JSize: 유닉스의 size에 대응하는 자바 등가 프로그램

양희재
경성대학교 컴퓨터공학과

JSize: A Java Equivalent of the UNIX size program

Heejae Yang
Dept. of Computer Engineering, Kyungsung University
E-mail : hjyang@star.ks.ac.kr

요 약

JSize 는 유닉스 운영체제의 size 프로그램에 대응하는 자바 등가 프로그램이다. 유닉스 size 프로그램은 실행 파일을 분석하여 그 파일이 메모리에 적재되었을 때 얼마 정도의 코드와 데이터 영역을 차지할지를 예측하게 한다. 마찬가지로 JSize 는 자바 클래스 파일을 분석하여 그 파일이 메모리에 적재되었을 때 얼마 정도의 클래스 영역 메모리를 차지할지를 예측하게 하는 기능을 갖는다. 본 논문은 클래스 파일을 분석하여 얻은 정보로부터 클래스 영역의 크기를 예측할 수 있게 하는 원리에 대해 소개한다. 아울러 실험을 통해 JSize 의 예측 정확성을 실제로 알아보았다.

ABSTRACT

JSize is a Java equivalent of the Unix size program. The Unix size program analyzes an executable file and estimates the size of code and data segment when the file is loaded on memory. Likewise, JSize analyze a Java class file and estimates the size of class area when the file is loaded on memory. This paper presents the principles necessary to estimate the class area size with the information obtained from a class file. An experimental result is included to show the accuracy of estimation the JSize provides.

키워드

Java, Class File, Java Virtual Machine, Memory

Ⅰ. 서 론

유닉스 운영체제에서는 size 라는 프로그램이 기본 제공된다. 이 프로그램은 a.out 형식을 갖는 실행 파일을 분석하여 텍스트, 데이터, 그리고 bss 의 크기를 알려준다. 이 값들은 실행 파일이 메모리에 실제로 적재되었을 때 어느 정도의 메모리를 사용할 것인지 미리 예상할 수 있게 한다.[1)]

유닉스의 a.out 파일은 일반적으로 헤더와 텍스트(또는 코드), 데이터, 그리고 옵션인 심볼 테이블 등으로 구성된다. 이 형식은 유닉스 운영체제 규격에서 상세히 정의되어 있다.

자바 환경에서 실행 파일에 대응하는 것이 클래스 파일이다. 클래스 파일 형식 역시 자바가상기계 명세에서 엄격히 정의되어지고 있다. 클래스 파일은 헤더와 상수 풀 (constant pool), 클래스 정보, 필드 정보, 메소드 정보 등으로 구성되며, a.out 형식과는 전혀 다른 구조를 갖는다 [1].

클래스 파일을 읽어들여 클래스 정보나 필드 정보, 메소드 정보 등을 볼 수 있게 하는 프로그램들이 ClassViewer 라는 이름으로 다수 개발되어 현재 사용 중에 있다. 그러나 ClassViewer 는 단지 클래스 내부 정보 열람만을 가능하게 하며 유닉스의 size 와 같은 역할, 즉 클래스가 메모리에 적재될 때 어느 정도의 메모리를 사용할 것인지를 알게 하지는 않는다.

메모리 사용량을 미리 예측하는 것은 컴퓨터 시

1) 이 논문은 2003년도 정보통신부 지원 정보통신기초기술연구지원사업에 의해 연구되었음.

스템에서 중요한 일이며, 특히 임베디드 시스템과 같이 자원의 제약을 받는 시스템에서는 더욱 그렇다.

우리는 유닉스 운영체제의 size 와 마찬가지로 자바 클래스 파일이 메모리에 적재될 때 어느 정도의 메모리를 사용할 것인지를 미리 예상할 수 있게 하는 JSize 라는 프로그램을 개발하였으며, 이 논문에서는 JSize 의 기능과 구조, 메모리 사용량의 예측 정확도 등에 대해 설명한다. JSize 는 ClassViewer 와 같이 클래스 파일 내부 정보를 분석할 뿐 아니라 각각의 정보들이 메모리 사용량에 얼마 정도의 영향을 끼칠 것인지에 대해서도 분석하고 있다. simpleRTJ 자바가상기계 상에서 실제 적용해 본 결과 JSize 는 거의 모든 API 클래스에 대해 오차 범위 내에서 메모리 사용량을 정확히 밝혀 주었다.

