메모리가 적은 자바 시스템을 위한 자동 동적 메모리 관리 기법

최형규° 문수묵

서울대학교 전기컴퓨터공학부 마이크로프로세서 구조 및 시스템 소프트웨어 연구실

hectoct@altair.snu.ac.kr° smoon@altair.snu.ac.kr

Automatic Dynamic Memory Management Techniques for Memory Scarce Java system

Hyung-Kyu Choi° Soo-Mook Moon

MASS Lab., School of Electrical Engineering and Computer Science, Seoul National University

1. 서 론

많은 내장형 시스템들이 자바(Java)를 널리 채택하고 있다. 내장형 시스템은 자바 가상 머신(Java Virtual Machine, JVM)을 통해서 자바를 지원하며, 자바 가상 머신은 쓰레기 수집기 (Garbage Collector)를 통해서 동적 메모리를 자동으로 관리한다. 메모리가 적은 환경에서는 자바 가상 머신을 위한 메모리 공간과 자바 프로그램을 위한 메모리 공간을 공유함으로써 메모리를 효율적으로 사용한다. 또한, 여러 자바 프로그램을 하나의 자바 가상 머신에서 동작시킴으로 자바 가상 머신이 소모하는 자원을 최소화한다. 여러 자바 프로그램과 자바 가상 머신이 메모리를 공유하는 환경에서는 빈 메모리가 여러 개여 작은 조각으로 나뉘어 존재하는 외부 단편화 문제를 반드시 해결해야 하며, 자바 프로그램들이 종료된 후에 사용하지 않는 메모리를 찾아 수거해야 한다. 본 논문에서는 여러 자바 프로그램을 동시에 실행할 수 있는 자바 가상 머신에서 적은 메모리를 사용하면서 메모리를 효율적으로 관리할 수 있는 메모리 관리 기법을 제안한다. 우선 개선된 압축(compaction)기법 기반의 쓰레기 수집 기법을 소개하여 움직일 수 없는 메모리 영역이 존재할 경우에 발생하던 외부 단편화 (external fragmentation) 문제를 극복한다. 다음으로 수행 중 메모리 사용을 줄이기 위해서 쓰레기 수집기가 메모리에서 필요 없는 클래스(class)들을 선택적으로 수거하는 class unloading 기법을 소개한다. 마지막으로 소개한 기법들을 실제 환경에 구현하여 그 성능을 평가해 보았다.

2. 압축 기법과 쓰레기 수집기를 이용한 Class Unloading

외부 단편화 문제는 기존에 오랫동안 연구된 압축(compaction) 기법을 쓰레기 수집기에 적용함으로써 극복할 수 있다. 하지만 기존의 압축 기법들은 일반적으로 모든 메모리 공간이 움직일 수 있다고 가정하고 있어 그대로 적용하기 어려운 측면이 있다. 자바 프로그램에서 사용하는 Java object와 달리 자바 가상 머신에서 사용하는 메모리는 움직일 수 없는 C object인 경우가 많기 때문이다. 이를 위해서 기존에 추가적인 메모리를 요구하지 않고 복잡도 면에서도 선형적이라고 알려진 Table-based compaction 기법을 개선하여 움직일 수 없는 C object가 있는 경우에도 적용할 수 있도록 하였다. Table-based compaction은 사용 중인 메모리를 한 쪽으로 모으는 compaction 단계와 옮겨진 메모리 영역을 가리키는 참조들을 갱신하는 갱신 단계로 동작한다. 본 논문에서는 다음과 같은 방법으로 움직일 수 없는 메모리가 있는 환경에 이 압축 기법을 적용하였다. 우선 C object들을 경계로 구분되는 압축가능 영역을 찾는다. 그 후 Table-based compaction의 압축 단계를 각 압축가능 영역에 적용한다. 이 때 미리 정해져 있는 개수 M개의 압축 가능영역에만 압축 단계를 적용하며, 이는 압축가능 영역의 위치를 저장하는 메모리 공간이 추가적으로 필요하기 때문이다. 다음으로 압축된 M개의 영역에서 옮겨진 메모리 영역을 가리키는 참조들을 갱신한다. 그 후 다시 아직 압축되지 않은 압축가능 영역들에 압

이 논문은 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 일부 받아 수행된 연구임 (KRF-2006-311-D00757).

축과 갱신단계를 적용하기를 반복한다. 이와 같은 방법을 활용하면 Table-based compaction의 장점들을 유지하면서 움직일 수 없는 메모리가 존재하는 경우에도 압축을 수행할 수 있다.

