실시간 운영체제 iRTOS에서의 CVM 네트워크 설계 및 구현

Design and Implementation of Network in CVM on Real-Time Operation System, iRTOS

임재석, 이철훈*
충남대학교 컴퓨터공학과

Lim jae-seok, Lee cheol-hoon*
Dept. of Computer Engineering,
Chungnam National Univ.

요약

임베디드 시스템이 발전함에 따라 다양한 플랫폼을 가진 임베디드 디바이스에서 플랫폼 독립성을 위한 자바 기술이 급속도로 발전하고 있다. SUN 사의 CDC (Connected Device Configuration) 에 정의된 CVM (Classic Virtual Machine)은 이러한 플랫폼 독립적인 자바 환경을 제공한다. 특히 셋톱박스나 스마트폰과 같은 임베디드 시스템에서는 네트워크 기능을 위해 CDC의 기본 프로파일인 FP(Foundation Profile)를 사용한다. 본 논문에서는 실시간 운영체제 iRTOS에서 네트워크 기능을 구현하기 위한 네트워크 API인 FP의 네이티브 메소드에 대해 설계 및 구현한 내용을 기술한다.

Abstract

According to the development of the embedded system, a java technology which has a various platform has developed rapidly on the embedded device. CVM(Classic Virtual Machine) which is defined in the CDC(Connected Device Configuration) of the SUN mircosystems provides a java environment that is independent of a platform. Specially, embedded devices like a set-top box or a smart phone can be implemented by using FP(Foundation Profile) in CDC for a network faculty. In this paper, we design and implement native methods of FP which are network's API for implementation of CVM network on Real-Time Operating System iRTOS.

I. 서 론

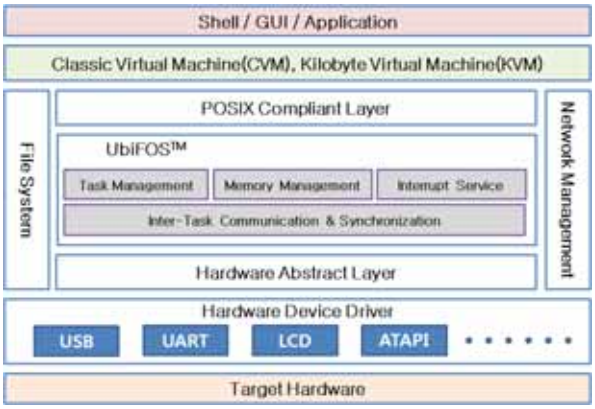
최근 임베디드 디바이스를 살펴보면 플랫폼 독립성, 보안성, 네트워크 이동성, 실행코드의 재사용성, 작은 실행파일 크기, 동적 적응성, 이식성, 개발의 용이성 등의 장점을 가진 자바를 지원하기 위해 CVM의 탑재가 증가하는 추세이다. 또한 하드웨어 성능 향상으로 데이터 처리 속도가 증가하고 인터넷 장비와 기술의 발전으로 인한 전송속도의 눈부신 향상으로 많은 정보를 네트워크를 통해 빠르게 제공받게 되었다.

다양한 플랫폼을 가진 임베디드 디바이스들은 네트워크 지원을 위해 J2ME(Java 2 Micro Edition)에서 정의하는 여러 개의 프로파일 중 FP(Foundation Profile)를 사용한다. FP 는 네트워크를 위한 표준 API를 java.net패키지에 정의하고 있다. FP 설계 및 구현 시 *i*RTOS 내에 구현되어 있는 경량 TCP/IP 네트워크 스택인 LWIP(A Light Weight TCP/IP stack)를 이용한다. 본 논문에서는 *i*RTOS상에서의 CVM네트워크를 위해 FP의 네이티브 메소드를 설계 및 구현하고자 한다. 2장에서는 관련연구로서 CVM의 기반 운영체제인 실시간 운영체제 *i*RTOS와 CDC, FP에 대한 개념을 기술하고, 3장에서는 CVM내의 FP를 통한 네이티브 메소드의 설계 및 구현을, 4장에서는 테스트 환경 및 결과를 기술한다. 마지막으로, 5장에서는 결론 및 향후 연구과제에 대해서 기술한다.