본 논문의 구성은 다음과 같다. 2장에서는 유닉스 실행 프로그램의 메모리 모델과 자바가상기계에서의 메모리 모델 및 클래스 파일의 구조에 대해서 나타내었다. 3장에서는 JSize 의 구조와 기능에 대해 다루었으며, 4장에서 실제 실험을 통해 JSize 의 정확성을 살펴본다. 5장에서 이 논문의 결론과 함께 향후 연구 방향에 대해 설명한다.

II. 메모리 모델

유닉스를 비롯한 전통적 운영체제에서 메모리는 일차원 또는 이차원 배열로 인식되어지며, 배열의 인덱스 값이 메모리 주소에 해당된다.

한 프로그램의 실행을 위해서는 메모리에 세 가지 부분이 적재되는데, 이들은 각각 코드 부분, 데이터 부분, 스택 부분이다 (그림 1). 코드 부분에는 각종 명령어들이 놓여져 있으며, 데이터 부분은 초기화된 데이터들과, 초기화되지 않은 데이터 (bss) 등으로 구성된다. 스택은 지역변수의 저장이나 파라미터 전달, 복귀주소의 저장 등의 목적으로 사용된다.

프로그램 실행 전에 실행 파일에 대해 size 등과 같은 프로그램을 사용하면 이 파일이 메모리에 적재될 때 사용되는 코드 부분과 데이터 부분의 크기를 미리 알 수 있다. 이 크기는 프로그램 실행 시간 동안 늘 일정 크기를 가지며, 따라서 이 값을 참조함으로서 프로그램이 사용하는 메모리의 양을 어느 정도 짐작할 수 있다. 반면 스택의 크기는 가변적이며 프로그램 실행 시간 동안 끊임없이 변한다. 그 외에도 프로그램 실행 시 필요에 따라 힙 메모리를 할당받기도 한다.

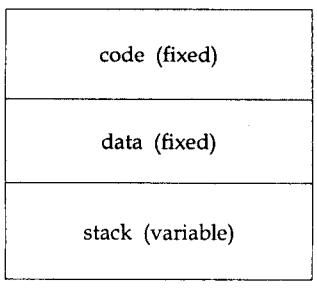

그림 1 전통적 메모리 모델

자바는 위와 전혀 다른 메모리 모델을 따른다. 자바에서 메모리는 클래스 영역, 자바 스택 영역, 힙 영역, 네이티브 메소드 스택 영역 등 네 가지로 나뉘어진다 [2]. 클래스 영역은 클래스에 대한 거의 대부분의 정보들이 놓여지는 곳이다. 이곳에는 유닉스 메모리 모델에서 코드 부분에 대응되는 바이트코드들이 놓이며, 그 외에도 클래스의 상속관계, 접근범위, 상수, 필드, 메소드 등이 위치한다. 이 영역은 프로그램 실행 시간 동안 항상 일정 크기를 갖는다.

자바 메모리 모델에서 자바 스택 영역, 힙 영역, 네이티브 메소드 스택 영역 등은 모두 그 크기가 가변적이다. 프로그램 실행 시 이들의 크기는 끊임없이 변한다.

유닉스의 size 프로그램이 고정 크기를 갖는 코드 부분과 데이터 부분의 크기를 예측할 수 있게 하는데 비해 자바에서는 고정 크기를 갖는 클래스 영역의 크기를 알 수 있게 하는 프로그램이 알려지지 않고 있다. 유닉스 메모리 모델에서 코드 부분과 데이터 부분은 그림 1과 같이 매우 단순한 구조를 갖는데 비해 클래스 영역은 그림 2와 같이 클래스 정보, 상수 정보, 필드 정보, 메소드 정보 등 매우 복잡한 구조를 갖고 있기 때문이다.

