실시간 운영체제 iRTOS™ 상의 KVM 메모리 관리 체계 내에서 효율적인 가비지 콜렉션의 설계 및 구현

최인범°, 유용선, 이철훈
충남대학교 컴퓨터공학과
(ibchoi°, ysyoo, chlee)@ce.cnu.ac.kr

The Design and Implementation of Garbage Collection in KVM Memory Management Facility on Real-Time Operating System, iRTOS™

In-Bum Choi, Yong-Sun Ryu, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

최근 IT 산업의 발전과 더불어, 리소스가 제한된 소형 기기들의 사용이 비약적으로 증가하고 있는 추세이다. 자바는 플랫폼 독립성(Platform Independency), 보안성(Security), 네트워크 이동성(Network Mobility) 등의 장점을 가지고 있어, 이러한 소형 기기들에 자바 환경을 적용하게 되면 여러 가지 이점을 가지게 된다. 임베디드 장치나 모바일 같은 제한된 리소스를 사용하는 기기들에는 SUN 사의 CLDC(Connected, Limited Device Configuration)에서 정의하고 있는 K 가상 머신(K Virtual Machine: KVM)을 탑재하여 사용하게 된다. 본 논문에서는 실시간 운영체제 iRTOS™ 와 KVM 을 탑재한 소형 기기에서 좀더 효율적으로 KVM 의 메모리를 관리하기 위한 Garbage Collection 기법을 설계하고 구현한 내용을 설명한다.

1. 서론

연성 실시간 시스템(Soft Real-time System)에서 사용하는 iRTOS™ 운영체제에 KVM 을 탑재한 소형 기기들은 여타의 실시간 시스템과 마찬가지로 응용 태스크의 빠른 응답시간이나 높은 수준의 예측가능성이 보장되어야 한다.

그러나 자바는 플랫폼 독립성, 보안성, 네트워크 이동성, 실행코드의 재사용성, 작은 실행 파일 크기 등의 장점에도 불구하고, 인터프리터 방식의 느린 수행 엔진, 이식성을 고려한 하드웨어 직접접근 제한, 시간 제약성의 고려가 없는 병행 수행 모델, 동적 메모리 관리 등의 실시간성을 위협하는 문제점 때문에 실시간 시스템에 적용하기가 쉽지 않다.

자바의 동적 적응성, 이식성, 개발의 용이성 등의 장점은 소형 기기에 적용 시 개발 비용 감소 등 여러 가지 이득이 예상되기 때문에, 앞에서 언급한 문제점들을 극복하고 소형 기기에 적용하기 위한 연구들이 다방면에서 진행되어 왔다. 본 논문은 위의 문제점들 중 동적 메모리 관리 측면에서 실시간 운영체제 iRTOS™ 상의 KVM 메모리를 효율적으로 관리하기 위한 Garbage Collection 기법에 대해 기술한다.

본 논문에서는 2 장에서 실시간 운영체제 iRTOS™ 의 특성과 KVM 메모리 관리 체계에 대한 관련 연구를, 3 장에서 효율적인 KVM 메모리 관리를 위한 Garbage Collection 구현을, 4 장에서 테스트 환경 및 결과를, 5 장에서 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

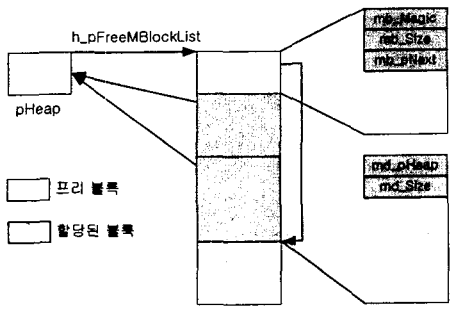
2.1 iRTOS™ 특징 및 메모리 관리

iRTOS™는 우선순위 기반의 멀티태스킹 환경 및 태스크간 통신을 위한 ITC(InterTask Communication) 환경을 지원하고, 효율적인 자원관리 및 태스크간 통신을 위해 세마포, 메시지 큐, 메시지 메일박스를 제공한다. 또한 iRTOS™는 동적 메모리 관리를 위해 가변크기의 메모리를 할당 또는 해제하는 힙 스토리지 매니저(Heap Storage Manager)와 고정크기의 메모리를 할당 또는 해제하는 메모리 풀, 두 단계의 메모리 관리 기법을 제공한다. 하드웨어 상의 메모리는 힙 스토리지 매니저에 의해 관리되며, 메모리 풀은 이미 생성된 힙 스토리지 매니저에서 시스템 초기화 시에 미리 할당 받아 여러 개의 고정크기의 버퍼로 관리한다.

