2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

포터블 프로그래밍 도구를 활용한 멀티플랫폼 S/W 개발

최진우**, 박상서*, 이진석*, 박성우**,이정국**, 황선태**, 우종우**
*한국전자통신연구원 부설 국가보안기술연구소
**국민대학교 컴퓨터 학부
e-mail: jwchoi@kookmin.ac.kr

Multiplatform Software Development Guide using Portable Programming Tool

Jinwoo Choi**, Sangseo Park*, Jinseok Lee*,
Sungwoo Park**,Jeongguk Lee**, Suntae Hwang**, Chongwoo Woo**
*National Security Research Institute
**School of Computer Science, Kookmin University

요 약

최근 컴퓨터 및 컴퓨터 통신의 발달은 서로 다른 플랫폼에 접근할 수 있는 기회를 증가시키고 있으며, 이와 더불어 새로운 하드웨어 및 운영 체제의 지속적인 출현은 멀티플랫폼 소프트웨어에 대한 요구를 증가 시켜왔다. 따라서 프로그래머는 새로운 소프트웨어를 개발하거나 기존의 소프트웨어를 포팅할 때, 멀티플랫폼에서의 실행을 염두에 두어야 하는데 이를 위한 도구의 설계 및 프로토타입 구현[11]에 대해서 이미 소개한 바 있다. 본 논문에서는 기존의 GNU 시스템 도구들과 연계되어 개발된 이 포터블 프로그래밍 도구를 활용하여 멀티플랫폼 소프트웨어를 제작하는 단계를 서술하고 이 때 우리의 도구가 어떤 역할을 하는지에 대해서 논한다.

1. 서론

최근의 컴퓨터는 그 발전 속도가 매우 빨라 소프트웨어 개발자의 부담을 가중시키고 있다. 특히 컴퓨터 통신의 발달로 인해서 이 기종의 시스템에 접할 수 있는 기회가 많아 졌고, 게다가 새로운 하드웨어 및 운영체제도 꾸준히 추가[1,2]되고 있는 실정이어서 동일한 기능의 소프트웨어를 플랫폼에 상관없이 실행하고자 하는 요구가 지속적으로 늘고 있는 추세이다.

대부분의 프로그래머는 다양한 플랫폼에 대한 지식이 부족할 수 있는데 이는 여러 가지 도구를 활용함으로써 극복할 수 있고 이런 도구들을 유기적으로 연계해서 활용할 수 있는 도구[4,5]도 소개된 바 있다. 본 논문에서는 이 도구를 활용하면서 멀티플랫폼 소프트웨어를 개발하는 과정에 대해서 논한다.

본 연구는 ETRI 부설 국가보안기술연구소 2000년도 정보 보호 기술개발 위탁연구과제에서 지원 받았음

2. 멀티플랫폼 소프트웨어 개발

멀티 플랫폼 소프트웨어 개발 시 또는 기존의 소프트웨어를 멀티 플랫폼으로 포팅 시 모두에게 요구되는 공통 사항으로는 목적 플랫폼과의 이질성을 완전하게 이해해야 한다는 점이다. 그러나 동일한 플랫폼이라도 해당 버전에 의존적인 라이브러리가 필요함으로 인해 완전한 이해는 불가능하며 그 결과 개발자의 부담이 증가된다. 이러한 제약에서 공통적으로 잠재되어 있는 문제점은 다음과 같다.

첫째, 하나 이상의 특정 목적 플랫폼을 겨냥한 소스 코드 내에는 조건부 코드(conditional code)들이 존재해야 하며, 조건부 코드들로 둘러싸인 부분에 희망하는 코드를 삽입할 지식이 개발자들에게 요구된다. 동시에 개발자에게는 완성된 소프트웨어 패키지를 목적 플랫폼에서 구축 시 Makefile 에 관련된 총체적인 지식이 요구된다.

둘째, 포함해야 하는 시스템 파일들(#include)의 위치 정보가 문제가 된다. 목적 플랫폼에 따라 이미 설치되어 있는 시스템 파일들의 위치가 다를 수 있거나 혹은 존재하지 않을 가능성이 있으므로 개발자에게 이러한 위치 정보에 대한 지식이 요구된다.

셋째, 개발자에 의해 의도적으로 사용되는 라이브러리 함수들에 대한 문제가 발생한다. 플랫폼에 따라 제공되는 라이브러리에 따라 함수의 정의가 존재하지 않을 가능성이 있다. 이때 개발자에게 모든 플랫폼에 의존적인 라이브러리 함수의 정의에 대한 정확한 지식이 요구된다.