클래스 정보, 상수 정보, 필드 정보, 및 메소드 정보 등은 클래스 파일을 분석함으로서 밝힐 수 있다. 이미 많은 ClassViewer 프로그램들이 개발되어서 이런 정보들을 조회할 수 있도록 하고 있다.

그러나 ClassViewer 프로그램은 이들 정보들을 알 수 있게만 할 뿐 이들이 실제 메모리에서 얼마나 많은 메모리를 사용할 지에 대해서는 어떤 예측도 하지 않는다. 즉 현재까지 클래스 영역의 크기를 예측할 수 있는 프로그램은 존재하지 않았다.

본 연구에서 개발한 JSize 는 그림 2의 클래스 영역의 크기를 예측하고자 하는 목적의 프로그램이다. 유닉스의 size 프로그램은 그림 1의 메모리

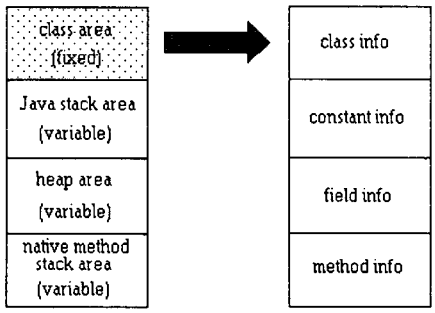

그림 2 자바 메모리 모델 및 클래스 영역 구조

모델에서 고정 크기를 갖는 코드와 데이터의 크기를 말해주며 가변 크기를 갖는 스택이나 힙 메모리의 크기에 대해서는 말해주지 않는다. 마찬가지로 JSize 는 그림 2의 메모리 모델에서 고정 크기를 갖는 클래스 영역의 크기를 말해주며 가변 크기를 갖는 자바 스택 영역이나 힙 영역, 그리고 네이티브 메소드 스택 영역의 크기에 대해서는 말해주지 않는다.

III. JSize 구조

3.1 클래스 파일

그림 2와 같이 클래스 영역이 차지하는 메모리는 클래스 정보, 상수 정보, 필드 정보, 및 메소드 정보 등으로 구성된다. 각각의 정보들을 얻기 위해서 JSize 는 ClassViewer 와 마찬가지로 먼저 클래스 파일을 분석한다. 클래스 파일 구조는 자바가상기계 명세에서 정한 바에 따라 헤더, 상수 풀, 클래스 정보, 필드 정보, 메소드 정보 등 모두 다섯 개 부분으로 구성된다 [2].

헤더는 매직 번호와 버전 번호로 이루어지는 8 바이트의 고정된 영역이며, 상수 풀은 연산의 오퍼랜드로 사용되는 일반 상수 외에 클래스나 필드, 메소드 등의 이름과 형식 등을 나타내는 각종 상수들로 이루어진다. 이들 상수들은 클래스 적재 시 다른 클래스들과의 링크 목적으로 사용되며, 실행 시 형식 확인 등의 목적으로도 일부 사용된다. 클래스 정보는 이 클래스의 접근 제어값, 자신 및 상위 클래스의 이름, 구현하고 있는 인터페이스의 이름 등을 나타내며, 필드 정보와 메소드 정보는 각각 이 클래스가 갖는 필드와 메소드에 대한 정보를 가지고 있다.

클래스 파일이 메모리의 클래스 영역에 놓일 때는 다른 클래스와의 연결, 즉 레졸루션(resolution)이 완료된 상태이므로 상수 풀의 내용 중 많은 부분이 생략되어진다. 헤더 부분도 확인이 끝난 후이므로 역시 생략된다. 나머지 부분은 보다 접근하기 쉬운 형태로 변형되어 메모리에 적재된다.

3.2 메모리 사용량의 예측

클래스 영역의 내용은 클래스 파일에서 얻어지는 것이므로 이 영역의 크기, 즉 메모리 사용량은 클래스 파일의 분석을 통해 예측할 수 있다. 여기서는 클래스 영역 내의 각 정보들이 차지하는 메모리의 양을 분석해보자.

