EVM을 위한 로더의 설계 및 구현

김성진[0] 고광만
상지대학교 컴퓨터정보공학부
yes756@kornet.net, kkman@mail.sangji.ac.kr

Design and Implementation of the Loader for the EVM

Kim Seong-Jin[0] Ko Kwang-Man
School of Computer and Information Engineering, SangJi University

요 약

가상기계는 소스 프로그램에 대한 실행 파일 형태를 다양한 종류의 플랫폼에 대한 독립성을 지원하는 프로그램 실행 환경으로서 로더/링커, 인터프리터 및 가상 기계를 특정 시스템에 탑재하기 위한 어댑터로 크게 구성되어 있다. 본 연구팀의 최종 목표는 기존의 가상기계를 기반으로 임베디드 시스템에 적합한 가상기계(EVM)를 개발하고자 한다. 이를 위해 다양한 연구 시도가 진행되고 있으며 본 논문에서는 EVM 개발시에 설계된 실행 파일 형식(*.evm)에 대한 로더를 설계하고 구현하였다. 또한 인터프리터의 실행 효율을 위하여 로더의 출력이 실제 메모리에 저장되는 구조를 개선하였다.

1. 서 론

가상기계는 컴파일된 특정 프로그램의 중간코드와 실제 프로그램의 명령어를 실행하는 플랫폼 간에 인터페이스를 담당하는 소프트웨어를 말한다. 대표적인 예로서 JVM을 보면, 작성된 자바프로그램을 컴파일하여 실행 파일 형식인 클래스 파일을 생성하여 다양한 플랫폼에서도 실행 가능하게 해 준다. 이런 가상기계를 통해서 특정 응용프로그램에 대해서 독립적으로 실행 될 수 있는 환경을 만들 수 있는 것이다. 또한 최근에 와서는 컴퓨터 시스템이 아닌 핸드폰, PDA, 디지털TV, 셋톱박스와 같은 소규모 장치에서도 특정 응용프로그램으로 작성된 프로그램이 수행되게 하는 환경이 개발 되고 있다.

EVM(Embedded Virtual Machine)은 구현 중인 가상기계로서 링커, 로더 및 인터프리터를 포함하고 있는 형태로 설계되었으며, 현재 로더를 구현하였다.

본 논문에서는 구현된 로더의 입력파일은 검증을 위하여 자바의 중간코드인 클래스파일의 형식을 재설계하였다.(*.evm). 또한 이렇게 만들어진 중간코드인 *.evm을 이용하여 로더가 필요한 정보를 추출하고, 추출되어진 정보를 효율적으로 저장하기 위한 자료구조에 대하여 설계 및 구현하였다.

본 논문의 구성은 제 2장에서 가상기계의 전반적인 설명과 가상기계 내에서의 로더의 기능을 설명하고, 제 3장에서는 본 논문에서 설계하고, 구현 된 로더에 대해 살펴본다. 마지막 제 4장에서는 결론 및 향후 연구 내용에 대해서 설명하였다.

이 논문은 한국과학재단의 특정기초연구(과제번호: R01-2002-000-00041-0)지원에 의한 것임.

2. 기반 연구

2.1 가상기계

가상기계는 특정 응용프로그램에 대해서 독립적으로 실행 될 수 있는 환경을 만드는데 사용된다.

현재 사용되는 가상기계는 대표적 가상기계인 JVM을 비롯하여, KVM, Waba-VM, GVM, BREW등 사용되는 종류가 많다. 이들 가상기계들의 큰 차이점은 사용되어지는 용도가 다르며, 또한 서로 다른 API를 사용한다는 것이다.

JVM(Java Virtual Machine)은 OS, 하드웨어, 웹브라우져등에서 자바를 구동시키기 위해서 설치되어지며, 자바 내 정의되어진 API를 사용한다[1].

KVM(KiloByte Virtual Machine)은 핸드폰, PDA, 임베디드 장비등에서 자바로 구동시킬 때 설치되어지며, q 별도의 API를 사용한다[4].