2.2 힙 스토리지 매니저 (Heap Storage Manager)

[그림 1]은 힙 스토리지 매니저가 힙 메모리를 리스트로 관리하는 구조를 보여준다. 힙 스토리지 매니저는 할당되지 않은 프리 블록(Free Block)을 블록 앞에 12 바이트의 구조체를 선언하여 관리한다. 또한 할당된 메모리는 사용자가 원하는 크기에 추가적으로 8 바이트의 구조체를 할당된 블록 앞에 선언하고 관리한다.[1]

* 본 논문은 산업자원부 중기거점과제 연구비 지원에 의한 것임

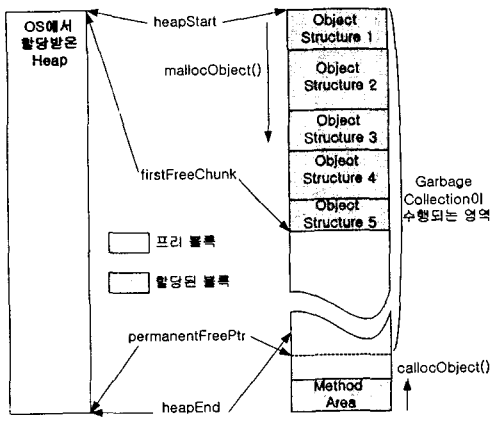


[그림 1] 힙 스토리지 매니저 구조

2.3 KVM 메모리 관리 체계

[그림 2]는 KVM 의 힙 메모리 관리를 보여준다. 실행중인 자바 어플리케이션은 객체나 배열을 생성할 때 필요한 메모리 공간을 할당 받기 위해 mallocObject() 함수를 이용하는데, 이를 이용하여 힙 메모리를 할당할 경우 firstFreeChunk 위치부터 시작하는 프리 리스트(Free List)에서 퍼스트 핏(First Fit) 알고리즘에 따라 힙 메모리를 할당하고, 할당되지 않은 공간은 다시 프리 리스트에 추가한다. 프리 공간이 남아 있지 않다면 CHUNK 구조체를 프리 리스트에서 삭제한다.

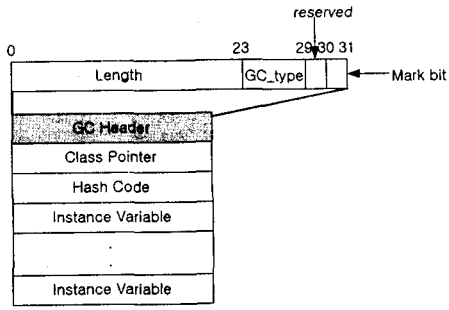
callocObject() 함수는 컨스턴트 풀 (Constant Pool)이나 필드 또는 메소드 정보 등 변하지 않는 클래스 정보를 저장하기 위한 공간 할당 시 사용한다. 이 함수를 이용하여 힙 메모리를 할당 할 경우, 요구하는 메모리 크기가 현재 사용 가능한 메모리 공간 (permane ntFreePtr - heapEnd)보다 작을 경우에는 메모리 공간을 할당하고, 그렇지 않을 경우에는 heapEnd 값을 재 설정한 후 메모리 공간을 할당한다.



[그림 2] KVM 메모리 관리

2.3.1 오브젝트 구조(Object Structure)

오브젝트 구조는 GC 헤더를 포함하여 클래스에 대한 포인터, 모니터와 해쉬코드에 대한 포인터, 클래스 객체에 필요한 데이터를 저장하기 위한 공간을 포함한다.