다음으로 쓰레기 수집기를 활용하여 자바 프로그램이 종료된 후 사용되지 않는 class들을 쉽게 unloading하는 방법을 소개하기로 한다. 자바 가상 머신은 메모리로 읽어 들인 class들의 목록을 class table이라는 자료구조로 관리한다. 이 class table을 다음과 같은 두 종류로 나누었다. system class들을 관리하는 global class table 그리고 해당 자바 프로그램을 구성하는 class들을 관리하는 local class table로 구분하였다. 자바 프로그램이 시작될 때 해당 프로그램을 위한 local class table이 생성되며, 프로그램이 수행되는 동안에 읽어 들이는 class들은 local class table에 등록이 된다. 결과적으로 자바 프로그램이 종료된 후, 쓰레기 수집기는 해당 local class table에 속한 모든 class들은 더 이상 사용되지 않음을 쉽게 알 수 있어 메모리에서 해당 class들을 unload하게 된다.

3. 성능 평가

본 논문에서 소개한 두 가지 기법을 CDMA 핸드폰 단말기에서 동작하는 자바 가상 머신에 적용하여 그 성능을 평가해 보았다. 이 단말기는 ARM7 기반의 CPU를 탑재하고 있으며 자바 가상 머신을 위해서 400,000 bytes의 메모리가 할당되었다. 실험에서는 총 6가지 자바 프로그램을 사용하였으며 게임, 앨범, 주식 거래 등 다양한 종류로 구성되어 있다.

외부 단편화 문제를 극복하였는지를 확인하기 위해 6가지의 프로그램을 하나씩 실행 시키는 동안, 압축을 수행한 경우와 하지 않은 경우의 가장 큰 빈 영역의 크기와 빈 영역이 몇 개로 나뉘어 존재하는가를 측정해 보았다. 압축을 수행하였을 경우 수행하지 않았을 때와 비교하여 최대 10배 크기의 빈 영역을 얻을 수 있었으며 평균적으로 3.3배 큰 빈 영역을 얻을 수 있었다. 또한 빈 영역은 압축을 수행하였을 경우 2~4개로 나뉘어진 반면 압축을 하지 않은 경우에는 수백 개로 나뉘어지는 것도 확인하였다. 이를 통해서 소개된 압축 기법이 움직일 수 없는 메모리가 있더라도 외부 단편화 문제를 상당부분 극복하여 상대적으로 큰 빈 영역을 얻을 수 있다는 것을 알 수 있었다.

다음으로 class unloading의 성능을 평가해 보았다. 각 자바 프로그램을 실행하고 종료한 후 메모리 사용량을 측정하였다. 그 결과 class unloading을 수행하면 메모리 사용량이 평균적으로 약 20kbytes 줄어든 것을 관측할 수 있었으며 이는 현재 자바 가상 머신이 사용할 수 있는 메모리 400,000bytes에서 약 5%를 차지하는 크기이다. 6개 프로그램을 모두 수행하고 종료한 후에 메모리 사용량을 측정한 결과 class unloading을 통하여 100kbytes의 메모리 사용량이 줄어드는 것을 관측하였다. 이는 실험에서 사용된 환경에서는 약 25%의 메모리 사용량을 절감할 수 있다는 것을 뜻한다. 이처럼 class unloading만으로도 내장형 시스템과 같이 메모리가 적은 환경에서는 유의미한 크기의 메모리 사용량을 줄일 수 있음을 확인할 수 있었다.

마지막으로 6개의 자바 프로그램을 동시에 수행하는 실험을 수행하였다. 압축 기법과 class unloading을 동시에 사용하는 경우에 6개 프로그램들은 모두 동시에 수행될 수 있었다. 반면 압축 기법을 사용하지 않는 경우에는 외부 단편화 문제 때문에 6개의 프로그램들을 모두 동시에 수행하지 못하였다. Class unloading 기법의 경우는 프로그램들이 동시에 수행될 때에는 효과가 없었으며, 이는 class unloading이 프로그램이 종료되는 경우에만 수행되기 때문이다. 여러 프로그램들이 수행 및 종료되는 환경에서는 class unloading을 통해 더 많은 빈 메모리 공간을 확보할 수 있어 더 많은 프로그램을 수행할 수 있는 환경을 제공할 수 있다.

4. 결론

본 논문에서는 메모리가 적은 내장형 자바 시스템에서 효과적인 동적 자동 메모리 관리 기법들을 소개하였다. 움직일 수 없는 메모리 영역이 존재하는 경우에도 적용 가능한 압축 기법을 제안하여 외부 단편화 문제를 상당부분 해결할 수 있었으며 분리된 class table을 통하여 쓰레기 수집기가 종료된 프로그램에서 사용하던 class들을 쉽게 메모리에서 제거하는 class unloading 기법을 제안하였다. 그리고 이들 실제 환경에 구현하여 실험한 결과 상당한 메모리 절감 효과를 확인할 수 있었다. 내장형 자바 시스템에서는 여러 제약 조건 때문에 복잡한 쓰레기 수집 기법을 도입하기 쉽지 않지만 본 논문에서 보인 것처럼 간단하면서도 효과가 큰 기법들을 도입하면 적은 메모리를 효율적으로 사용할 수 있음을 알 수 있었다.