자바 기반의 임베디드 리눅스 소프트웨어 개발 환경 구축

이진관, 노시영, 박덕규, 박상준, 이종찬, 박기홍
군산대학교 컴퓨터정보공학과
e-mail: leejinkwan@kunsan.ac.kr

JAVA Based a Software Development Environment for Embedded Linux Systems

Jin-kwan Lee, Si-Yong No, Deok-Gyu Park, Sang-Jun Park, Jong-Chan Lee, Kihong Park
Dept of Computer Information Engineering, Kunsan National University

요 약

임베디드 시스템을 지원하는 JAVA API를 개발하면 임베디드 시스템을 개발자가 JAVA를 사용함으로서 코드의 재사용, 객체지향 개념의 시스템 개발들을 가능하게 한다. JAVA API를 구현하는데 있어 시스템에 의존적인 부분들이 존재하게 되는데, 이는 native 함수에서 구현한다. 본 논문에서는 리눅스 기반의 임베디드 시스템 디바이스를 제어하기 위한 JAVA API를 구현하는데 있어 플랫폼 독립적인 자바 부분과 의존적인 native 부분으로 나누어 설계 및 구현하였고, 임베디드 시스템 디바이스의 JAVA API를 통한 제어에 초점을 두었다.

키워드 : 임베디드 시스템, JAVA API

I. 서론

임베디드 시스템은 홈네트워킹, 헬스케어, U-씨티 등 다양한 어플리케이션이 개발되고 있다. 이러한 임베디드 시스템에는 하드웨어와 소프트웨어가 결합된 고기능의 전자제어 시스템을 내장하고 있다. 이렇게 임베디드 시스템이 소형화 되고, 고성능으로 발전함에 따라 응용 범위도 넓어지고 있고, 많은 연구들이 진행되고 있다. 이러한 하드웨어 적인 측면의 발전과 달리, 소프트웨어는 아직도 전문적인 기술과 하드웨어에 의존적인 프로그램을 벗어나지 못하고 있다. 즉, 임베디드 시스템 기반의 어플리케이션 개발은 대부분 C/C++로 이루어지고 있어 플랫폼에 의존적인 개발이 된다. 또한 객체지향 개념을 도입하는 것 또한 어려운 실정이다[7].

따라서, 임베디드 시스템를 지원하는 JAVA API를 개발하면 기존 JAVA 개발자와 현재 임베디드 시스템을 개발하는 개발자들도 JAVA를 사용함으로서 코드의 재사용, 객체지향 개념의 시스템 개발들을 가능하게 한다. JAVA API를 구현하는데 있어 시스템에 의존적인 부분들이 존재하게 되는데, 이는 native 함수에서 구현한다.

본 논문에서는 리눅스 기반의 임베디드 시스템 디바이스를 제어하기 위한 JAVA API를 구현하는데 있어 플랫폼 독립적인 자바 부분과 의존적인 native 부분으로 나누어 설계 및 구현하였고, 임베디드 시스템의 디바이스를 JAVA API를 통해 제어하는데에 초점을 두었다. 플랫폼에 의존적인 native 부분은 JNI(Java Native Interface)를 통해 구현하였다. JNI란 자바와 자바 이외의 언어로 만들어진 애플리케이션이나 라이브러리가 상호 작동할 수 있도록 연결시켜주는 인터페이스로 자바에서 지원하지 못하는 플랫폼 종속적인 기능들과 이미 다른 언어로 만들어진 애플리케이션이나 라이브러리를 사용할 수 있게 한다. JNI를 통한 자바 네이티브 프로그램은 자바가 아닌 다른 프로그래밍 언어들로 자바 객

체를 생성하고, 검사하고, 갱신하는 것과 다른 언어에서 자바 메소드 호출을 가능하게 하고 클래스를 로드하고 클래스 정보를 얻을 수 있도록 하는 것 그리고 런타임 타입 체크를 수행하는 것 등을 가능하게 해준다[3][4].

본 논문에서는 임베디드 시스템의 플랫폼에 상관없이 어플리케이션에서 이용할 수 있는 JNI와 JAVA API를 통해 임베디드 리눅스 시스템을 위한 자바 개발환경을 구축하였다.