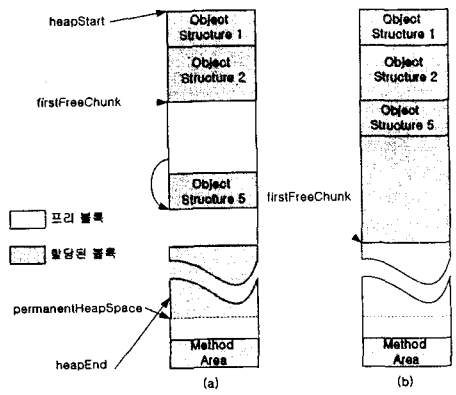
[그림 3] GC Header

[그림 3]에서 Length 는 GC 헤더 (1 byte)를 제외한 unsigned long(4 byte)을 하나의 크기로 가지는 길이를 나타내며, GC_type 은 가비지 콜렉션 타입을 나타낸다. 또한 Mark bit는 가비지 콜렉션 수행 시 오브젝트의 참조여부를 마크하기 위한 비트이다.

2.3.2 기본 가비지 콜렉션

마크-회수 압축 방법을 이용하여 가비지 콜렉션을 구현하였다. 더 이상 메모리를 할당하지 못할 경우 이 방법을 이용하여 가비지 콜렉션을 수행하며, 수행한 뒤에도 메모리를 할당하지 못한다면, 압축방법을 이용하여 가비지 콜렉션을 수행하여 힙 메모리를 할당한다.

[그림 4-a] 마크-회수 방법을 이용하여 가비지 콜렉션을 수행한 후의 상태이고, [그림 4-b]는 압축 방법을 이용하여 가비지 콜렉션을 수행한 후의 상태이다.



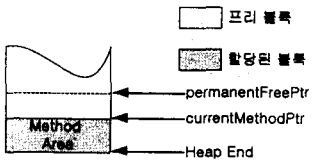
[그림 4] 가비지 콜렉션 후의 메모리 상태

3. KVM 메모리 관리를 위한 가비지 콜렉션구현

기본적으로 마크-회수 압축 방법을 이용하여 가비지 콜렉션이 수행되는데, KVM 에서 새로운 객체나 배열을 위한 메모리를 할당할 경우, 매번 가비지 콜렉션이 수행되어 *i*RTOS™의 특성인 시간 결정성을 위협할 수 있다. 그래서 본 논문에서는 마크-회수 압축 방법을 이용한 가비지 콜렉션이 수행되어지는 시점이 매번 일어나는 것이 아니라, 할당할 메모리의 영역이 부족할 경우를 예측하여 이 경우에만 가비지 콜렉션이 수행되게 하는 방법을 구현하였다.

KVM 은 새로운 객체나 배열을 위한 메모리를 할당

하기 위해 먼저 메소드 영역(Method Area)에 type 을 저장하게 된다. 저장된 이 데이터는 참조하는 힙의 데이터가 더 이상 필요없게 되더라도 항상 유지되는 정보이므로, 이것을 바탕으로 가비지 콜렉션이 수행될 시점을 선택하는 방법을 채택하였다. [그림 5]는 메소드 영역을 참조하기 위하여 새롭게 currentMethod Ptr 을 사용한 permanent 영역의 구조이다.



[그림 5] permanent 영역의 구조

3.1 가비지 콜렉션의 수행 시점 및 마크 과정

[그림 5]의 currentMethodPtr 와 permanentFree Ptr 를 이용하여 전체 permanent 영역에서 메소드 영역이 차지하는 비율을 계산, 일정 비율이 넘어가게 되면, 이 때 마크-회수 압축 방법을 이용한 가비지 콜렉션을 수행한다. 마크 알고리즘의 과정은 루트로부터 검사를 시작해서 마크되지 않은 객체에 대해 그 객체가 참조중인지를 판단하여 참조중인 객체에 대한 bit 를 ' marked' 하고, GC_type 에 따라 그 객체에 대해 참조중인 데이터를 모두 마크한다.

3.2 회수 과정

회수 알고리즘의 과정은 heapStart 로부터 검사를 시작하여 연속된 마크되지 않은 메모리 영역을 하나의 CHUNK 구조체를 생성하여 참조하고, 그 구조체를 프리 CHUNK 리스트에 추가하여 다른 객체에 의해 힙 메모리를 할당받을 수 있게 한다.

