파일 시스템이 지원되지 않는 실시간 운영체제에서의 KVM 설계 및 구현

조희남[0], 양희권, 성영락[†], 이철훈
충남대학교 컴퓨터공학과, [†] 국민대학교 전자정보통신공학부
(hncho[0], hkyang, chlee)@ce.cnu.ac.kr, [†] yeong@mail.kookmin.ac.kr

Design and Implementation of KVM on Real-Time Operation System without File System

Hee-Nam Jo[0] , Hui□ Kwon Yang, Yeong Rak Seong[†] ,and Cheol-Hoon Lee
Parallel Processing Laboratory Dept. of Computer-Engineering, Chung Nam National Univ.
[†] School of Electrical Engineering, Kookmin Univ.

요 약

오늘날 많은 가전제품들은 다양화, 쌍방성, 효율성, 개방성등 정보화 시대의 주요 특징들과 접목되어 생산되고 있으며, 이중 모바일 제품군(Low End)은 각방받는 주류 제품이라 할 수 있다. 그러나 대부분의 모바일 제품들은 그 특성상 파일 시스템을 지원하지 못하고 있기에, 본 논문에서 모바일 제품의 운영체제로서 파일 시스템이 지원되지 않는 실시간 운영체제(RTOS- Real Time Operation System)를 선택하여 가상머쉰(VM-Virtual Machine)을 설계 및 구현하였다. 위의 지원을 위해 자바 클래스 파일을 C 언어 형태로 변환하여 VM 과 직접적으로 링크하는 방법인 프리링킹(Prelinking), 프리로딩(Preloading) 혹은 로마이징(ROMizing)이라고 알려져 있는 기법을 도입하였다.

1. 서 론

모바일 제품군인 모바일 폰(Mobile Phone), 무선 장치(Wireless Devices)들의 특징은 하루게 다르게 변화하고 있다. 예전 모바일 장치들의 특징은 소프트웨어가 내부에 고정되어 유연적이지 못하였지만, 오늘날의 제품들은 소프트웨어를 동적 다운로드하여 구매자의 개성에 맞게 구성할 수 있다. 또한 정보화 시대의 특징에 맞게 제품의 다양화가 이루어지고 있으며, 통신 쌍방성, 개방성등 그 특성들은 더 부각되고 있다. 따라서, 이러한 모바일 제품들의 특징들을 지원하기 위한 API 로서 J2ME(Java™ 2 Micro Edition)와 가상머쉰인 KVM(Kilo -Virtual Machine)의 중요도는 커진다고 볼 수 있다. 그러나 대부분의 모바일 제품들은 그 특성상 파일 시스템을 지원하지 못하는 제한된 환경을 지니고 있기에 이를 만족시키는 운영체제 및 가상머쉰를 필요로 한다.

본 연구는 모바일 제품의 운영체제로서 파일 시스템이 지원되지 않는 실시간 운영체제(RTOS)위에 가상머쉰을 설계 및 구현하는데 그 목표를 두고 있다. 이러한 조건을 만족시키기위해서 자바 클래스 파일을 C 파일 형태로 변화하여 가상머쉰과 직접적으로 링크할 수 있는 방법인 로마이징 기법을 사용하였다. 본 논문의 구성은 다음과 같다. 2 장에서 실시간 운영체제의 간단한 특징, 로마이징 기법, 그리고 KVM 에 대해 살펴보고 3 장에서는 설계 구현된 가상머쉰에 대해 기술하며, 4 장에서 실험결과를 보여주고, 마지막으로 5 장에서는 결론 및 향후과제에 대해 기술한다.

* 본 논문은 한국과학재단이 지정한 지역협력 연구센터(RRC)인 충남대학교 소프트웨어 연구센터의 지원으로 수행된 과제의 결과입니다.

2. 관련 연구

포팅하게 될 기반 운영체제인 실시간 운영체제에 대해 살펴보고, KVM 의 일반 개념과 로마이징에 대해 기술한다.

