Object C의 가비지 컬렉션을 통한 메모리 관리 정책 분석 및 전망

권예진*, 박용범**
*단국대학교 전자계산학과
**단국대학교 컴퓨터공학과
e-mail:{kwon6838*, ybpark**}@dankook.ac.kr

Analysis and Forecast for Object-C garbage collection memory management policies.

Yejin Kwon*, Youngbom Park**
*Dept of Computer Science, Dankook University
**Dept of Computer Science, Dankook University

요 약

가비지 컬렉션(Garbage Collection)은 시스템에서 더 이상 사용하지 않는 동적 할당된 메모리 블록 혹은 개체를 찾아 자동적으로 다시 사용 가능한 자원으로 회수하는 것을 의미한다. 최근 대부분의 프로그래밍 언어에서는 메모리 관리를 자동으로 처리해주는 가비지 컬렉터를 기본적으로 포함하고 있으며 이러한 시스템 환경은 개발자들의 개발 속도 향상과 프로그램 가독성을 높여주는 이점을 주고 있다. 그러나 가비지 컬렉터는 자원이 한정되어 있는 스마트폰과 같은 환경에서는 큰 오버헤드를 가지며 성능 저하의 주 원인으로 꼽히기도 한다. 따라서 iOS의 경우에는 가비지 컬렉터를 지원하지 않는다.
이에 따라 본 연구에서는 스마트폰의 안드로이드와 iOS의 프로그래밍 언어인 Java와 Object C의 가비지 컬렉터의 알고리즘을 분석하여 두 언어의 개발환경의 차이를 비교 하였다. 또한 앞으로 Object C의 메모리 관리 정책에 대하여 서술하였다.

1. 서론

가비지 컬렉션(Garbage Collection)이란 시스템에서 더 이상 사용하지 않는 동적 할당된 메모리 블록 혹은 개체를 찾아 자동적으로 다시 사용 가능한 자원으로 회수하는 것을 말한다. 시스템에서 가비지 컬렉션을 수행하는 부분을 가비지 컬렉터(Garbage Collector)라고 하며, 최초의 가비지 컬렉터는 1958년 존 매카시(John McCarthy)에 의해 Lisp 언어의 일부로 구현되었다[1]. 가비지 컬렉션은 더 이상 프로그램에서 사용하지 않는 불필요한 개체 영역을 찾아내고 해당 개체가 사용하는 리소스를 회수하는 단계로 동작한다. 그러나 실제로는 어떠한 개체 영역이 더 이상 사용되지 않을 영역인지를 알아내기는 일반적으로 불가능하다. 따라서 가비지 컬렉터는 프로그램에서 사용하지 않을 개체를 찾아내기 위해 해당 개체를 참조할 수 있는 방법이 있는가를 체크해 접근 방법이 존재하지 않는 개체를 유효하지 않다고 판단해 리소스를 회수한다.

가비지 컬렉터는 가비지 컬렉션이 메모리 릭(memory leak)이나 이중해제(double free), 너무 빠른 해제(premature free)와 같이 수정하기는 까다롭지만 저지르기 쉬운 실수에 대해 자동으로 처리해 주므로 이러한 문제가 일어났을 때 추적하는 노력을 제거해 줄 뿐만 아니라 프로그램의 복잡도를 낮춰주므로 전체적인 생산성에도 긍정적인 영향을 준다. 그러나 대부분의 가비지 컬렉션은 다른 형태의 메모리 관리보다 속도가 느리며, 가비지 컬렉션 에러로 인한 버그는 디버그가 어렵다. 또한 사용되지 않는 포인터를 null로 설정하지 않는다면 메모리 누출이 일어날 가능성이 크다[2].