넷째, 플랫폼에 따라 제공되는 라이브러리 내에 함수들이 정의가 명시되어 있더라도 그 의미(semantics)가 다를 시 문제가 야기된다. 예를 들어 SVR4 와 BSD 의 경우에서 함수 signal 의 사용 시, SVR4 상의 signal 은 신뢰할 수 없는 신호들만을 제공하는 전형적인 UNIX V7 신호들과 동일한 반면, BSD 상에서의 signal 은 신뢰할 수 있는 신호로써 시스템 호출의 재시작을 제공한다[3]. 이렇게 플랫폼에 따라 동일한 함수라도 그 의미가 서로 상이한 모든 경우들의 지식이 개발자에게 요구된다.

다섯째, 각 플랫폼에 따른 바이트 정렬이 문제가 된다. 예를 들어 SPARC 시스템 구조는 빅 엔디안(Big Endian) 이고 Alpha 와 Intel x86 시스템 구조는 리틀 엔디안(Little Endian)이다. 따라서 개발자에게 코드 작성 시 이에 대한 지식이 요구된다.

여섯째, 다양한 플랫폼에서의 소프트웨어 패키지 구축 시, 개발자는 각 플랫폼에서 요구되는 환경 설정을 고려해야 한다. 예를 들어 각 플랫폼에서의 사용 가능한 컴파일러의 선택과 제공되는 라이브러리의 사용 시 필요한 옵션들, 또는 인스톨 시의 경로 설정들과 같은 모든 환경 정보에 대한 지식이 개발자에게 요구된다.

기타, 대부분의 경우 ANSI C 에서는 문제를 삼지 않지만, 그렇지 않은 경우에서는 워드(word)의 길이, 타입들의 범위에 관한 최소값 또는 최대값 등이 문제를 발생할 경우도 존재한다.

3. 포터블 프로그래밍 도구

위와 같은 문제점을 극복하고 성공적인 멀티플랫폼 소프트웨어의 개발을 위해선 우선 개발자들은 이식성(portability)을 고려하여 신중하게 작성하여야 하며, 이러한 개발자에게 도움이 되는 지침서[5,6]를 제공하거나 자동화를 지원하는 시스템이 절실히 필요로 하다.

이식 가능한 프로그램의 개발을 위한 대표적인 시스템 도구들로는 GNU 의 autoconf[7], autoscan[7], autoheader[8], automake[9], libtool[10] 등을 들 수 있다. 이러한 도구들은 개발자 측에서만 필요하며 최종 생성된 소스 패키지의 설치 시에는 요구되지 않는다. Autoconf 는 다양한 유닉스 계열의 시스템에 적응시키기 위해 소프트웨어 소스 코드를 자동으로 설정해 주는 쉘 스크립트를 생성하는 도구이다. autoconf 의 입력 파일은 소스 패키지 내의 프로그램들의 검증, 컴파일 시 링크되어질 라이브러리들의 검증, 헤더 파일들의 검증, 타입의 정의 및 구조체와 컴파일 환경의 검증, 라이브러리 함수들의 존재 여부 검증 등을 담당하는 부분으로 구성되어 있다. 각 소프트웨어 패키지에 대하여 autoconf 는 패키지에서 필요로 하거나 사용하는 시스템 기능을 명시한 템플릿(template)인 configure.in 을 입력 파일로 하여 설정 스크립트 configure 를 생성한다. libtool 은 공유 라이브러리 구현을 위한 표준화된 접근 방안을 제시하며, autoscan 은 특정 디렉토리 또는 현재 디렉토리를 기반으로 소스 파일을 점검하게 된다. 소스 파일을 검색하여 일반적으로 이식 시에 문제를 야기할 부분을 찾아 configure.in 의 골격 파일(skeleton file)인 configure.scan 을 생성한다. autoheader 는 configure.in 내의 검증 태그를 #undef 으로 초기화해놓은 config.h.in 을 생성한다. 이러한 시스템 도구들은 멀티플랫폼 소프트웨어 개발 시 소요 시간을 단축시키고 어느 정도 자동화 기능을 제공하고 있지만 다음과 같은 문제점을 가지고 있다.

