacting algorithm, *Communications of the ACM,* Vol. 13, No. 11, pp. 677-678, Nov. 1970.

[4] Sun Microsystems, Java2 Platform, Micro Edition, Connected Device Configuration(CDC), http://java.sun.com/products/cdc/index.jsp 2005.

[5] R. E. Jones and R. Lines, Garbage collection: Algorithms for automatic dynamic memory management, Wiley, Chichester, July 1996.

[6] H. Lieberman and C. E. Hewitt, A real-time garbage collector based on the lifetimes of objects, Communications of the ACM, Vol. 26, No. 6, pp. 419-429, 1983.

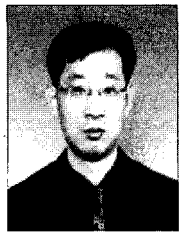
[7] W. Liu, Z. Chen, and S. Tu, Research and analysis of garbage collection mechanism for real-time embedded java, In Proceedings of International Conference on Computer Supported Cooperative Work in Design, pp. 462-468, May 2004.

[8] D. M. Ungar, Generation scavenging: A non-disruptive high performance storage reclamation algorithm," ACM SIGPLAN Notices, Vol. 19, No. 5, pp. 157-167, Apr. 1984.

[9] B. G. Zorn, Comparative performance evaluation

of garbage collection algorithms, PhD thesis, University of California at Berkeley, Technical Report UCB/CSD 89/544, Mar. 1989.
[10] B. Zorn, Barrier methods for garbage collection, University of Colorado, Technical Report CU-CS-494-90, Nov. 1990.
[11] C. J. Cheney, A non-recursive list compacting algorithm, Communications of the ACM, Vol. 13, No. 11, pp. 677-678, Nov. 1970.
[12] Tim Lindholm and Frank Yellin, The JavaTM Virtual Machine Specification (second edition). Addison-Wesley. 1999.
[13] G. Chen et al., Tuning garbage collection in an embedded Java environment, In Proceedings of International Symposium on High-Performance Computer Architecture, pp. 92-103, Feb. 2002.
[14] CDC reference implementation 1.1 corresponding to JSR 218, http://java.sun.com/products/cdc/index.jsp
[15] D. Stefanović, M. Hertz, S. M. Blackburn, K. S. McKinley, J. Eliot B. Moss , Older-first garbage collection in practice: evaluation in a Java Virtual Machine, ACM SIGPLAN Notices , MSP '02, Vol. 38 No. 2, pp. 25-36, June. 2002.
[16] D. Stefanović, K. S. McKinley, J. Eliot B. Moss, Age-based garbage collection, ACM SIGPLAN Notices, OOPSLA '99, Vol. 34 No. 10, pp. 370-381, October 1999.
[17] L. T. Hansen, W. D. Clinger, An experimental study of renewal-older-first garbage collection, ACM SIGPLAN Notices , ICFP '02, Vol. 37 No. 9, pp. 247-258, September 2002.

저 자 소 개



이 상 윤(정회원)
1994년 한양대학교 전자통신 공학과 학사 졸업.
1996년 한양대학교 전자통신 공학과 석사 졸업.
1999년~현재 한국전자통신연구원 임베디드S/W연구단
<주관심분야 : 모바일 플랫폼, 자바 가상 머신, 임베디드 시스템>

최 병 욱(정회원)
1973년 한양대학교 전자공학과 학사 졸업
1978년 일본 경응의숙(KEIO) 대학 전기공학과 석사 졸업
1981년 일본 경응의숙(KEIO) 대학 전기공학과 박사 졸업
현 한양대학교 정보통신대학 정보통신학부 교수
<주관심분야 : 영상처리, 멀티미디어 공학>

김 상 욱(정회원)
1989년 서울대학교 컴퓨터공학과 학사 졸업.
1991년 한국과학기술원 전산학과 석사 졸업.
1994년 한국과학기술원 전산학과 박사 졸업.
현 한양대학교 정보통신대학 정보통신학부 교수
<주관심분야 : 데이터베이스 시스템, 저장 시스템, 트랜잭션 관리, 데이터 마이닝, 멀티미디어 정보 검색, 이동 객체 데이터베이스/텔레매틱스>