Waba VM(Waba Virtual Machine)은 Palm, Windows CE와 같은 소형 장치들을 위해 개발되었다. 자바의 문법을 그대로 따르고 있으나, 별도의 API를 사용한다[3].

2.2 JVM에서 로더의 기능

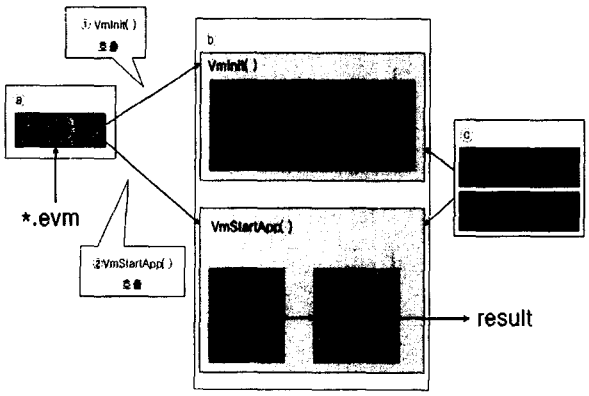
자바 가상기계의 클래스 로더는 시스템 클래스 로더(Bootstrap Class Loader)와 사용자-정의 클래스 로더(User-defined Class Loader)로 구분되며 입력된 클래스 파일에 대한 바이너리 데이터를 자바 가상기계에 로딩, 링킹, 초기화 동작을 순서대로 수행한다. 로딩 단계에서는 클래스 또는 인터페이스와 같은 자료형에 대한 바이너리 데이터를 찾아 적재하는 동작을 수행한다. 링킹 단계에서는 첫째, 적재된 자료형에 대한 정확성을 검증(verification)하며 둘째, 클래스 변수에 대한 메모리 할당

및 메모리를 기본 값으로 초기화하는 예비(preparation) 동작을 수행한다. 마지막으로 자료형에 심볼릭 참조를 상수 풀 정보를 이용하여 직접 참조로 변환하는 결정(resolution) 동작으로 구성되어 있다. 초기화 동작 단계에서는 클래스 변수를 초기화할 수 있는 메소드를 호출하여 적당한 시작 값으로 초기화한다[1].

3. 로더의 구현

3.1 개요

구현 된 로더는 [그림 1]처럼 크게 세부분으로 나누어 다. main함수를 포함하는 ⓐ와 저장을 위해 구현된 스택 및 힙을 초기화하기 위한 함수(VmInit( )) 및 로더 호출 함수(VmStartApp( ))를 포함하는 ⓑ, 마지막으로 기본 자료형 및 라이브러리 함수가 포함되어 있는 ⓒ로 구성되어 있다.



[그림 1] 로더의 세부 구성도

각각의 부분에 대한 기능을 자세히 보면, ⓐ에서는 main함수가 구현되어 있으며, 전체적인 가상기계의 초기화 함수 호출과 실행 함수 호출을 하게 된다. ⓑ에서는 로더가 추출한 정보를 저장하는 장소인 스택과 힙의 초기화 및 동적할당을 하게 되며, 로더가 구현되어 있다. ⓒ에서는 Windows와 Unix환경을 위한 헤더가 정의되어 있으며, 입력되어지는 중간코드 *.evm에서 원하는 정보를 바이트 단위로 얻기 위해 getXXX로 시작되는 매크로 함수가 정의되어 있다.