II. 기존 연구

자바는 인터넷의 빠른 확산과 더불어 차세대 컴퓨팅 플랫폼으로 인식되고 있는 기술이다. 자바의 장점은 플랫폼 독립성과 연결성, 동적 다운로드 그리고 보안성이 있다. 기존의 프로그래밍 언어들은 해당 플랫폼에 의존적인 바이너리 파일들로 컴파일 해야만, 해당 시스템에서 실행할 수 있다. 그러나 자바는 플랫폼에 관계없이 단 한번의 컴파일을 통해서 얻은 바이트 코드로 모든 환경에서 동일하게 실행 될 수 있다. 이러한 특징은 서비스 개발자와 제공자, 그리고 사용자에게 편리함과 효율성을 제공해준다. 서비스 개발자는 모든 플랫폼에 맞추어 프로그램을 작성하지 않아도 되며, 서비스 제공자는 동일한 프로그램을 서로 다른 플랫폼들에 제공하기 위해 여러 종류의 바이너리 파일들을 제공해야 할 필요가 없다. 또한 사용자는 임의의 프로그램을 설치하고자 할 때, 자신의 플랫폼에 알맞도록 프로그램을 찾아야 할 필요가 없으며, 자바로 작성된 프로그램은 무엇이든 필요할 때 네트워크를 통해 다운로드 받아서 사용할 수 있다.

보통 자바 플랫폼은 자바 프로그램이 실행되는 하드웨어와 OS 및 소프트웨어 환경에 따라 나뉜다. 그러나 보통 하드웨어나 소프트웨어적인 것보다는 지원되는 API의 기능에 따라서 구별된다. 즉, 자바가상머신이 동작하는 곳이 브라우저인지 자바 OS인지 보다는 지원되는 API가 자바 프로그램에 있어서는 더 중요하다. 자바 제반 기술의 핵심인 자바 가상머신은 일종의 소프트웨어 CPU로, 자바 번역기라고도 불린다. 자바 가상머신은 컴파일 된 자바 바이트 코드와 컴퓨터 운영 시스템 간의 번역기 역할을 하며, 자바의 플랫폼 독립성을 보장한다.

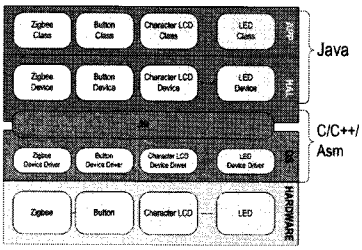
임베디드 시스템을 지원하는 자바가상머신으로 JamVM이 있다. JamVM은 C로 작성되어 있으며, 적은양의 플랫폼 종속적인 어셈블러로 되어 있다. 그리고 다른 아키텍처로 쉽게 포팅될 수 있다.

현재 급부상 중인 임베디드 시스템을 객체 기반 자바로 구현하는 것은 플랫폼 독립성, 네트워크 연결성, 안전성 등의 자바의 장점을 제공할 수 있다. 기존 임베디드 시스템 개발은 임베디드 리눅스 상에서 GTK나 C/C++을 이용한다. 그러므로 기술 구현의 난이도가 높을 뿐만 아니라, 코드의 재사용성 또한 떨어진다.

본 논문의 실행환경은 임베디드 리눅스 상의 자바 플랫폼 환경이다. JVM은 JamVM을 이용하였고, 자바 API 개발은 J2SDK 1.4 버전을 이용하였다. 자바 API는 자바 플랫폼의 구성요소이므로, 다른 요소인 JVM과 밀접한 연관이 있다. 이들 사이의 연결자 역할은 JNI를 사용함으로써 가능하다. JNI를 통하여, API들은 JVM의 특성을 이용하고 원하는 값을 얻으며, C 라이브러리와 리눅스 상에서 제공하는 시스템 호출을 이용 할 수 있다.

III. 자바 API 설계 및 구현

본 논문에서 제안한 자바 API의 구조는 두 부분으로 구성된다(그림 1 참조). 하나는 API를 구성하는 인터페이스와 클래스들을 제공하는 자바 부분(*.java)이고 다른 하나는 각각의 디바이스들을 제어하기 위한 플랫폼 의존적 시스템 호출 위해 JNI 형식을 사용하여 C나 C++로 구현한 네이티브 라이브러리이다. 공통 자바 API는 하위단인 디바이스와 상위단인 자바 어플리케이션의 인터페이스를 수행함으로서 상위 계층이 하위 계층과 독립적인 프로그래밍이 가능하도록 지원한다.



〈그림 1〉 자바 API 구조

리눅스 기반의 디바이스를 제어하기 위하여 디바이스를 커널에 포팅시키기 위한 드라이버를 제작해야 한다. 이를 위하여 디바이스 드라이버를 모듈방식으로 제공함으로서 커널의 유연한 확장성을 제공할 필요가 있다[5]. 본 논문에서는 크로스 컴파일러로 임베디드용 CLCD, CC2420, 버튼 드라이버를 제작하였다. CLCD를 통해 문자열을 출력하고, CC2420을 통해 지그비 통신을 가능케 하였다. 또한 버튼 클릭 시, 특정 기능을 처리할 수 있도록 버튼 이벤트를 추가하였다.

어플리케이션 자바 API에서 하드웨어에 종속적인 JNI에 상관없이 JNI를 이용하기 위해 공통 자바 API를 설계하였다.