이러한 단점에도 불구하고 가비지 컬렉션은 현재 많은 프로그래밍 언어에 의해서 지원되고 있다. 대표적인 언어로는 Java, Smalltalk, C#, VB 등의 객체지향 언어들이 있고, 최근 Object-C 2.0 버전에서는 가비지 컬렉션 기능을 추가하였다. 여러 프로그래밍 언어 중에서도 특히 Java는 객체지향성, 안전성, 유연성 등과 같은 장점으로 현재 널리 사용되며, JVM(Java Virtual Machine, 자바 가상 머신)을 이용한 메모리 관리 시스템은 Java의 가장 큰 특징 중 하나이다. 또한 애플의 Object-C 2.0 버전부터는 가비지 컬렉션 기능을 추가해 기존의 retain/release기반 메모리 관리를 가비지 컬렉션을 이용한 메모리 자동 관리 시스템으로 변경하여 제공하였다.
이에 따라 본 연구에서는 Java의 가비지 컬렉션의 알고리즘과 Apple의 Object-C의 가비지 컬렉션의 정책과 알고리즘을 비교•분석하였다. 그 결과로 두 프로그래밍 언어의 가비지 컬렉션을 이용한 메모리 관리의 차이점을 비교하며 또한 앞으로 Object-C의 메모리 관리 정책의 전망에 대해서 간략하게 제시할 것이다.

2. 관련 연구

2.1. Java 가비지 컬렉션

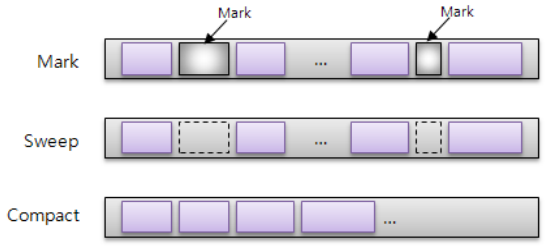
자바 언어로 작성된 소스코드는 컴파일 되면 자바 바이트코드로 변환된다. 이러한 자바 바이트코드는 JVM에 의해 해석되고 실행된다. 즉, JVM이란 컴파일된 자바 바이트코드와 실제로 프로그램의 명령어를 실행하는 마이크로프로세서 또는 하드웨어 플랫폼 간에 인터페이스 역할을 담당하는 소프트웨어를 의미한다[3].

JVM이 가지는 중요한 모듈 중 하나가 바로 가비지 컬렉터이다. JVM에서는 자바 프로그램을 실행하기 위한 여러 가지 실행시간 데이터 영역을 가지는데, 그 중 힙 영역은 다양한 객체들을 할당하기 위해 사용된다. 힙 영역에서 더 이상 사용되지 않는 객체가 차지한 공간은 가비지 컬렉터에 의해 재생된다[3]. Java의 JVM에서는 객체의 나이에 따라 영역을 분리하여 관리하는 세대별(Generation)방식을 사용한다. 세대별 방식의 Young영역은 복사(Copying) 방식으로 관리되는 Eden과 두 개의 Survivor 영역으로 구성되고, 일정한 나이에 도달한 객체는 Old 영역으로 복사된다. 또한 지속적으로 메모리에 잔류해야 하는 메소드 정의 코드나 static 변수와 같이 변하지 않는 값이 저장되는 Perm 영역이 있다. JVM에서 메모리 영역을 분류하는 방식을 나타내면 다음 <표 1>과 같다.

<표 1>JVM의 메모리 영역

메모리 영역	Young	Eden	생성되고 얼마 되지 않은 객체
		Suvivor1	
		Suvivor2	
	Old		생성되고 일정 시간이 지난 객체
	Perm		static 변수와 같은 변하지 않는 값이 저장되는 영역

