임베디드 자바가상머신을 위한 가비지 콜렉션 설계 및 구현

백대현[0] 박희상 양희권 이철훈
충남대학교 컴퓨터공학과
(dhbaek[0], hspark, hkyang, chlee)@ce.cnu.ac.kr

Design and Implementation of Garbage Collection Based On Embedded Java Virtual Machine

Dae-Hyun Baek[0], Hee-Sang Park, Hee-Kwon Yang, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

자바의 가장 중요한 특성 중 하나는 플랫폼 독립성이다. 즉, 자바가상머신(Java Virtual Machine: JVM)이 탑재된 모든 플랫폼에서 운영체제의 종류와 상관없이 Java로 작성된 프로그램을 수행시킬 수 있다는 것이다. 이를 위해서는 각각의 플랫폼에 맞는 JVM이 적재되어야 한다. 본 논문에서 구현하게 될 가비지 콜렉션은 JVM의 성능을 좌우하는 중요한 요소이다. 가비지 콜렉션을 구현할 때 이용되는 알고리즘에는 여러 가지가 있다. 본 논문은 stop-copy와 마크-회수 알고리즘에 대해서 설명하고, 마크-회수 알고리즘을 개선한 마크-회수 압축 알고리즘을 이용한 가비지 콜렉션의 설계 및 구현한 내용을 기술하고 있다.

1. 서 론

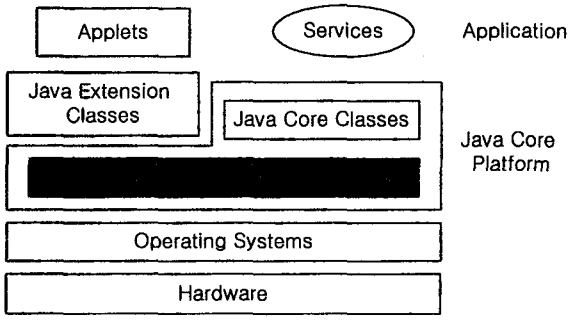
증가하는 임베디드 시스템에 대한 운영체제와 각각에 맞는 응용 프로그램들을 매번 개발해야 한다면, 개발자에게는 매우 번거로운 일이다. 그러나 각각의 임베디드 시스템에 맞는 자바가상머신(Java Virtual Machine: JVM)이 탑재되어 있다면 하나의 응용 프로그램을 작성하더라도 모든 플랫폼에서 사용할 수 있게 된다[1].

이러한 JVM에서 메모리 관리는 매우 중요하다. JVM에서 메모리는 가비지 콜렉터에 의해 관리된다. 가비지 콜렉터는 프로그램에 의해 더 이상 참조되지 않는 객체들을 찾아서 그 객체들이 차지하고 있던 힙(heap) 공간을 재사용할 수 있게 한다. 이러한 가비지 콜렉터는 생산성을 높여주고, 프로그램 무결성을 보장하며, 자바 security 전략에 중요한 역할을 한다. 그러나 단점으로 프로그램 성능에 영향을 미치는 오버헤드가 발생하는데 이를 줄이는 것이 본 논문에서 해결할 목표 중 하나이다[2][3].

가비지 콜렉터는 마크-회수 알고리즘을 이용하여 구현하게 되면 힙 단편화 문제가 발생하게 된다. 이를 해결하기 위해 본 논문에서는 가비지 콜렉터를 좀더 진보된 마크-회수 압축 알고리즘을 이용하여 구현한다. 본 논문은 2장에서 관련연구를 3장에서는 가비지 콜렉션 알고리즘을, 4장에서는 실험환경 및 결과를 5장에서는 결론 및 향후 과제에 대하여 기술한다.

2. 관련 연구

2.1. 자바 플랫폼



[그림 1] 자바 플랫폼

자바 코어 플랫폼은 자바 프로그램의 핵심적인 클래스들인 자바 코어 클래스들과 JVM, 그리고 다양한 운영체제 및 브라우저 중간에서 연결 역할을 하는 하드웨어 종속적인 부분과 자바의 활용을 확장시키는 역할을 하는 자바 표준 확장 클래스들로 구성된다[4].

JVM은 일종의 소프트웨어 CPU로 자바 번역기라고도 불리며, 자바 플랫폼에서 가장 핵심적인 부분으로 다양한 운영체제에서 애플리케이션이 실행될 수 있게 해주는 abstract computing machine이다. 이는 real computing machine 처럼 instruction set을 가지고 있으며 실행시에 필요한 여러 종류의 메모리 영역을 조작한다[5].