클래스 영역의 내용은 클래스 파일 내용 그대로가 아니라 보다 접근이 편한 형식으로 변형된 반영이미지 (mirror image) 형식을 취하므로 구현 시스템마다 차이가 있을 수 있다. 그러므로 모든 시스템에 정확히 적용될 수 있는 메모리 사용량 예측식의 유도는 불가능하다. 대신 JSize 는 클래스 파일에서 얻어지는 정보들 중 어느 시스템에서나 필수적으로 들어갈 내용만을 지적하고 그들이 차지할 메모리의 양만을 예측한다. 따라서 실제로는 JSize 가 예측하는 메모리의 양보다 다소 더 큰 크기의 메모리가 사용될 것으로 판단된다.

먼저 그림 2에서 클래스 정보를 이루는 요소들에 대해 고찰해보자. 주요 요소들은 다음과 같다.

- ☞ 접근제어값, 이 클래스의 인덱스, 상위 클래스의 인덱스, 필드 개수, 메소드 개수 (5개)
- ☞ 상수 테이블, 필드 테이블, 메소드 테이블
- ☞ 위 테이블을 가리키는 포인터들 (3개)

각 요소 당 4바이트를 가정하면 클래스 정보의 크기는 최소 {(5+3)x4 + 테이블 전체 크기} 가 된다. 테이블의 각 엔트리들도 4바이트를 차지한다고 가정하면 테이블 전체 크기는 $4(c+f+m)$ 이 된다. 여기서 c, f, m 은 각각 상수, 필드, 메소드의 개수를 의미한다. 따라서 클래스 정보가 차지하는 메모리의 양 M_k 는 $M_k = 32+4(c+f+m)$ 이다.

다음으로 그림 2의 상수 정보에 대해 분석해보자. 클래스 파일의 상수 풀 내용 중에서 레졸루션을 위한 정보들은 클래스 적재 시 이미 이용된 상태이므로 이들이 메모리에 있을 필요는 없다. 즉 메모리에는 바이트코드 실행 시 실제로 오퍼랜드로 사용되는 상수들만 있다고 가정한다. 각각의 상수는 실제 값과 더불어 형식을 나타내는 꼬리표와 길이값 등이 추가된다. 역시 각각의 4바이트라고 하면 상수 한 개당 8바이트의 추가부담이 붙게 되며, i번째 상수의 길이를 cl_i 라 하면 사용 메모리의 합은 $8+cl_i$ 가 된다. 따라서 모든 상수에 대한 메모리의 사용량 M_c 는 $M_c = \sum_{i=0}^{c-1}(8+cl_i) = 8c+\sum_{i=0}^{c-1}cl_i$ 가 된다.

세 번째는 필드 정보이다. 필드 정보에서 필수적 내용은 접근제어 값과 필드의 형식, 그리고 개별 필드의 인덱스 값 등이다. 각각의 크기를 4바이트라고 하면 필드 한 개당 12바이트가 되며, 전체 필드 정보의 크기 M_f 는 $M_f = 12f$ 바이트가 된다.

마지막으로 메소드 정보를 분석해보자. 메소드 정보에서 필수적 내용은 접근제어 값, 형식, 스택과 지역변수배열의 크기, 바이트코드의 길이 등이다. 각각의 크기는 4바이트라고 하면 메소드 한 개당 20바이트의 메모리를 사용하며, 여기에 실제 바이트코드가 포함되면 하나의 메소드는 $20+ml_i$ 바이트의 메모리를 필요로 한다. 여기서 ml_i 는 i번째 메소드의 바이트코드 길이다. 따라서 전체 메소드 정보의 크기 M_m 은 $M_m = \sum_{i=0}^{m-1}(20+ml_i) = 20m+\sum_{i=0}^{m-1}ml_i$ 가 된다.