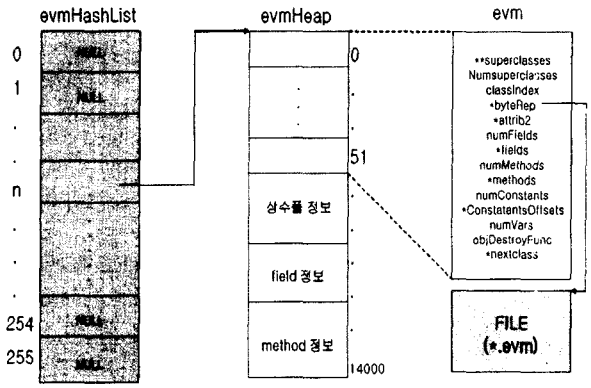
먼저, 중간코드인 *.evm을 입력으로 받게 되면 main함수는 VmInit함수를 호출하여 로더가 추출한 정보를 저장하기 위해 만들어 놓은 스택 및 힙을 초기화 한다. 초기화 성공여부를 다시 main함수에 알려주고, 초기화가 정상으로 이루어지면 main함수는 VmStartApp함수를 호출하여 로더를 실행하게 된다. 로더가 실행되면 중간코드인 *.evm을 읽어 들이고, 위치 이동을 위해 선언되어진 포인터 변수에 처음 위치에 대한 주소값을 넘겨주게 된다. 그런 후 로더는 중간코드 안을 이동하며 필요한 정보를 추출하게 되고, 추출되어 진 정보는 스택 및 힙에 저장되어진다.

3.2 내부 자료구조

실행 된 로더는 중간코드인 *.evm 내에서 필요한 여러 가지 정보를 추출하기 시작한다. 추출한 정보는 [그림 2]와 같이 두개의 힙 공간(evmHashList, evmHeap)과 하나의 구조체(EvmStruct* evm)에 저장된다.

먼저 evmHashList을 보면, evmHashList는 전역 포인터 배열로 선언되어 있다. 또한 처음 사용되는 구조체의 초기화를 시키며, evmHeap에 저장되어 있는 클래스 정보의 시작 주소를 가지고 있다.

evmHeap은 전역 포인터 변수로 선언되어 있으며, 중간코드인 *.evm의 모든 정보를 저장하게 된다. 이때 저장 되는 내용은 대부분 주소값을 저장하여 나중에 *.evm 내에서 필요한 정보를 빠르게 찾을 수 있도록 한다.



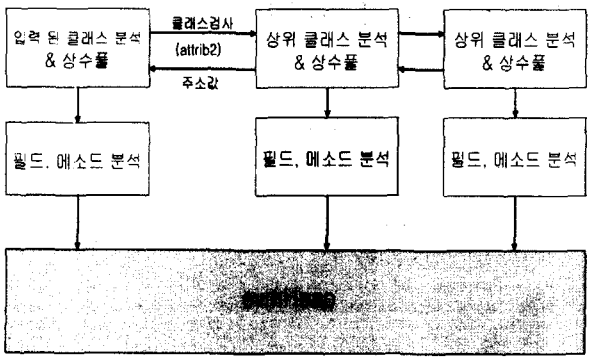
[그림 2] 로더의 출력 : 내부 자료 구조 구성도

evm은 클래스의 정보가 저장되는 구조체로서 새로운 클래스가 저장되면 반드시 저장되는 내용이다. 이 구조체를 통해 클래스의 기능과 위치를 알 수 있다. 구조체의 내용은 [그림 3]과 같다.

구조체 \ 내용	변 수	저장 내용
EvmStruct	superclasses	상위클래스의 주소 저장
	numsuperclasses	상위클래스의 개수 저장
	byteRep	입력된 evm파일의 시작주소를 저장
	attrib2	상위 클래스가 있는지의 판별을 위한 주소 저장
	numFields	필드의 개수를 저장
	fields	필드정보가 저장되어 있는 주소 저장

[그림 3] 로더 출력의 세부 구조체

로더는 중간코드인 *.evm에서 필요한 정보를 추출하여 evmHeap에 저장하는데, 저장 순서는 [그림 4]와 같다.