- IDevice.class
디바이스 장치를 나타내는 클래스로 인터페이스로 구현하였다. 모든 디바이스는 IDevice를 구현해야 한다.
- AbstractCharacterLCDDevice.class
AbstractCharacterLCDDevice 클래스는 칩셋 이름, CLCD의 라인수, 라인 당 표시할 수 있는 문자수 지정 속성과 CLCD에 출력할 수 기능을 갖는다.
- IPacket.class
IPacket 클래스는 패킷의 사이즈를 반환하는 메서드를

제공하는 인터페이스이다. 통신 디바이스 관련 클래스는 IPacket 클래스를 상속 받아야 하며, 어플리케이션에서 IPacket 클래스를 구현하여 필요한 데이터를 추출해야 한다.

- ICommunicationDevice.class

통신 디바이스는 OSI 7계층의 데이터 링크 계층과 동일한 기능을 수행한다. Packet 단위로 통신을 함으로 IPacket이라는 인터페이스를 사용하여 클래스를 운용한다. 또한 통신 디바이스 클래스는 패킷의 송수신을 여부를 파악하기 위하여 Runnable 인터페이스를 상속하여 이를 감시토록 한다.

- ICommand.class

디바이스 제어를 위한 명령 인터페이스로 디바이스의 기능을 실행하는 메서드를 제공한다. 디바이스의 제어 명령을 하나의 인터페이스로 묶기 위한 인터페이스로 각각의 제어 명령 하나하나를 이 인터페이스를 통하여 구현한다.

- ITranslator.class

하위 레이어인 데이터통신계층에 쓰이는 Packet과 그 위 계층에서 사용하는 명령 사이를 변환시켜주는 기능을 정의한 인터페이스이다. ICommand를 IPacket형태로 변환시키는 메서드가 제공된다.

CLCD는 제조사마다 다른 명령어 셋을 사용하여 디바이스를 제어함으로 디바이스가 변경될 때마다 서로 다른 명령어셋을 사용해야 한다. 하지만 명령어셋을 따로 분리하여 인터페이스를 만들 경우, 이종의 디바이스를 위한 JNI 작업 시간이 상당히 단축될 수 있다. 또한 AbstractCharacterLCD 형식에 맞도록 구현한다면 서로 다른 디바이스를 사용하여 프로그래밍할지라도 장치 관련 클래스(예, LCD_II)를 인스턴스화하여 CharacterLCD 생성 시 인자(Parameter)로 넘겨주어 응용프로그램의 기능에 전혀 영향을 미치지 않고 그대로 사용할 수 있게 된다.

- LCD_II.class

AbstractCharacterLCDDevice 클래스를 상속받아 어플리케이션에 사용할 수 있는 클래스로 CLCD의 On/Off 및 초기화, 출력 관련 메소드를 제공한다.

- ILCD_IICommandSet.class

LCD II Command Set 관련 상수를 정의한 인터페이스이다[2].

- CharacterLCD.class

CharacterLCD 클래스는 총 슬롯 개수, 슬롯 추가 삭제 시 사용되는 카운터, 해시 테이블을 이용한 맴버 그룹 저장소, AbstractCharacterLCD Device 객체를 속성으로 갖는다. 모든 컨텐츠는 member 형태로 존재하며 memberGroup에 속한다. member, memberGroup의 추가/삭제 메소드, member의 CLCD 출력 메소드를 제공한다.

- IMemberContent.class

member에 포함되는 컨텐츠 인터페이스로 문자열이 저장된다.

- CharacterLCDMember.class

ClaracterLCDMember 클래스는 CLCD에 들어갈 문자열을 가지는 클래스로 시작 컬럼, IMemberContent 인스턴스를 속성으로 갖는다.

- CharacterLCDMemberGroup.class

CharacterLCDMemberGroup는 CLCD의 라인수와 CharacterLCDMember 인스턴스를 갖는다.

IV. 시스템 실험

실험 환경은 호스트 컴퓨터와 타겟 컴퓨터로 구성된다. 호스트 컴퓨터는 RedHat FC4 운영체제에서 gcc와 J2SDK 1.4 버전의 개발툴을 이용하였다. 타겟 컴퓨터는 하이버스의 XHyper270-TKU를 이용하여 임베디드 시스템용 JAVA API를 개발하였고, 임베디드 시스템의 JVM은 JamVM을 이용하였다. JamVM은 GNU Classpath 자바 클래스 라이브러리를 이용하게 설계되었다[1].

우선 이들을 통합하기 전에 각각의 요소의 개별적인 동작을 테스트해야 하므로, 구현한 자바 API의 테스트를 위해 다음과 같은 테스트 환경을 설정하였다.