II. 관련연구

1. 실시간 운영체제 iRTOS

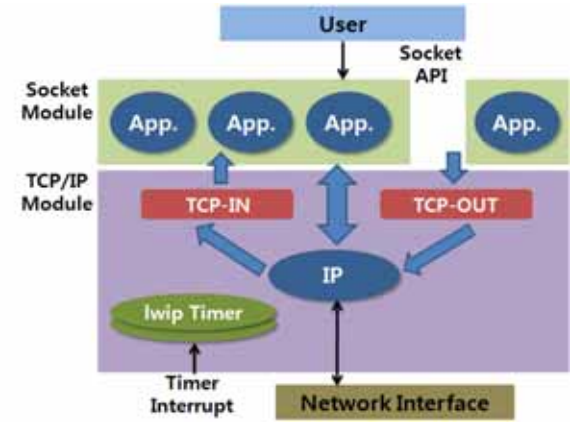
*i*RTOS는 선점형 우선순위 기반의 실시간 운영체제(Real-time Operating System)이다. 태스크는 중요도에 따라 0부터 255까지 256단계의 우선순위가 부여되며 가장 높은 우선순위의 태스크가 CPU를 점유하여 수행한다. 또 태스크 관리 기능과 가변/고정크기의 메모리 관리 기법을 제공하며, 그 외에 태스크 간 통신을 위한 메시지 메일박스, 메시지 큐, 메시지 포트, 태스크 포트, 시그널 등과 태스크 동기화를 위해 세마포와 이벤트 플래그를 제공하고 있다. 그림 1. 은 *i*RTOS의 기능을 블록 다이어그램으로 나타낸 것이다.[1]



▶▶ 그림 1. iRTOS의 기능 블록 다이어그램

2. LWIP(A Light Weight TCP/IP stack)

LWIP 는 스위스의 SICS(Swedish Institude of Computer Science)의 CNA(Computer and Network Architecture)연구실에서 개발한 독립적으로 구현된 경량 TCP/IP프로토콜 스택으로 *i*RTOS에 구현되어 있다. LWIP는 TCP의 모든 기능을 제공하면서 40Kbyte 정도 크기의 소스코드와 수십 Kbyte 크기의 데이터만을 필요로 하는 임베디드 시스템을 위한 네트워크 스택이다.[2]



▶▶ 그림 2. LWIP in iRTOS

LWIP는 크게 네 가지의 구성요소로 나눌 수 있다. 첫 번째는 IP, ICMP, UDP, TCP, DHCP, ARP 등의 프로토콜 처리 모듈이고, 두 번째는 메모리 관리 시스템, 세 번째는 하위 네트워크 인터페이스 함수들이며, 마지막으로 운영체제 에뮬레이션 계층이다. 그림 2. 는 *i*RTOS에서 동작하는 LWIP의 모습이다.

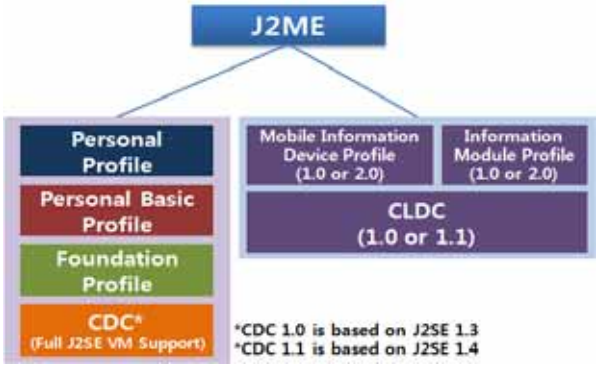
3. FP(Foundation Profile)

