내장형 리눅스 상에서의 퍼스널자바 플랫폼의 개발

성민영 김태현 김소영 김광영 김형수 장래혁 신현식
서울대학교 컴퓨터공학부
minyoung@snu.ac.kr

PersonalJava™ on the Embedded Linux

Minyoung Sung Taehyoun Kim Soyoung Kim Kwangyoung Kim
Hyungsoo Kim Naehyuck Chang Heonshik Shin
School of Computer Science & Engineering, Seoul National University

요 약

자바는 플랫폼 독립성, 높은 보안 수준, 멀티 쓰레드 지원 등의 다양한 장점을 가진 언어로서 인터넷 응용에서 이제 차세대 내장형 시스템을 위한 실행 환경으로 기대를 모으고 있다. 특히 퍼스널자바는 셋톱박스나 PDA 등의 내장형 응용을 위해 경량화된 자바 가상 기계로서 현재 다양한 운영체제 및 하드웨어에 활발히 이식되고 있다. 본 논문은 내장형 시스템을 위한 플랫폼으로서 내장형 리눅스상에서의 퍼스널자바 수행 환경을 제안하고 그 개발 내용을 기술한다. 내장형 리눅스는 기존 상용 실시간 운영체제에 비해 비용이 매우 저렴하며 소스 코드가 공개되어 있어 다양한 수준의 수정 및 개발이 가능하다는 장점을 가진다. 본 논문에서는 이러한 내장형 리눅스에 퍼스널자바를 이식함에 있어 구축된 개발 환경 및 이식에 적용된 개발 방법과 내용을 소개한다.

1. 서론

자바 응용 프로그램은 플랫폼 독립적인 자바 바이트 코드(bytecode)로 구성되어 특정 플랫폼 상에서만이 아니라 자바 가상 기계(JVM; Java Virtual Machine)가 구현되어 있는 어떠한 플랫폼에서도 수행될 수 있는 장점을 가지고 있다. 또한 보안, 멀티 쓰레드 지원, 가베지 콜렉터 등 많은 장점에 힘입어 자바는 인터넷 응용 뿐 아니라 내장형 시스템(embedded system)을 위한 응용 개발용으로까지 그 영역을 확대해 나가고 있다. 썬 마이크로시스템즈에서는 이미 내장형 시스템을 위한 퍼스널자바(PersonalJava)[1]를 발표하고, 셋톱박스나 PDA와 같은 내장형 시스템에 적용하고 있다.

본 논문은 내장형 시스템을 위한 플랫폼으로서 내장형 리눅스(embedded Linux)상의 퍼스널자바 수행 환경을 제안한다. 내장형 리눅스는 기존의 리눅스를 내장형 응용에 맞게 경량화시킨 운영체제로서, VxWorks[2], pSOS, Windows CE와 같은 기존의 상용 실시간 운영체제(RTOS)에 비해 비용이 매우 저렴하다는 장점을 갖는다. 또한 상용 RTOS의 경우, BSP나 디바이스 드라이버의 주요 기능이 바이너리 형태로 제공되어 수정이 어렵고 최신 기술에 대한 BSP 지원이 늦는 것에 반해 리눅스는 소스 코드가 공개되어 있어 운영 체제나 드라이버의 수정이 용이하고 전세계의 수많은 개발자들에 의해 최신의 하드웨어/소프트웨어 기술이 신속히 지원되는 등 많은 장점을 지니고 있어 차세대 RTOS 솔루션으로 떠오르고 있다. 그러나, 리눅스는 기존 RTOS보다 메모리 요구량이 크며, MMU 기능이 없는 프로세서에는 이식이 어렵다는 단점도 존재한다. 특히, 범용이 아닌 내장형 시스템의 개발을 위해서는 독자적으로 개발환경을 구축해야 하는 어려움이 따른다. 본 논문에서는 내장형 리눅스 응용을 위해 구축한 개발 환경을 소개하고 퍼스널자바 이식에 따른 개발 내용을 상세 기술한다

본 논문의 구성은 다음과 같다. 2 절에서는 퍼스널자바의 리눅스로의 이식을 위한 개발환경을 소개한다. 3 절에서는 이식에 따른 기술적 어려움을 설명하고 이의 해결방법을 기술한다. 본 논문은 4 절 결론 및 향후 진행 방향을 끝으로 마친다.