따라서 어떤 클래스가 그림 2의 클래스 영역에서 차지하는 메모리의 양 M 은 다음과 같다.

$$M = M_k + M_c + M_f + M_m$$
$$= 32+4(c+f+m) + \quad // \text{ class info}$$

$$
\begin{aligned}
& 8c + \sum_{i=0}^{c-1} cl_i + && // \text{ constant info} \\
& 12f + && // \text{ field info} \\
& 20m + \sum_{i=0}^{m-1} ml_i && // \text{ method info} \\
& = 32 + (12c + \sum_{i=0}^{c-1} cl_i) + 16f + (24m + \sum_{i=0}^{m-1} ml_i)
\end{aligned}
$$

3.3 JSize 의 구조 및 기능

앞에서 우리는 클래스 영역의 크기를 나타내는 일반적 수식을 유도하였다. JSize는 일반적인 ClassViwer 프로그램과 마찬가지로 클래스 파일을 읽어들이고 그 파일로부터 수식에 필요한 값들, 즉 c, f, m, cl_i, ml_i 를 구하는 구조를 갖는다. 구한 값들은 위 수식에 대입되어 클래스 영역의 크기를 출력하게 된다.

다만 위에서 구한 수식에서 상수값들은 구현된 자바가상기계마다 다소 차이가 날 수 있다. 즉 어떤 시스템에서는 접근제어값을 2바이트로 하고, 어떤 시스템에서는 4바이트로 하는 등의 차이가 있을 수 있다는 것이다. 또한 경우에 따라 추가적 정보들을 포함할 수도 있다. 이 차이는 위 수식에서의 상수값(32, 12, 16, 24 등)에 영향을 미치지만 그 효과는 그다지 크지 않은 것으로 실험을 통해 밝혀졌다.

IV. 실험 및 분석

JSize 가 얼마나 정확히 클래스 영역의 크기를 예측하는지 알기 위해 simpleRTJ [3] 라는 내장형 자바가상기계에 대해 적용해 보았다. 표1은 simpleRTJ API 클래스 중 java.lang 패키지에 포함되어있는 주요 클래스들에 대한 실험 결과를 나타낸 것이다. 이 표에서 볼 수 있듯이 예측값(total)과 실제값(actual)이 모든 클래스에 대해 거의 근접하고 있음을 확인할 수 있다. 주목할만한 또 다른 사실은 클래스 영역의 크기 중 상당 부분을 메소드 정보가 차지하고 있으며, 기타 상수나 필드의 영향은 상대적으로 미미하다는 것이다.

표 1 simpleRTJ API 클래스들에 대한 실험

	class	const	field	method	total	actual
Boolean	32	33	48	347	460	496
Integer	32	85	48	922	1,087	1,160
Object	32	0	0	303	335	400
String	32	16	64	2,852	2,964	3,012
Thread	32	0	48	689	769	900
Throwable	32	0	16	209	257	292
Exception	32	0	0	59	91	104
Error	32	0	0	59	91	104
Arithmetic-Exception	32	0	0	59	91	104
Internal-Error	32	0	0	59	91	104

V. 결 론

본 논문에서는 자바 클래스 파일이 메모리에 적재되었을 때 클래스 영역의 메모리를 얼마나 사용할지를 분석해주는 JSize 프로그램의 구조 및 원리에 대해 고찰하였다. 유닉스 운영체제에서 사용되는 size 프로그램과 유사한 역할을 담당하지만, 유닉스 등에서 사용되는 전통적 메모리 모델과 자바의 메모리 모델은 전혀 다르므로 기존 방법으로 메모리 사용량을 예측할 수는 없다. 본 논문은 자바 메모리 모델을 설명하고 클래스 영역의 메모리 사용에 영향을 주는 각종 요소를 분석하였으며, 클래스 파일로부터 이들 요소를 추출하여 메모리 사용량을 예측할 수 있는 원리를 제시한 것이다. 향후 연구는 클래스 영역과 같은 정적 메모리 뿐 아니라 동적 메모리의 크기로 예측할 수 있는 프로그램의 개발 등을 들 수 있다.

참고 문헌

[1] 양희재, 자바가상기계, 한국학술정보(주), 2001년 3월, ISBN 89-5520-342-4

[2] T. Lindholm and F. Yellin, The Java Virtual Machine Specification, Addison-Wesley 1997

[3] 양희재, "simpleRTJ 임베디드 자바가상기계의 ROMizer 분석 연구," 한국정보처리학회 논문지, 제 10-A 권 5호, 2003년 10월