네트워크 연결이 필요한 임베디드 디바이스들과 전자제품 등에서 사용되는 CDC를 위한 API세트의 명세서가 바로 FP이다. 이는 특정 장치의 완벽한 실행 환경 제공과 추가 클래스 라이브러리 없이 장치 위에서 동작 가능한 API세트의 제공을 그 목적으로 하며 하드웨어 시스템마다 각각의 프로파일이 존재한다. FP라는 이름이 의미하듯 다른 CDC기반 프로파일을 구축하기 위한 토대가 된다. 또한 CDC의 인터페이스와 클래스를 모두 제공할 뿐 아니라, 보안(java.security.cert, java.security.acl, java.security.interface, java.security.spec), 유틸리티(java.util, java.util.jar, java.util.zip), 로케일(locale)클래스를 추가하여 구성을 확장한다. 특히 FP에는 스트림 기반 소켓과 HTTP연결에 필요한 java.net클래스가 포함된다. 하지만 AWT나 Swing을 비롯한 어떤 UI 클래스도 제공하지 않으므로 사용자 인터페이스가 필요 없는 소형기기에 적합한 프로파일이다.

Ⅲ. CVM의 네트워크 설계 및 구현

1. 설계 및 구현 시 고려사항

CVM의 네트워크를 지원하는 FP를 구현하기 위해서는 CVM에서 정의하는 CDC가 우선적으로 구현되어 있어야 한다. CDC는 J2SE(Java 2 Standard Edition)의 코어 라이브러리 및 클래스 로딩 기능을 포함하며, 중소형 디바이스에서 사용되기 위해 J2SE에 필요한 코어 클래스 라이브러리의 인터페이스를 디바이스에 맞게 수정하고, 불필요한 라이브러리 클래스는 삭제되었다. CDC는 클래스 로더, 클래스 런타임 데이터 영역, 실행 엔진으로 세분화 된다. 이러한 기능을 수행하는 SubSystem이 실시간 운영체제 *i*RTOS와 연결되어 태스크 및 메모리 관리를 지원하는 CDC Core를 구성하게 되며 이는 이미 설계 및 구현되어 있음을 명시한다.

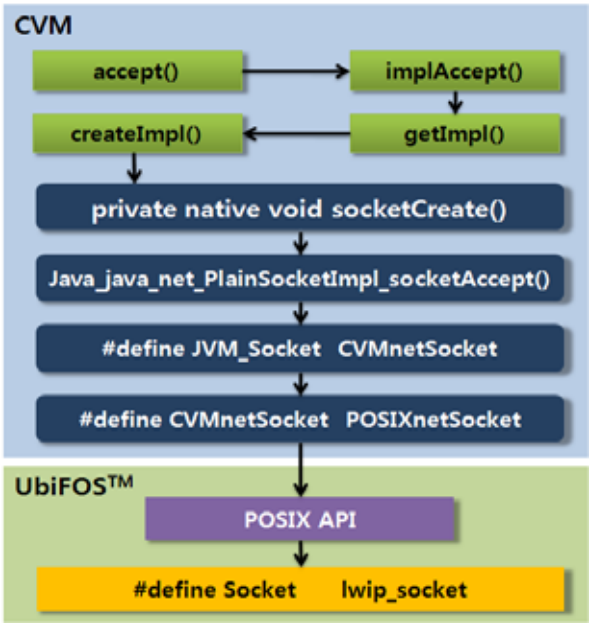


▶▶ 그림 3. J2ME의 구조

2. FP의 네이티브 메소드를 이용한 TCP 소켓 설계 및 구현

본 논문에서는 자바 응용프로그램이 *i*RTOS의 LWIP를 통

해 TCP 통신에 필요한 FP의 네이티브 메소드를 구현하였다. Java Class Method는 TCP 소켓 통신을 위해 소켓 생성을 위한 Java Native Method를 호출한다. Native Method는 실제 구현된 Native Function과 연결시켜주는 인터페이스 역할을 한다. 이 때 JNI(Java Native Interface)를 사용해서 연결하며, 파일명 뒤에 _md 가 붙은 c언어로 구현된 파일에서 해당 함수를 호출하게 된다. 이는 Java_vm_include_jvm2cvm.h 에 정의된 CVM함수로 맵핑이 되어 해당 POSIX API함수를 호출하게 되고 이와 맵핑되는 실제 *i*RTOS 내부의 관련 함수인 LWIP함수들을 수행하게 된다.