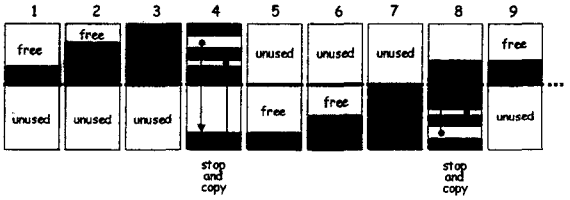
2.2. 가비지 콜렉션 알고리즘

모든 가비지 콜렉션 알고리즘의 기본적인 기능으로 가비지 객체를 찾아서 그 객체에 의해 사용된 힙 영역을 다른 객체가 재사용할 수 있게 해야 한다[2].

2.2.1. Stop - Copy 알고리즘

이 알고리즘의 특징은 힙 영역을 두 부분으로 나누어 한쪽 영역만 사용하는 것이다. 한쪽 메모리 공간이 모두 소비되면 프로그램의 실행이 잠시 멈추고 가비지 콜렉터는 힙을 순회한다. 힙을 순회하면서 현재 참조중인 객체는 다른 한쪽 영역으로 복사된다. 이 과정이 끝나면 잠시 정지된 프로그램을 다시 수행한다[2][5].

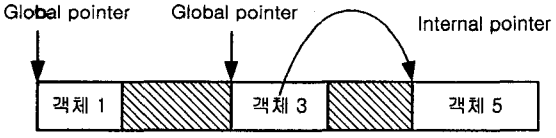
[그림2]의 1에서 밑의 반은 아직 사용되지 않는 힙 영역을 나타내고, 위쪽의 색칠한 영역은 객체가 할당된 부분을 나타낸다. 3에서와 같이 한쪽 영역이 모두 소비되면 가비지 콜렉터가 실행되어 현재 객체에 의해 참조 중인 영역을 아래쪽 영역으로 복사한다. 다시 아래쪽 영역에서 객체에 필요한 공간을 할당한다.



[그림 2] 가비지 콜렉티드 힙

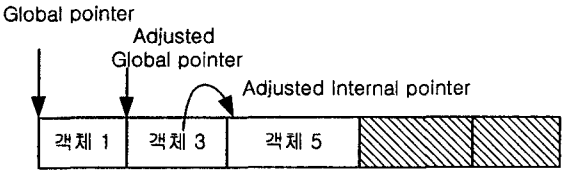
2.2.2. Mark-Sweep(with Compact) 알고리즘

마크-회수 알고리즘은 두 단계의 과정을 거쳐 가비지 콜렉션을 수행한다. 첫 번째 단계로 마크 단계에서는 root로부터 시작하여 참조 트리를 순회하며 살아있는 객체에 대해 마크한다. 다음 단계인 회수 단계에서는 마크되지 않은 객체를 반환하여 그 객체가 차지하고 있던 힙 영역을 프로그램이 다시 사용할 수 있도록 한다[2][3].



[그림 3] mark-sweep 가비지 콜렉터

이러한 알고리즘은 포인터 조작이 필요없고 작은 공간을 요구하지만 메모리 단편화를 발생시키는 커다란 문제가 있다. 이런 단편화를 해결하기 위한 대안으로 압축 알고리즘을 적용한다. 압축 알고리즘은 사용하지 않는 힙 영역을 한쪽 끝으로 압축하여 힙 영역을 연속적으로 만든다. [그림4]는 [그림3]에 압축 과정을 적용했을 때 나타나는 결과이다.



[그림 4] mark-compact 가비지 콜렉터

3. JVM을 위한 가비지 콜렉션 구현

마크-회수 압축 방법을 이용한 가비지 콜렉션의 전체적인 구조는 [표1]과 같다. 객체에 필요한 메모리 공간을 더 이상 할당 받지 못할 경우 mark_sweep() 함수를 호출하여 가비지 콜렉션을 수행한다.

[표1] 마크-회수 압축 과정

```
mark_sweep() {
    for R in Roots
        mark(R);
    sweep();
        if (free_pool is empty)
            compact();
        if (free_pool is empty)
            abort " Memory exhausted"
}
```

3.1. 마크 과정

본 논문에서는 bitmap marking 기법을 사용한다. 한 비트는 힙에서 객체의 시작 주소와 맵핑된다. 만약 p가 객체의 시작 주소라면 아래와 같이 표현된다.

mark_bit(p) = return bitmap[p >>3];

[표2] 마크 알고리즘의 과정은 루트로부터 검사를 시작해서 마크되지 않은 객체에 대해 그 객체가 참조 중인지를 판단하여 참조중인 객체에 대한 bitmap을 'marked' 하고, 그 객체에 대한 자식 객체가 있는지 검사하여 자식 객체가 존재하면 위 과정을 반복적으로 수행한다.