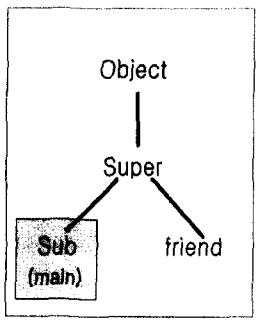
[그림 4] *.evm에 대한 메모리 적재 순서

먼저 main함수가 포함되어 있는 클래스의 분석 - evm 구조체에 저장 - 및 상수풀 내용이 evmHeap에 저장되고, 그런 다음 저장된 구조체의 변수중 attrib2값을 검사하여 상위 클래스가 있는지를 알아본다. 만약 상위 클래스가 존재한다면 그 상위클래스의 분석 및 상수풀 내용을 evmHeap에 저장하게 된다. 이런 식으로 최상위 클래스의 클래스 분석 및 상수풀의 내용까지 모두 저장하게 되면 최상위클래스의 필드와 메소드 정보부터 evmHeap에 저장되고, 이런 식으로 main함수가 포함되어 있는 클래스의 필드와 메소드 정보를 내려가면서 저장하게 된다.

3.3 예제

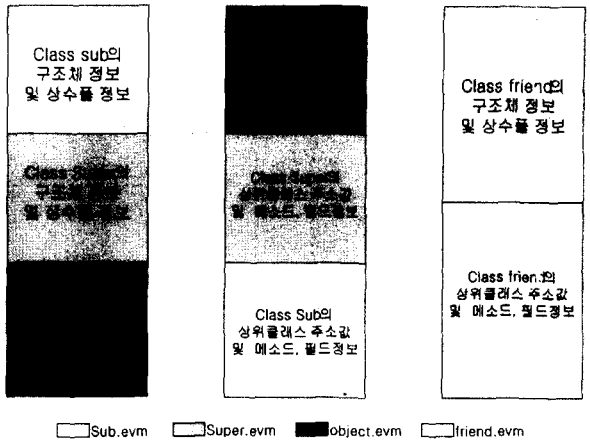
간단한 예제를 통해 구현된 로더가 중간코드인 *.evm의 정보를 evmHeap에 저장하는 결과는 [그림 6]과 같다.

```
class Super{
        int super_;
        public static void for_loop(int j){
                for(int i=0; j>i; i++){
                        APIClass.println(i);
                }
        }
public class Sub extends Super{
        int sub_;
        public static void main(String[] args){
                for_loop(5);
                friend.friendos();
        }
}
class friend extends Super{
        int friend_;
        public static void friends( ){
                for_loop(4);
        }
}
```



[그림 5] 예제 프로그램 및 관계도

예제 소스에서는 main함수를 포함하는 Sub 클래스와 그 상위 클래스인 Super 클래스, 그리고 Sub 클래스와 같은 위치인 friend 클래스를 작성하였다. 또한 [그림 6]에서의 object 클래스는 최상위 클래스를 나타낸다.



[그림 6] 로더를 통한 메모리 저장 형태

4. 결론 및 향후 연구

가상기계의 가장 큰 사용 목적은 오스 프로그램에 대한 실행 파일 형태를 플랫폼에 독립적으로 실행하기 위한 환경을 제공하는 것이다. 현재까지 다양한 종류의 가상기계가 개발되어 사용되고 있으며 그 사용 범위가 점차 확대되고 있다. 본 연구팀의 최종 목표는 기존의 가상기계를 기반으로 새롭게 임베디드 시스템에 적합한 가상기계를 개발하고자 한다. 이를 위해 다양한 연구 시도가 진행되고 있다.

본 논문에서는 기존의 가상기계 분석을 통하여 *.evm을 위한 로더의 설계 및 구현을 하였으며, 실행 효율을 위한 내부 자료 구조를 개선하였다. 차후에는 로더의 결과를 이용하여 인터프리터의 설계 및 구현중에 있으며, 특정 플랫폼에 포팅을 위한 요소를 분석하고 있다. 또한 구현된 로더의 성능을 평가하기 위한 부분을 보완중에 있다.

5. 참고 문헌

[1] "Java Virtual Machine". Jon Meyer & Troy Downing. March 1997
[2] "Virtual Machine Design and Implementation in c/c++", Blunden
[3] www.wabasoft.com, Waba JVM
[4] "Kaffe Java Virtual Machine" www.kaffe.org/
[5] "The Java Virtual Machine Specification", Second Edition. Tim Lindholm Frank Yellin.
[6] "자바가상기계 프로그래밍" Engel저 곽용재 역. Addison.