2. 퍼스널자바의 리눅스로의 이식을 위한 개발환경

2.1 목표보드의 사양

그림 1은 개발에 사용된 실험용 목표보드(target board)를 보인 것이다. 목표보드는 모토롤라의 50 MHz

MPC860 CPU를 탑재하고 있으며, 64 MB SDRAM, 내장 기본 8 MB 플래쉬 메모리, 256 KB SGRAM, 4 KB 명령어/데이타 캐쉬, 직렬 통신 제어기, 이더넷 인터페이스를 기본 구성으로 한다. 디스플레이로는 640x480 해상도의 LCD를 장착했다.

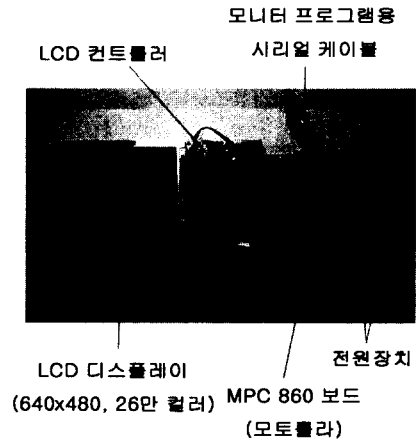

그림 1. 목표 보드

2.2 개발환경 및 방법

진행과정은 다음과 같다. 먼저, 퍼스널자바를 데스크탑용 리눅스(CPU: i386)에 이식하는 작업을 진행하여 OS 의존적인 부분을 수정하였다. 다음, 이를 다시 MPC860 용 내장형 리눅스에 이식하는 작업을 진행하여 CPU 의존적인 부분을 수정하였다. 이는 데스크탑 환경이 개발에 용이함에 착안하여 대부분의 작업을 데스크탑 개발환경에서 진행함으로써, 개발기간과 소요를 절감하기 위한 것이었다. 개발환경은 그림 2와 같다. PowerPC용 크로스 컴파일러를 이용하여 퍼스널자바 및 리눅스 커널을 빌드한다. 리눅스 커널은 부트스트랩로더와 함께 부트롬에 저장되어 목표시스템을 가동시킨다. 그리고, 목표시스템에서의 퍼스널자바로의 접근은 NFS를 통해 이루어진다. 즉, 그림의 리눅스(NFS)서버에서 소스 코드 수정 작업을 하고 실행 이미지를 빌드하면, 목표보드는 NFS를 통해 리눅스서버에 접근하여 생성된 퍼스널자바 이미지를 실행시키는 방식이다.

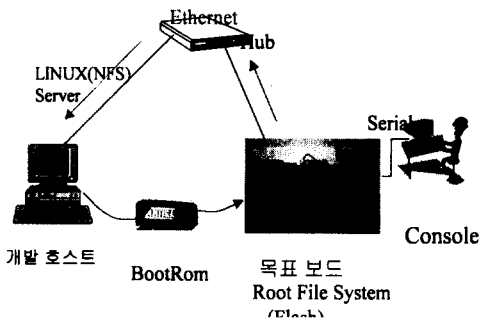

그림 2. 개발 환경

3. 퍼스널자바의 리눅스로의 이식

3.1 퍼스널자바의 구조 및 주요 이식 부분

자바 수행 환경은 호스트 프로그래밍 인터페이스를 기준으로 플랫폼 의존적인 부분과 독립적인 부분으로 구분되며, 이식 작업은 플랫폼 의존적인 부분의 수정 위주로 진행된다(그림 3 참조). 주요 이식 부분으로 쓰레드(thread)와 모니터(monitor), 네이티브 메쏘드 호출(native method call), 네이티브 API, AWT의 구현 등이 있다.

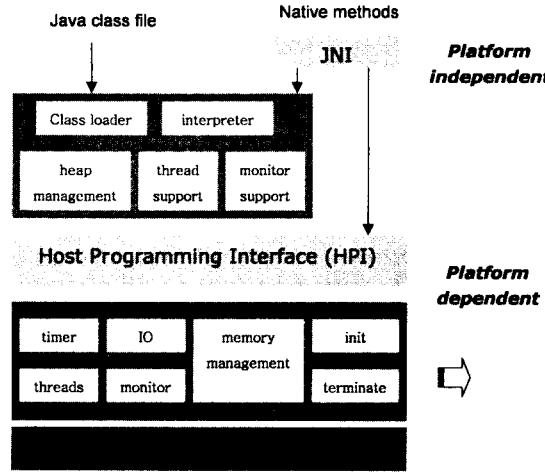

그림 3. 자바 수행 환경의 내부 구조

3.2 이식 개발 내용

본 절에서는 이식 관련 주요 개발 내용으로 쓰레드와 모니터의 구현, 네이티브 메쏘드 호출 수정, AWT의 구현 내용을 기술한다.