▶▶ 그림 4. CVM의 TCP 소켓 동작 메커니즘

그림 4. 는 CVM의 TCP 소켓을 생성하기 위한 과정을 나타낸 것이다. 우선 자바 응용프로그램이 수행되면 ServerSocket 클래스의 accept() 함수가 호출되고, 소켓을 생성하여 해당 소켓의 PlainSocketImpl클래스로 소켓 구현 객체를 생성해 PlainSocketImpl클래스 내부의 메소드인 accept()를 호출한다. 이 때 JNI에 의해 소켓 연결을 위한 네이티브 메소드인 socketAccept()함수가 불려지는데, accept() 뿐만 아니라 기타 다른 네이티브 메소드도 PlainSocketImpl_md.c파일에 구현해 놓았다. 구현된 네이티브 메소드 내부에서는 플랫폼 독립적인 특성을 위해 사용되는 자바 가상 머신을 통해 POSIX API함수를 호출함으로써 운영체제와 응용 프로그램의 인터페이스 역할을 하게 된다.

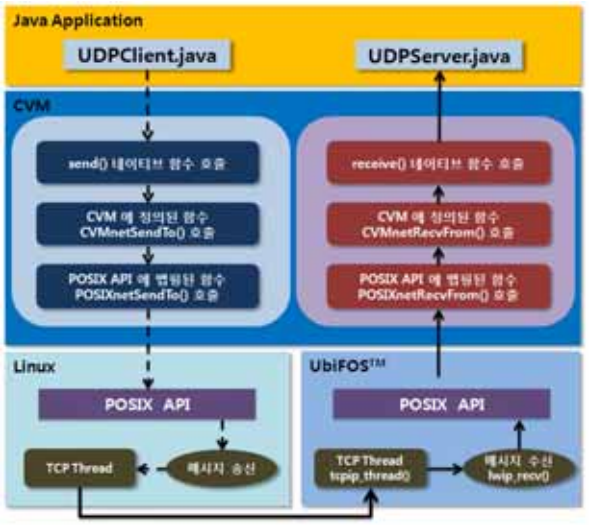
다음의 표 1. 은 TCP 통신을 위해 구현된 네이티브 함수에 대한 설명이다. 함수명 앞에 공통적으로 들어가게 되는 'Java_java_net_PlainSocketImpl_' 은 생략하였다.

[표 1] TCP를 위해 구현된 네이티브 함수

함수	설명
initProto()	filedID를 캐시에 저장
socketCreate()	소켓 생성
socketConnect()	서버와의 연결 설정
socketBind()	소켓 어드레스와 포트 바인딩
socketListen()	소켓의 연결 요구를 큐에 저장
socketAccept()	소켓 큐에서 다음 연결을 꺼냄
socketAvailable()	소켓의 유효성 검사
socketClose0()	소켓 종료
socketShutdown()	소켓 종료
socketSetOption()	인자로 받은 소켓 옵션 설정
socketGetOption()	해당 소켓의 옵션 반환
socketSendUrgentData()	데이터 전송

3. FP의 네이티브 메소드를 이용한 UDP 소켓 설계 및 구현

UDP는 프로토콜 특성상 신뢰성의 보장 없이 데이터를 보내기만 하면 되기 때문에 소켓은 수신포트만 알고 있으면 된다. UDP의 소켓은 TCP의 바이트-스트림 서비스와 달리 메시지 경계를 유지하기 때문에 한 번의 수신만을 하며, 서버와 통신하기 전에 connect()함수를 호출할 필요가 없다. 위와 같은 차이를 제외하면 UDP의 네이티브 메소드 구현은 TCP의 구현과 유사하다.