2.1 실시간 운영체제

실제적으로 포팅하게 될 기반 실시간 운영체제는 iRTOS™ 이며, iRTOS™ 는 실시간 운영체제의 핵심인 멀티 태스킹, 및 ITC(InterTask Communications)환경을 제공하고 있다. 멀티태스킹 환경이라 함은 여러 개의 태스크(Task)를 동시에 실행할 수 있는 스케쥴링(Scheduling) 정책을 제공함을 의미하며, ITC 환경은 태스크간의 상호 협력을 위한 동기화(Synchronization) 및 통신 세마포어(Semaphore) , 메시지 메일박스(Message Mailbox), 메시지 큐(Message Queue) 등을 지원함을 의미한다. 아래는 이러한 iRTOS™ 의 특징들을 간략하게 요약한 것이다.

(1) 멀티태스킹
우선 순위에 의한 태스크 선정방식이며, 동일한 태스크에 대해서는 라운드 로빈 방식을 따른다.
(2) 세마포어
공유자원관리에 필요한 메커니즘이며, 동기화 및 상호배제(Mutual Exclusion)를 위해서 사용된다.
(3) 메시지 큐
특정 태스크간 혹은 ISR 상에서 다른 태스크로의 메시지 전송을 할 수 있도록 해준다.
(4) 메시지 메일박스
메시지 큐의 특수한 형태로써, 빠른 메시지 전송을 위해 사용된다.

iRTOS™는 임베디드 시스템에서 수행 할 수 있도록 하기 위해서 실행 이미지(Image)를 최소화하였다. 또한 Linux™ 나 Windows™ 같은 일반 운영체제와는 달리 실시간 운영체제인 iRTOS™ 은 시간 결정성을 보장할 수 있도록 설계되었다. [1]

2.2 KVM 과 로마이징

가상머쉰이란 어느 특정 하드웨어나 운영체제에 의존적이지 않은체 일을 수행할 수 있는 구조를 가진 추상적인 개념이다. 소프트웨어 머쉰(Software Machine)이라고도 불리며 운영체제의 위에서 작동된다. 이러한 가상머쉰의 특징은 다음과 같다.

(1) 플랫폼 독립적
(2) 하드웨어와 프로그램의 분리
(3) 소프트웨어의 동적 다운로드
(4) 추가적인 보안을 제공
(5) 하드웨어나 플랫폼의 복잡성을 숨김

가상머쉰은 구현은 주로 C 나 C++언어로 구현이 되며, 민감한 성능 부분에 있어서는 어셈블리 언어(Assembly Language)을 사용한다. [2]

CLDC(Connected, Limited Device Configuration)의 디바이스들은 대부분 파일 시스템이 없다. 이를 위해 자바에서는 JCC(JavaCodeCompact)라는 유틸리티를 제공하는데 일반적으로 프리로더, 프리링커 또는 로마이저라고 알려져 있다. 로마이저는 KVM 의 모든 시스템 클래스들을 사전에 링킹시킬 수 있도록 해줌으로써 응용 프로그램의 시작을 빠르게 할 수 있도록 도와준다. [3]

본 논문에서는 시스템 클래스뿐만 아니라 실행할 응용 프로그램의 클래스까지 같이 로마이징하여 사용할 수 있도록 VM 을 수정하여 설계 및 구현한 논문이다.

3. 설계 및 구현

SUN™ 에서 제공하고 있는 KVM 은 크게 Unix™ 용과 Windows™ 으로 나뉜다 할 수 있다. 이를 iRTOS™ 맞게 설계, 구현하기 위해서는 클래스 로딩과 기타 부분을 플랫폼과 운영체제에 맞게 변경해주어야 한다. 이 논문에서 다루고자 하는 설계, 구현의 범위는 로마이징과 관련된 부분으로만 제한한다.