- **쓰레드와 모니터의 구현**

퍼스널자바의 자바쓰레드 구현은 사용자-수준 쓰레드 패키지인 그린 쓰레드(Green thread)를 이용하는 버전과 네이티브 쓰레드를 이용하는 버전의 두가지가 존재한다. 본 개발에서는 두가지 메커니즘을 모두 지원하여, 목표 시스템에 적합한 방식을 선택할 수 있도록 하였다.

그린 쓰레드는 다수의 쓰레드가 하나의 커널-수준 프로세스에 대응되는 다대일 방식으로서, 쓰레드간 문맥교환은 사용자 수준에서 `setjmp()/longjmp()` 호출을 통해 이루어진다. 쓰레드의 문맥이 사용자 수준에서 관리되므로, 낮은 오버헤드로 많은 수의 쓰레드를 생성할 수 있다는 장점을 갖는다. 그러나, 하나의 쓰레드가 블록(block)되면 전체 쓰레드들이 블록되는 문제가 있다. 따라서, 블록킹 가능성이 있는 I/O 관련 시스템 서비스를 논블록킹(nonblocking)하게 처리해야 하는 어려움이 존재한다. 이식을 위해서는 `setjmp()/longjmp()`에 사용되는 레지스터 집합을 목표시스템의 CPU에 맞추어야 한다. 이를 위해 `gcontext_t`라는 새로운 타입을 정의하여 문맥을 표현하도록 하고, `getcontext()`와 `setcontext()`함수를 구현하여 문맥 교환에 사용될 수 있도록 하였다.

네이티브 쓰레드는 하나의 쓰레드가 하나의 커널-수준 쓰레드(혹은 프로세스)로 대응되는 일대일 방식으로서, 모든 쓰레드 관리는 커널 수준에서 이루어진다. 네이티브 쓰레드 이식을 위해서는 목표 플랫폼에 커널-수준 쓰레드가 존재해야 한다. 본 개발에서는 리눅스용 커널-수준 쓰레드 패키지인 LinuxThreads를 이용하였다. LinuxThreads는 현재까지 상당한 안정성이 입증되었고, 특히 POSIX 쓰레드 표준을 따르므로 이식이 용이하다는 장점을 지닌다. 리눅스에서의 커널-수준 쓰레드는 **clone()** 시스템 호출을 통해 생성된다. 특이한 점은 리눅스의 경우 커널-수준 쓰레드 관리에 별도의 자료구조를 두지 않고, 프로세스와 마찬가지로 태스크 제어 블록을 이용한다는 것이다. 따라서 프로세스와 쓰레드가 혼용되는 독특한 구조를 갖는다. 특히 clone()을 사용하면 부모-자식 관계가 형성되지 않으며, **waitpid()**를 사용할 수 없다는 제약을 갖는다. 이는 부모 자바 쓰레드가 새로 생성한 쓰레드의 종료를 기다리는 **join()** 메소드의 구현을 어렵게 한다. 본 개발에서는 리퍼(reaper) 쓰레드라고 하는 부모 프로세스를 두고 모든 쓰레드의 생성을 전담하여 부모-자식 관계를 맺도록 하였다. 쓰레드가 종료하면 **waitpid()**를 통해 리퍼 쓰레드가 알게되고, 종료된 쓰레드에 대해 **join()**을 기다리는 쓰레드에게 조건 변수(condition variable)로 알려주는 방식으로 이러한 문제를 해결할 수 있었다. 이 밖에, LinuxThreads에는 쓰레드를 suspend/resume하는 기능이 없으므로, 시그널을 이용하여 구현해 주었다. 모니터는 LinuxThreads의 뮤텍스(*mutex*)와 조건변수를 이용하여 쉽게 구현될 수 있었다.