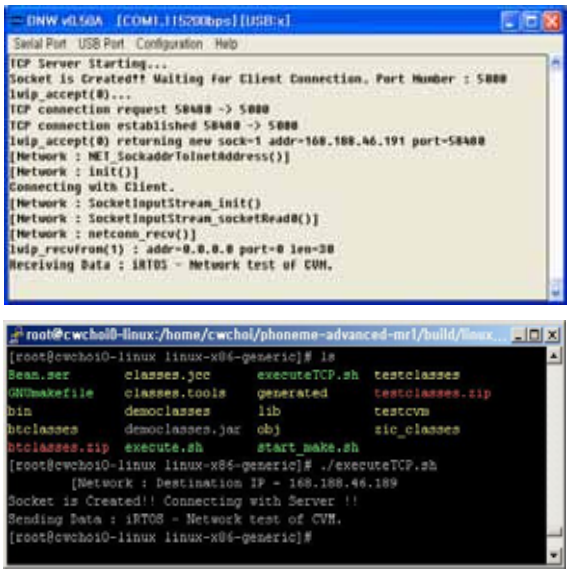
▶▶ 그림 5. CVM의 UDP 소켓 동작 메커니즘

위의 그림 5. 에서 UDP 소켓을 통해 데이터의 송수신이 이루어지는 동작을 나타내었다. UDP 서버는 *i*RTOS상에서, UDP 클라이언트는 리눅스 상에서 동작하도록 설정한 것이다. UDP 서버는 포트번호와 메시지를 저장할 버퍼의 길이를 가지고 소켓을 생성해 데이터를 받을 준비를 한다. 이 때 생성된 소켓의 receive()메소드는 Java Native Method를 호출한다. Native Method는 실제 구현된 네이티브 함수를 찾기 위해 JNI(Java Native Interface)를 사용하며 TCP와 유사한 동작

방식으로 LWIP의 sys_arch_mbox_fetch()함수를 통해 데이터 수신을 기다리게 된다. 한편, UDP 클라이언트는 서버의 IP와 포트번호, 전송할 데이터를 인자로 소켓을 생성해 send()메소드를 호출한다. 호출된 send()메소드는 서버의 receive()함수와 같이 네이티브 함수를 구현해 POSIX함수에 접근, 메시지 송신 관련 리눅스 함수를 호출해 메시지를 서버에 전송한다. 앞서 말한 바와 같이 UDP는 메시지 전송에 대한 역할만 있을 뿐, 신뢰성을 기대하지 않는 프로토콜이므로 메시지를 전송한 후에 close()함수로 소켓을 종료하게 된다.

Ⅳ. 테스트 환경 및 결과

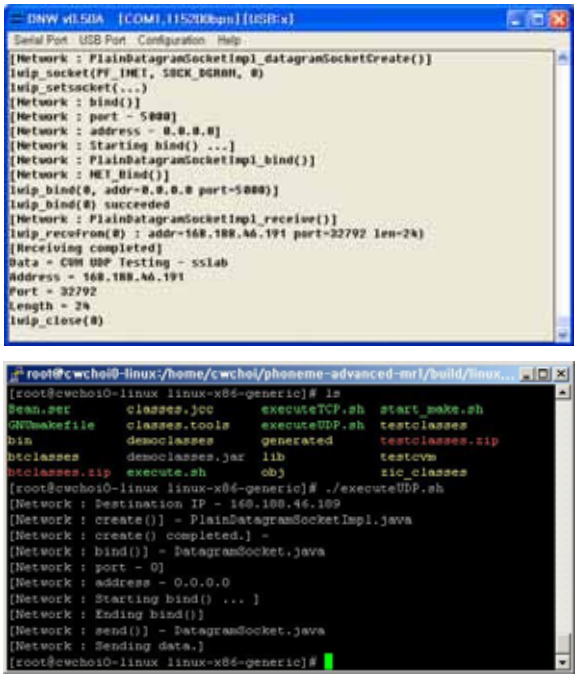
본 논문에서는 ARM920T기반의 S3C2440 32-bit RISC Micro Processor를 사용한 MBA2440보드 상에 ADS v1.2 (ARM Developer Suite ver 1.2)를 개발도구로, OPENice-A1000을 디버거로 사용해 실시간 운영체제 *i*RTOS상에 CVM 네트워크를 설계 및 구현하여 테스트하였다. 테스트는 MBA2440보드를 서버로, 리눅스를 클라이언트로 하여 메시지 전송을 통해 TCP/UDP 통신이 원활하게 이루어짐을 확인하는 방법으로 수행되었다.