3.3 압축과정

마크-회수 과정을 수행한 후에도 할당해야 할 오브젝트나 배열을 위한 메모리 공간이 힙에 남아있는 메모리 공간보다 작아서 할당할 수 없는 경우 수행된다. 압축 과정은 현재 참조되고 있는 모든 객체에 대한 포인터와 데이터에 대한 포인터를 재배치 시키고, 프리 CHUNK 리스트도 초기화 하여 firstFreeChunk 포인터만 남겨지게 된다. 압축 과정이 끝나면 힙의 아래쪽 영역에 사용 가능한 메모리 공간이 존재하게 된다.

4. 테스트 환경 및 결과

본 논문의 테스트 환경은 S3C2400X 보드에서 실시간 운영체제는 *i*RTOS™를 사용하였고, 개발도구로 ARM SDT v2.51 을 사용하였다. 그리고 디버그를 위해서 ARM MultiICE Emulator를 사용하였다. 가비지 콜렉션의 테스트를 위하여 *i*RTOS™의 힙 스토리지 매니저를 통하여 할당받은 KVM Run-time Data Area의 permanent 영역을 32Kbyte로 가정하였고, 이것을 기준으로 currentMethodPtr과 permanentFreePtr을 사용하여 Method Area의 할당영역 대 permanent 전체영역의 비율을 20 퍼센트(20%)로 하였다. 이 비율은 여러번의 테스트 과정에서 20%, 30% 40% 등으로 변경하여 적용하였으며, 최적의 비율을 찾아내기 위하여 각 퍼센트당 테스트 회수를 3 회로 하였다.

[표 1] 테스트 결과

할당비율	애플리케이션 수행시간 (1 차)	애플리케이션 수행시간 (2 차)	애플리케이션 수행시간 (3 차)
적용안함	0.0074	0.0073	0.0069
20%	0.0062	0.0063	0.0068
30%	0.0055	0.0054	0.0054
40%	0.0056	0.0058	0.0054

※ time 함수 사용, 단위 : 초

[표 1]의 결과에서 먼저 할당 비율을 적용하지 않고 수행했을 경우와 적용하여 수행한 경우를 비교해 보면 할당 비율을 적용했을 때가 수행시간이 줄어드는 것을 볼 수 있다. 또한 할당 비율을 적용했을 경우를 살펴보면 이 비율에 따라 애플리케이션의 수행시간에 차이가 나타나는 것을 알 수 있다. 할당비율을 늘려 가비지 콜렉션 회수를 줄인다고 성능이 향상되지 않으므로 최적의 할당비율을 찾아 내는 것이 중요하다.

5. 결론 및 향후 연구과제

본 논문에서는 실시간 운영체제인 *i*RTOS™에서 힙 스토리지 매니저를 사용하여 KVM 에서 사용할 메모리를 할당 받고, 할당받은 메모리를 사용한 후 가비지 콜렉션이 일어날 경우 이를 얼마나 효율적으로 수행할 것인가에 대한 가비지 콜렉션 기법을 설계 및 구현하였다.

가비지 콜렉션의 수행 회수를 계속 줄이는 것이 성능의 향상을 가져오는 것이 아니기 때문에 최적의 수행 시점을 찾아내는 것이 매우 중요하다. 따라서 향후 연구과제로 좀 더 최적의 가비지 콜렉션 시점을 찾기 위한 다양한 연구가 계속 되어야 한다.

7. 참고문헌

[1] http://www.inestech.com
[2] Sun Microsystems, " *JSR-30 J2ME Connected, Limited Device Configuration*"
[3] Bill Venners, " *Inside the Java Virtual Machine*" ,1999
[4] Tim Lindholm, " *The Java™ Virtual Machine Specification Second Edition*" , 1999
[5] Sun Microsystems, " *KVM Porting Guide, CLDC, Version 1.0.4*"
[6] Sun Microsystems, " *Java™ 2 Platform Micro Edition Technology for Creating Mobile Devices*"