3.1 로마이징

우선 간단한 응용 프로그램인 Hello.java 를 javac 로 컴파일한 후, JCC 유틸리티를 이용하여 로마이징할 수 있다. [그림 3.1.1] 보게되면 로마이징 대상 클래스 수가 133 개 로 나와있는데 이는 132 개의 CLDC API 클래스 외에 Hello 클래스가 추가되어 133 개의 클래스를 대상으로 로마이징 했음을 보여 주고 있다.

```
make[2]: Entering directory
`/lab3/PP/hncho/KVM/j2me_cldc/api'
make[2]: Leaving directory
`/lab3/PP/hncho/KVM/j2me_cldc/api'
... classes.zip
make[2]: Entering directory
`/lab3/PP/hncho/KVM/j2me_cldc/tools/jcc'
make[2]: `nativeFunctionTableUnix.c' is up to date.
make[2]: Leaving directory
`/lab3/PP/hncho/KVM/j2me_cldc/tools/jcc'
make[2]: Entering directory
`/lab3/PP/hncho/KVM/j2me_cldc/tools/jcc'
... ROMjavaUnix.c
java -classpath classes JavaCodeCompact ₩
-arch KVM -o ROMjavaUnix.c classesUnix.zip
Total rename saving = 0
133 Total Classes
934 method blocks
29602 bytes of Java code
85 catch frames
350 field blocks
1772 constant pool entries
270 Java strings
```

[그림 3.1.1] 로마이징 결과

로마이징 결과물로는 c 언어 형태인 ROMjavaUnix.c 와 nativeFunctionTableUnix.c 라는 2 개의 c 파일이 생성되며 KVM 을 설치할 때 가상머쉰과 직접 링킹되어 사용된다.

3.2 KVM 설계 및 구현

KVM 과 로마이징의 관계에 있어 고려되어야 부분은 크게 세 부분으로 나뉜다 할 수 있다.

첫째, 로마이저에 의해 생성된 ROMjavaUnix.c 에 Hello 클래스에대한 포인터를 추가해 주어야 한다. 이는 가상머쉰 내에서 파일시스템에서 Hello 클래스를 로딩하는 것이 아니라 로마이저에 의해서 생성된 파일에서 직접적으로 로딩할 수 있어야 하기 때문이다. [그림 3.2.1]은 ROMjavaUnix.c 에 Hello 인스턴스 클래스(Instance Class)를 삽입한 모습을 보여주고 있다.

```
// … 6개의 나머지 시스템 클래스에 대한 포인터 …
INSTANCE_CLASS JavaLangError =
(INSTANCE_CLASS)&AllClassblocks.java_lang_Error;
INSTANCE_CLASS Hello =
(INSTANCE_CLASS)&AllClassblocks.Hello;
```

[그림 3.2.1] Hello 인스턴스의 삽입

둘째, 첫번째 과정에서 ROMjavaUnix.c 에 추가된 Hello 인스턴스 클래스에 대한 포인터를 가상머쉰내부에서도 선언해주는 부분이 필요하다. 즉 Hello 인스턴스를 전역변수처럼 가상머쉰 내에서 자유롭게 사용할 수 있도록 해주기 위해서다. [그림 3.2.2]는 Hello 인스턴스에 대한 포인터(Pointer)를 가상머쉰 내에 선언한 것을 보여주고 있다.

```
// class.h 내부
extern INSTANCE_CLASS JavaLangObject;
extern INSTANCE_CLASS Hello;
// class.c 내부
EXTERN_IF_ROMIZING INSTANCE_CLASS Hello;
EXTERN_IF_ROMIZING INSTANCE_CLASS JavaLangObject;
```

[그림 3.2.2] 가상머쉰내부로의 Hello 인스턴스 포인터 삽입

셋째 가상머쉰내부에서 클래스 로딩, 쓰레드(Thread), 그리고 응용프로그램 실행 절차에 대한 수정이 필요하다. 클래스 로딩에 있어 설정된 환경변수들은 모두 무시해도 좋다.