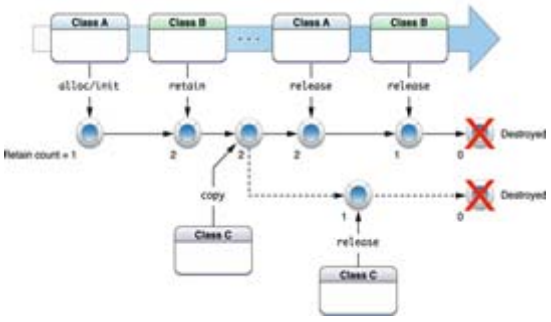
위의 메모리 영역 중 Old 영역은 객체가 생성된 후 일정 시간이 지난 객체로 사용 확률이 낮은 객체들이 저장되어 있다. Old 영역의 객체들은 garbage로 인식되어 메모리 사용 영역을 반환한다. 이 메모리 영역을 반환할 때 사용되는 알고리즘 방식에 따라 다른 가비지 컬렉션이 일어나게 된다[4, 5, 6]. JVM에서 주로 사용하는 가비지 컬렉터 알고리즘은Mark-Sweep-Compact 이다. 이 알고리즘은 힙 영역의 핸들 메모리 공간이나 객체 메모리 공간이 부족하여 메모리를 할당할 수 없을 경우 루트 셋을 스캔하여 필요하지 않은 메모리 영역에 마크하고 해당 메모리 영역을 sweep 작업을 통해 여유 메모리 영역을 확보한다. 또한 compact 작업은 sweep 작업을 통해 충분한 여유 메모리 영역을 확보하지 못한 경우 수행한다. 이 작업은 여러 개의 빈 공간들을 합쳐서 가능한 큰 덩어리의 블록으로 만드는 작업이다[7]. 다음 (그림 1)은 알고리즘의 수행 과정을 표현한 그림이다.



(그림 1) Mark-Sweep-Compact 가비지 컬렉터 알고리즘

2.2. Object-C 2.0 가비지 컬렉션

Object-C 2.0 이전의 버전에서는 retain/release 기반 메모리 관리 방식을 이용하여 프로그래머가 직접 메모리를 관리할 수 있는 시스템을 제공한다. 이 방법은 리테인 카운트(retain count) 기법을 사용하여 해당 객체를 참조하고 있는 수를 체크하여 그 수치를 가지고 있는 개념으로 만약 리테인 카운트가 0이 되면 해당 객체를 지우고 메모리를 반환하는 개념이다. 개념적으로는 가비지컬렉터와 유사하나 가비지컬렉터가 시스템에서 지속적으로 메모리를 체크해 여유 공간을 확보하는 방식이라면 리테인 카운트 기법은 로직으로 제거되는 방식이다. 이 방식은 프로그래머가 메모리를 수동으로 할당하고 해제하는 작업을 직접 수행해야 하는 번거로움이 있으나 메모리 관리를 효과적으로 수행한다면 성능이 뛰어난 응용프로그램을 만들 수 있다. Object-C의 retain/release 기반 메모리 관리 방식을 그림으로 나타내면 다음 (그림 2)와 같다[8].



(그림 2) retain/release 기반 메모리 관리 방식(Apple Developer)

Object-C의 retain/release 기반 메모리 관리 방식은 성능이 좋은 응용프로그램을 만들 수는 있지만 메모리 관리를 일일이 체크해야 하기 때문에 프로그래밍이 까다로울 수 있다. 또한 대부분의 객체지향 프로그래밍 언어에서는 가비지 컬렉터를 지원해 메모리 관리를 보다 편리하게 지원해 주고 있기 때문에 이에 맞춰 애플에서도 Object C 2.0에서부터는 가비지 컬렉터를 지원한다. 가비지 컬렉터를 사용하면 객체의 retain과 relase, auto relase poll, retain count에 대해 걱정할 필요가 없다. 시스템이 자동으로 어떤 객체가 어떤 객체를 소유하는지에 대한 정보를

끊임없이 알아내고, 프로그램에서 실행되는 동안 메모리 공간이 필요하면 더는 참조되지 않는 객체들을 자동으로 메모리에서 해제한다[9].

Object C 2.0의 가비지 컬렉터는 메모리를 세대(generations)별로 나누어 할당하고 가장 최근 세대를 가비지 컬렉션의 대상 우선순위로 한다. 객체를 세대별로 관리하기 위해Object C 2.0의 가비지 컬렉터는 write-barrier 기법을 사용한다. write-barrier는 각 객체의 포인터가 메모리에 할당된 다른 객체를 가리키는 객체 포인터인지 가비지로 수집되는 블록을 가리키고 있는 포인터인지를 감지한다. write-barrier는 객체가 생성되고 메모리에서 일정 시간이 지나면 “clump”이라고 객체에 마크(mark)한다. 보통 생성한지 오래 되어 “clump”으로 마크된 객체는 수가 적다. 가비지 컬렉션이 필요할 때, 스택 또는 “clump“으로 마크된 객체들과 최근 생성된 객체들의 연결을 재귀적으로 체크해 연결되어 있으면 “older”로 표시하고 “older”로 마크되지 않은 객체들은 더 이상 메모리에 잔류할 필요성이 없으므로 메모리를 반환하게 된다. Object C 2.0 의 가비지 컬렉터는 2에서 8의 세대로 나누어 이루어진다[11].