- **네이티브 메쏘드 호출 수정**

퍼스널자바 내에는 네이티브 함수의 주소와 인자들의 타입 및 값을 전달받아 실제로 호출하는 부분이 있다. **sysInvokeNative()**라는 함수가 그것인데, 원래의 퍼스널자바 배포본에는 Sparc 용으로 최적화된 어셈블리 소스만이 존재했다. 본 개발에서는 PowerPC로의 이식을 위해 **sysInvokeNative()**의 C언어 버전을 구해 수정하였다. 특히 long과 double타입은 64비트를 사용하는데 원래의 C 소스는 32비트를 가정하여 작성되어 있었다. 64비트를 지원하기 위해서 이를 수정하여 long과 double 타입에 대해 2개의 자바 스택 슬롯이 할당되도록 하였다.

- **AWT의 구현**

AWT(Abstract Windows Toolkit)은 자바의 그래픽 지원을 위한 라이브러리를 뜻하는 것으로서, 퍼스널자바에서는 Win32 API를 이용한 버전과 Motif를 이용한 버전의 두가지 구현을 제공하고 있다. 특히 최근에는 플랫폼 의존적인 코드를 줄이고, 많은 부분을 자바로 구현한 Truffle을 제공하여 AWT의 이식성을 높이고 있다. 그림 4는 Truffle의 구조를 기존의 AWT 구조와 비교한 것이다. 그러나, Truffle도 여전히 Win32나 X Windows API를 가정하고 있다. 이는 그래픽 지원이 전혀 없거나, 경량의 윈도우 시스템만을 갖추고 있는 내장형 시스템으로의 이식을 어렵게 한다. 본 개발에서는 플랫폼 그래픽 시스템으로서 내장형 시스템을 고려한 경량 윈도우 시스템인 마이크로윈도우즈(microwindows) [3]를 채택하였다. 마이크로윈도우즈는 특히 Win32와 유사한 API를 제공하므로 Truffle의 Win32 버전을 이용하는 것이 이식에 유리했다. 마이크로윈도우즈와 Truffle을 인터페이스하는 작업과 동시에 마이크로윈도즈를 목표 시스템의 디스플레이 하드웨어에 맞추는 작업이 진행되었다. 이는 마이크로윈도우즈가 리눅스의 프레임버퍼를 사용하도록 하고, 프레임버퍼를 LCD 출력주소에 매핑함으로써 이루어졌다.

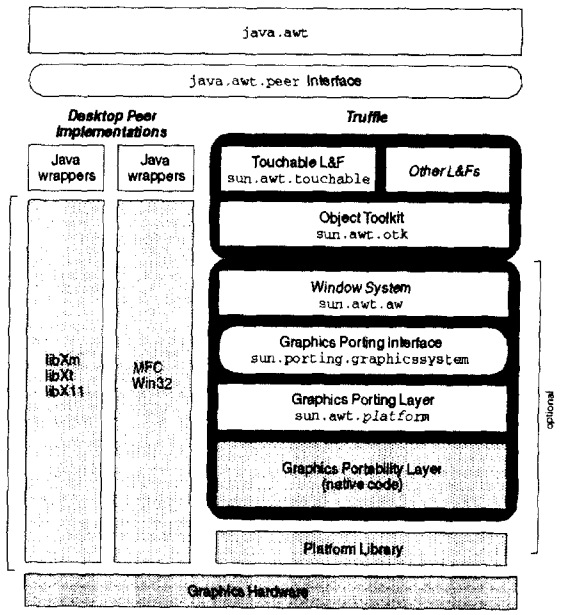

그림 4. Truffle구조의 AWT

4. 결론 및 앞으로의 진행 방향

본 논문에서는 내장형 응용을 위한 플랫폼으로서 내장형 리눅스상의 퍼스널자바 수행환경을 제안하고, 이를 위한 개발환경 및 개발내용을 소개하였다. 퍼스널자바를 목표 시스템에 이식하기 위해 그린/네이티브 쓰레드의 이식, 네이티브 메쏘드 호출 수정, AWT의 구현 등의 작업이 진행되었다. 현재까지 퍼스널자바 코어의 이식이 완료되었으며 호환성 검사툴(PersonalJava Cmpatibility Kit)을 통해 안정성이 입증되었다. 현재, AWT에서 터치스크린 방식의 입력을 지원하도록 하기 위한 작업이 진행중이다.

참고문헌

[1]PersonalJava,http://java.sun.com/products/peronaljava/index.html.
[2]강희구외, "내장형 시스템을 위한 실시간 자바 쓰레드의 구현 및 성능 평가," 정보과학회 춘계학술대회, pp 95-97,2000.
[3]microwindows, http://microwindows.censoft.com.