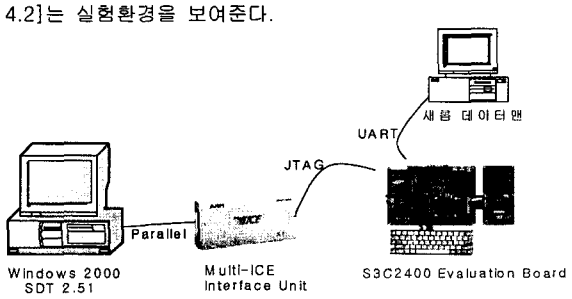
이유는 이미 로마이저에 의해서 생성된 C 파일에서 시스템 클래스들과 응용 프로그램의 클래스에 대한 모든 포인터를 가상머쉰 내부에 지니고 있기 때문이다. 따라서 클래스 로딩에 대한 초기화 및 자원해제는 더 이상 필요가 없게 된다. 쓰레드 부분에 있어서는, 두번째 단계에서 끝마친 Hello 인스턴스에 대한 포인터를 인자로 한 그린 쓰레드(Green Thread) 를 생성해야 한다. 이때 오직 하나의 피지컬 쓰레드(Physical Thread)가 같이 생성되게 되며 Hello 에 대한 바이트코드를 해석하는 역할을 담당하게 된다. [그림 3.2.3]은 셋째 단계에서 이루어지는 절차에 따르는 변경된 모습을 보여주고 있다. 그림을 살펴보면 클래스 로딩은 파일시스템이 존재하지 않는 운영체제에 맞게 주석처리 되어 더 이상 파일시스템에서는 클래스를 로딩하지 않음을 알 수 있다. 따라서 실행하고자 하는 클래스인 Hello 인스턴스도 파일 시스템에서 클래스 로딩하는 것이 아니라 이미 가상머쉰 내부에서 선언된 포인터를 할당받으면 된다.

```
// InitializeClassLoading();
InitializeJavaSystemClasses();
InitializeEvents();
mainClass =Hello;
```

[그림 3.2.3] 가상머쉰의 초기화 및 Hello 인스턴스의 로딩

4. 실험결과

본 연구는 ARM-920T CPU가 탑재된 S3C2800 보드 상에 SDT2.51 ARM 용 통합개발환경을 사용하여, 실시간 운영체제인 iRTOS™ 와 가상머쉰을 포팅하여 실험하였다. [그림 4.2]는 실험환경을 보여준다.



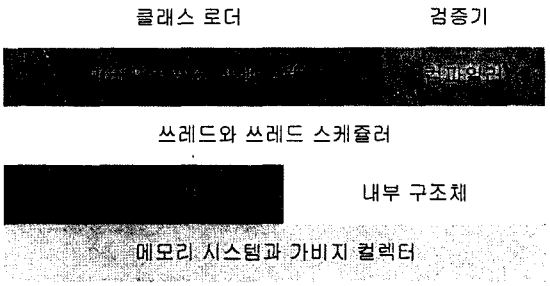
[그림 4.2] 실험환경

[그림 4.3]의 실험결과 화면을 살펴보면 프리로딩된 시스템 클래스 목록과 Hello 인스턴스 클래스를 보여주고 있으면 결과 화면이 그 아래에 위치하고 있음을 알 수 있다.

[그림 4.3] 실험 결과 화면

5. 결론 및 향후과제

KVM 은 8 개의 필수 부분으로 나뉜다고 볼 수 있다. 본 논문은 8 개의 필수 요소 중 파일 시스템과 관련된 클래스 로딩 부분을 다루웠다. 본 연구를 통해 살펴본 바와 같이, 파일 시스템을 지원하지 않는 디바이스에서 로마이징 기법을 사용하여 가상머쉰을 구현해본 결과, 응용 프로그램을 문제없이 실행시킬 수 있으며, 알려져 있는 바와 같이 직접적으로 클래스 파일로부터 로딩할 경우보다 속도면에서도 월등한 차이를 갖게 된다. 이는 파일 로딩시 걸리는 속도와 검증 시간이 제외되기 때문이다.



[그림 5.1] 가상머쉰 8 개필수 구성요소

현재 컴파일러(Compiler)부분을 제외한 나머지 6 개 요소 부분에 대해서도 설계, 구현 중에 있으며 이들 모든 부분을 종합하여 iRTOS™에 맞는 최적화된 가상머쉰을 개발하는 것을 최종 목적으로 하고 있다.

참 고 문 헌

[1] http://www.inestech.com.
[2] Antero Taivalsaari,Sun Microsystemds, Inc. , *Designing Virtual Machines for Mobile Devices*, February 4,2003.
[3] Sun™, *KVM Porting Guide, CLDC, Version 1.04 Java™ 2 Platform, Micro Edition*, October 2002.
[4] Bill Venners, *Inside the Java™ Virtual Machine Second Edition*, 1999.
[5] Tim Lindholm•Frank Yellin, *The Java™ Virtual Machine Specification Second Edition*, April 1999.