3. Java와 Object C의 가비지 컬렉터 비교

가비지 컬렉터를 이용하면 불필요한 객체를 메모리에서 자동으로 제거함으로써 프로그래머가 일일이 메모리를 체크하고 관리하는 수고를 덜 수 있다. 이는 가비지 컬렉션을 지원하는Java 프로그래밍의 개발 속도를 향상시키며, 프로그램 복잡도를 낮추고 가독성을 높인다. 이러한 자바의 가비지 컬렉터는 오랜 연구를 통해 속도나 성능을 지속적으로 향상시켜 왔다. 스마트폰의 안드로이드 앱은 기본적으로 Java를 기반으로 프로그래밍 되며, 이 또한 메모리 관리를 프로그래머가 직접 할 필요가 없다. 따라서 대부분의 안드로이드 앱은 일반 유저들도 메모리의 관리와 접근, 참조 등의 문제를 고민할 필요 없이 쉽게 개발할 수 있다.

Java와 Object C의 가장 큰 차이점은 스마트폰과 같이 한정된 자원과 실시간적인 특성을 지니는 어플리케이션을 개발할 때 가비지 컬렉션을 지원하는지에 대한 여부이다. 구글에서 제공하는 Java 기반의 안드로이드는 앱을 개발할 때 가비지 컬렉터를 지원한다. 그러나 Object C에서는 스마트폰 환경인 iOS에서는 가비지 컬렉터를 지원하지 않는다. 또한 Object C 개발자들은 가비지 컬렉터가 지원되는 환경임에도 불구하고 이전에 사용하는 retain/release 기반 메모리 관리 방식을 선호한다. 그 이유는 여러 가지가 있는데 가장 큰 특징은 스마트폰과 같이 한정된 자원 환경에서 가비지 컬렉터라는 시스템 자원 소모가 큰 모듈을 올리기엔 좋은 성능을 기대할 수 없다는 점이다. 개발자가 직접 메모리를 관리하고 relase를 시켜줘야 한다는 불편함 대신 빠른 속도와 성능을 선택한 것이다. 게다가 스마트폰과 같은 휴대용 기기에서는 사용자 인터페이스의 반응성이 대단히 중요하다. 직접적이고 생생한 사용자 인터페이스를 유지하려는 애플은 아이폰을 다른 스마트폰과 차별화시켜주는데 큰 역할을 하고 있다. 그리고 메모리의 한계 때문에 개발자들은 어쩔 수 없이 반응용 인터페이스에서 iOS 네이티브 API의 low level까지 관리할 수밖에 없다.

그러나 애플의 가장 큰 경쟁자인 안드로이드는 메모리가 자동으로 관리되는 언어와 API를 자사의 모바일 플랫폼에 제공하고 있다. 현대적인 개발기술에 있어서 애플을 앞서는 분야 중 하나가 이것이다. 그리고 구글은 최신 안드로이드에 Dalvik 가상머신(레지스터 기반의 가상 머신)을 활용해 성능을 향상시켰다. 메모리 자동화 관리 시스템을 위한 노력은 마이크로소프트 또한 .NET 플랫폼을 통해 프레임워크 환경을 구성함으로써 전달해왔다.