▶▶ 그림 6. iRTOS 서버와 리눅스 클라이언트 간의 TCP 송수신

그림 6. 은 TCP를 사용한 *i*RTOS 서버와 리눅스 클라이언트 간의 메시지 송수신을 테스트한 것이다. 이는 MBA2440에서의 *i*RTOS 서버가 소켓을 생성하여 클라이언트를 기다리고, 리눅스 클라이언트가 메시지를 송신하면 해당 메시지를 받은 서버가 메시지를 출력하는 동작방식을 가진다. 그림에서 보는 바와 같이 테스트 결과 TCP 송수신에서 문제가 없음을 알 수 있다.

위의 TCP 테스트와 동일한 조건으로 *i*RTOS 서버와 리눅스 클라이언트 간의 메시지 송수신을 UDP를 사용하여 테스트한 것이 그림 7. 이다. 이는 클라이언트로부터 메시지를 기다리는 서버가 수신된 메시지를 출력한 후에 소켓을 종료하는 모습으로 올바른 동작을 통해 원하는 결과를 얻을 수 있었다.



▶▶ 그림 7. iRTOS 서버와 리눅스 클라이언트 간의 UDP 송수신

Ⅴ. 결론 및 향후 연구 과제

본 논문에서는 제한된 리소스를 사용하는 임베디드 시스템에 CVM을 통한 네트워크 통신이 가능하도록 *i*RTOS의 TCP/IP 프로토콜 스택인 LWIP와 CVM의 네트워크 API 프로파일인 FP사이의 상호 연동을 위한 네이티브 함수를 설계 및 구현하였다.

향후 연구과제로는 TCP/UDP 통신의 유용성을 높이기 위해 telnet, FTP, SMTP 등의 프로토콜을 구현해야 하며, 통신의 보안 기능을 위한 JSSE(Java Secure Socket Extension) 및 암호화를 위한 JCE(Java Cryptography Extension)를 구현함으로써 데이터의 신뢰도를 높여야 할 것이다. 여기에 애플릿, 고급 GUI 기능을 지원하는 PP(Personal Profile)을 구현함으로써 실시간 운영체제 UbiFOS™ 에서 다양한 자바 기능을 지원하도록 한다.

▌참 고 문 헌▐

[1] 최찬우, 이철훈, “실시간 운영체제 UbiFOS™ 에서의 CVM 설계 및 구현”, 한국콘텐츠학회, 2007 추계 종합학술대회 논문집 제5권 제2호(하), 2007.11, pp. 812-816
[2] 이정원, 전상호, 이승열, 이철훈, “실시간 운영체제를 위한 경량 네트워크 스택의 설계 및 구현”, 한국정보과학회, 2006 한국컴퓨터종합학술대회 논문집 Vol. 33, No .1(A), pp 373-375
[3] Sun Microsystems, "CDC (Connected Device Configuration) Runtime guide".
[4] Sun Microsystems, "CDC : JAVATM Platform Technology For Connected Devices".
[5] Sun Microsystems, "JSR-000046 Foundation Profile 1.0b Maintenance Release"