타겟 시스템 : XHyper270-TKU
커널 : Linux 2.6
JVM : JamVM 1.4.5
Classpath : GNU Classpath 0.9.3
호스트 컴퓨터 : IBM 호환 PC
OS : Redhat FC4
자바 컴파일러 : J2SDK 1.4
IDE : Eclipse 3.3

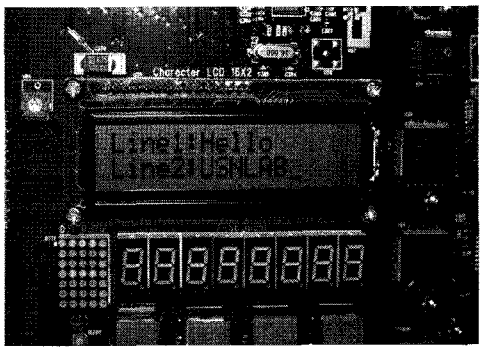
자바 API를 구현하였으므로, 이들 API가 올바르게 동작하는지를 알아보기 위하여 간단한 임베디드 어플리케이션으로 테스트를 한다.

<그림2>에서 CLCDTest는 테스트용 클래스로 main 메소드만을 가진다. ①은 공통 자바 API인 AbstractCharacterLCDDevice 클래스를 상속받아 어플리케이션에 사용할 수 있는 네이티브 LCD_II 객체를 생성한다. LCD_II 클래스는 CLCD의 On/Off 및 초기화, 출력 관련 메소드를 제공한다. ②는 네이티브 LCD 객체를 인수로 하여 어플리케이션 영역에서 사용할 수 있는 CLCD 객체를 생성한다. ③은 CLCD 화면에 출력할 2줄의 컨텐츠을 생성한다. ④⑤⑥에서는 멤버그룹을 등록하고, 생성된 컨텐츠를 멤버그룹에 등록한다. ⑦은 CLCD에 출력하는 메소드이다.

<그림3>은 XHyper270TKU에 임베디드용 자바 API를 이용하여 제작한 CLCDTest를 실행한 화면이다. 본 논문에서 연구한 임베디드용 자바 API를 활용하면 다양한 임베디드용 어플리케이션을 개발할 수 있다.

```
public class CLCDTest {
  public static void main(String[] args){
    AbstractCharacterLCDDevice nativeLCD = new LCD_II(); ---①
    CharacterLCD clcd = new CharacterLCD(2, nativeLCD); ---②
    IMemberContent line1Content = new LineContent("Line1:Hello World"); ---③
    IMemberContent line2Content = new LineContent("Line2:USNLAB");
    CharacterLCDMemberGroup Line1 = new CharacterLCDMemberGroup(0, 16, ""); ---④
    CharacterLCDMemberGroup Line2 = new CharacterLCDMemberGroup(1, 16, "");
    Line1.addMember(new CharacterLCDMember(0,line1Content)); ---⑤
    Line2.addMember(new CharacterLCDMember(0, line2Content));
    clcd.addMemberGroup(Line1); ---⑥
    clcd.addMemberGroup(Line2);
    clcd.displayAll(); ---⑦
    try {  Thread.sleep(1000);  }
    catch (InterruptedException e) { e.printStackTrace();  }
    clcd.cursorOnOff(true);
  }
}
```

〈그림 2〉 CLCDTset.java



〈그림 3〉 임베디드 시스템 결과 화면

V. 결론

본 논문에서 구현한 자바 API 들은 네트워크 연결성, 플랫폼 독립성, 클래스 동적로딩, 안전성 등의 자바 장점을 임베디드 시스템에 제공함으로서 홈 네트워크, U-씨티, 헬스케어 구축에 큰 기여를 할 것으로 예측된다.

본 연구에서 구현한 JAVA API는 임베디드 시스템 디바이스 중 주요 기능 만을 구현 한 것으로서 추후 다양한 임베디드 시스템 디바이스를 지원할 수 있는 JNI와 자바 API을 추가할 필요가 있다. 위에서 언급한 바와 같이 버튼 디바이스 경우 상위 단계인 자바에서 직접 디바이스를 감시하는 클래스를 인스턴스화하여 감지하지만, 추후 독자적인 데몬 형태의 JNI기반 네이티브 라이브러리를 생성하여 ButtonEventListener가 PollThread와 ButtonHook의 지원 없이도 ButtonEvent가 발생하도록 하려한다.

참고문헌

[1] http://jamvm.sourceforge.net/
[2] http://www.stanford.edu/class/ee281/handouts/hd44780_lcd_controller_datasheet. pdf
[3] http://java.sun.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html
[4] Java(TM) Native Interface: Programmer's Guide and Specification by Sheng Liang, Sun Microsystems
[5] http://www.xml.com/ldd/chapter/book/
[6] http://www.waset.org/pwaset/v26/v26-66.pdf
[7] 남상엽, PXA270을 이용한 임베디드 시스템 구조 및 응용, 상학당, 2007.
[8] Bill Venners, Inside the Java Virtural Machine 2nd, McGraw-Hill, 1997.