최근에 수행한 갤럭시 넥서스와 아이폰 4S, 갤럭시 S2의 벤치마크 성능 비교를 한 결과를 살펴보면 갤럭시 넥서스가 브라우저 속도와 자바스크립트 로딩 테스트에서는 아이폰4S 보다 빠르게 측정되었다. 그러나 OpenGL 테스트 결과를 보면 반대로 아이폰4S가 갤러시 넥서스보다 좋은 성능을 보였다. 이는 아이폰 환경에서는 사용자 인터페이스 반응성이나 게임과 관련된 앱들의 실행이 더 빠른 반면, 갤럭시 넥서스와 같이 안드로이드 환경에서는 브라우저 속도가 빠르다는 것을 의미한다[12]. 앞으로 스마트폰화 시장이 어떻게 변화될지는 확신할 수 없지만 스마트폰의 주요기능인 웹 브라우저 환경에서의 속도가 안드로이드 스마트폰이 앞섰다는 것은 주목할 만한 것이다.

4. Object C 메모리 관리 전망

완전 자동화 메모리 관리는 결국 주요 애플리케이션 개발환경의 기능이 될 것이다. 애플의 Object C 외에 대부분의 다른 언어들은 완전 자동화 메모리 관리를 해주는 API를 채택할 것이다. 이미 애플을 제외한 다른 경쟁사들은 메모리가 부족하고 CPU도 한계가 있는 모바일 영역에서 애플보다 한세대 앞선 언어와 API를 제공하고 있다. 이러한 시장 속에서 애플은 현재제공하고 있는 프로그래밍 언어인 Object C와 API가 새로운 다른 기술보다 중요한 장점을 가지고 있으며 자신들만의 경쟁성을 가지고 있으므로 현재 시스템을 유지해야 한다고 한다. 이는 과거 Java가 프로그래밍 언어로서 성공을 이루어 냈을 때, C++에 집착하던 마이크로소프트사가 대단히 빠르게 인재와 자금력을 동원하여 .NET 가상머신과 C#을 개발했을 때와

는 다른 방향성을 지닌다.

기존의 애플의 기술과 그 결과물들은 메모리 관리와 API에 대한 애플의 필요를 충족시켜주지 못할 것이다. 스마트폰과 같은 모바일 환경에서 Object C는 분명 다른 언어보다 더 빠른 속도와 좋은 성능을 내고 있다. 하지만 구글의 안드로이드 기반의 스마트폰이 계속적으로 연구되고 업데이트되는 과정을 살펴보면 그리 안심할만한 상황이 아니다.

개발 환경에서 좋은 프레임워크가 생산성에 주는 영향은 무시하지 못한다. Object C의 자동화 메모리 관리 문제는 주목해야 할 과제로 남아 있을 것이며 또한 개선 방안에 대한 연구가 필요할 것이다.

참고문헌

[1] Roger S. Pressman ″Software Engineering A Practltiners′ Approach″ 3rd Ed. McGraw Hill

[1] Wikipedia, “Garbage collection(computer science)”

[2] Jonathan Bartlett, “메모리 관리”, IBM developer Works, 2004.11.16.

[3] Tim Lindholm and Frank Yellin, “The JavaTM Virtual Machine Specification(second edition)”, Addison-Wesley. 1999.

[4] Paul. R. Wilson, “Uniprocessor Garbage Collection Techniques”, Technical report, Unicersity of Texas, Expanded version of the IWMM92 paper, Jan 1994.

[5] Sun microsystems, “The HotSpot Virtual Machine”, Technical White Paper, 2001

[6] Jacob Seligmann, Steffen Graup, “Incremental Mature Garbage Collection Using the Train Algorithm”, 1995

[7] 임동기, 배철성, 정민수, “자바가상기계 최적화를 위한 가비지 컬렉션 알고리즘과 힙 메모리 연구”, 한국멀티미디어학회 춘계학술발표논문집, 2002

[8] Mac OS X Developer Library, “Advanced Memory Management Programming Guide” 2011. 9. 28.

[9] Kochan, Stephen G, “Programming in Objective-C 2.0”, Pearson P T R, 2008. 12.11

[10] Mac 개발자 센터, “개발자를 위한 Leopard 기술”, Apple Developer Connection, 2007.11.6.

[11] “Garbage Collection Programming Guide”, Apple Inc, 2010.8.27.

[12] Anand Lal Shimpi, “Galaxy Nexus & Ice Cream Sandwich: Initial Performance Analysis”, 2011.11.18