[표2] 마크 알고리즘

```
mark(N) {
    if (mark_bit(N) == unmarked) {
        mark_bit(N) = marked;
        for M in markChildren(N)
            mark(*M)
} }
```

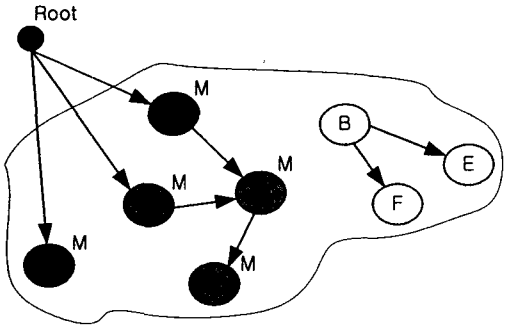
3.2. 회수 과정

[표3] 회수 알고리즘의 과정은 힙의 아래부터 검사하여 bitmap에 해당하는 객체가 'unmarked' 되었다면 그 공간을 다른 프로그램에 의해 사용될 수 있도록 해제하고, 'marked'가 되어있다면 다음 가비지 콜렉션이 일어날 때를 대비하여 'unmarked' 표시를 하고 다음 객체를 검사한다.

[표3] 회수 알고리즘

```
sweep() {
    N = heap_bottom;
    while (N < heap_top) {
        if (mark_bit(N) == unmarked)
            free(N);
        else {
            mark_bit(N) = unmarked
            N = N + size();
} }
```

[그림 5]는 마크-회수 단계를 마친 후의 상태로 B,E,F 객체는 더 이상 참조되는 객체가 아니므로 가비지 콜렉션 된 경우를 나타낸다.



[그림 5] 가비지 콜렉션 후의 상태

3.3. 압축 과정

[표4] 압축 알고리즘의 과정은 먼저 마크 과정을 수행한다. 이는 회수 과정을 수행하고 나면 bitmap에 대한 값이 ' unmarked' 로 세팅되기 때문이다. 다음 과정으로 힙의 위쪽부터 아래쪽으로 검사하여 객체가 참조되고 있는지 체크한다. 객체가 더 이상 참조 되고 있지 않으면 다음 과정을 수행한다. 객체가 참조되고 있으면 객체에 대한 bitmap을 ' unmarked' 로 세팅하고, free가 가르키는 위치에 현재의 객체에 대한 정보를 이동한다. 다음 과정은 현재 객체가 차지하고 있던 힙 영역을 해제하고 다음 객체에 대한 검사를 수행한다. 압축과정이 모두 끝나면 힙의 아래쪽에 사용 가능한 영역이 존재하게 된다.

[표4] 압축 알고리즘

```
compact () {
    free=heap_bottom;
    live=heap_top;
    for R in Roots
        mark(R);
    while (live > heap_bottom) {
        if (mark_bit(live) == unmarked)
            live = live- size();
        else {
            mark_bit(live) = unmarked;
            move(live, free);
            free(live);
            live = live -  size();
            free = free -  size();
} } }
```

4. 실험환경 및 결과

실험환경은 Solaris sparc 운영체제에 JVM과 클래스 라이브러리는 PJAE 3.1을 사용하였고, 컴파일러는 Java Development Kit 1.2.2의 컴파일러와, gcc와 cc를 사용하였다. 본 연구에서 결과를 검증하기 위해 무한 루프를 수행하면서 객체를 생성하는 프로그램을 실행해본 결과 가비지 콜렉션이 일어남을 알 수 있었다.

5. 결론 및 향후 과제

JVM의 가비지 콜렉션 알고리즘으로 사용되는 마크-회수 방식은 가지비 콜렉션이 진행됨에 따라 단편화 문제가 발생하게 된다. 이를 해결하기 위해 마크-회수 방식에 압축 기법을 추가하였다. 그러나 이 압축과정은 객체에 대한 모든 정보를 재 위치시켜야 하기 때문에 프로그램이 잠시 정지 상태로 있어야 하는 불가피한 경우가 발생하게 된다.

대부분의 프로그램에 의해 생성된 대부분의 객체는 매우 짧은 시간동안 사용된다. 대부분의 프로그램은 매우 오랜 시간동안 살아있는 몇 개의 객체만 생성한다[2]. 위와 같은 특성을 이용하여 마크-회수 압축 알고리즘을 보완하기 위해서 generation 알고리즘을 이용한다. 마크-회수 방식과 generation 방식의 결합, 또는 stop-copy 방식과 generation 방식의 결합과의 비교가 필요하고 이에 대한 보완을 해 나가야 할 것이다.

참고문헌

[1] http://www.inestech.com
[2] Bill Venners, "Inside the JAVA2 Virtual Machine second edition", p356-384, 1999
[3] Richards Jones, "Garbage Collection Algorithm for Automatic Dynamic Memory Management", p75-182, 1999
[4]http://java.sun.com/j2se
Java2 Platform Standard Edition
[5] Sun Microsystems, "The Java™ Virtual Machine Specification second edition"