첫째, 이러한 시스템 도구들은 그 기능과 많은 옵션이 지나치게 세분화 되어있으므로 이러한 지식의 습득 또한 개발자에게는 큰 장애가 될 수 있다.

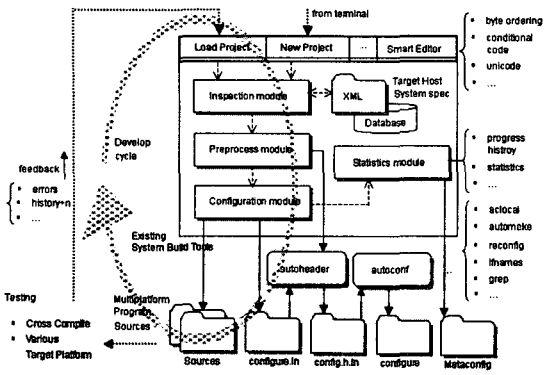
둘째, 소프트웨어 패키지의 구축 시 시스템 도구들은 일련의 절차를 따라 사용하여야 하는데, 이러한 진행 절차를 제대로 파악하지 못한 미숙한 개발자에게는 그 사용이 어렵다.

셋째, 시스템 도구들은 소스의 작성 단계가 아닌 완전한 소스가 생산된 이후 소프트웨어 구축 시점인 최종 단계에 해당하므로 소프트웨어 개발 진행 과정에 있어서 그 구실을 할 수 없다.

이러한 문제점의 해결 방안으로 기존의 GNU 시스템 도구들과 연동함과 동시에 이미 언급했던 포팅 시의 문제점을 해결하기 위한 멀티플랫폼 소프트웨어 제작에 필요한 지식을 개발 진행 중에 개발자에게 대화식으로 제공함으로써 일반적인 개발 작업들을 도울 수 있는 기능들을 포함한 멀티플랫폼 소프트웨어 개발 도구가 요구된다.

4. 종합 개발 도구의 설계

개발자가 오류의 개연성이 존재하는 코드의 작성 시 시스템이 이를 인식하여 지침서를 제공할 수 있으며 앞에서 언급한 포터블 프로그래밍 도구들을 집약한 자동화 시스템이 필요로 하다. 이러한 요구를 만족하기 위해 멀티플랫폼 소프트웨어 개발 도구는 다음 그림과 같이 설계하였다[그림 1]



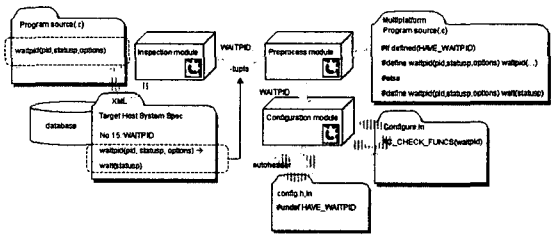
[그림 1] 시스템 설계도

[그림 1]과 같이 시스템은 검증 모듈(Inspection module). 전처리 모듈(Preprocessor module)과 설정 모듈(Configuration module)인 3 개의 핵심 모듈과 진행 중인 프로젝트의 유지를 위한 통계 모듈(Statistics module)로 설계되었다. 그리고 이들은 기존 시스템 구축 도구들(autoconf, autoheader, automake 등)과 연동을 통해 이식 가능한 소프트웨어를 생성해 낸다[11].

제안된 멀티플랫폼 소프트웨어 개발 도구는 개발자에게 크게, 스마트 에디터, 개발 에디터, 시스템 자동화 빌더, 프로젝트 관리의 기능들로써 4 가지 기능들을 제공한다.

우선 스마트 에디터의 기능은 개발자에게 유틸리티 메뉴를 제공하도록 설계하여 개발의 효율성을 강조한 기능이다. 예를 들어 다양한 플랫폼에서의 실행을 목적으로 개발된 소스 코드는 많은 조건부 코드로 인하여 이를 직접 검색하기에는 개발자에게 상당한 어려움이 존재한다. 또한 중첩된 조건부 코드는 개발의 해독을 어렵게 한다. 이에 시스템은 시스템 보완 도구들(System supplement tools)을 이용하여 조건부 코드들만을 이들의 위치만을 정확히 검색하여 개발자에게 제시할 수 있다(예를 들어 ifnames, grep 등). 또한 시스템이 인식하기에 곤란한 개발자 오류(human mistake)의 범주에 속하는 바이트 정렬, 조건부 코드, 유니코드 등에 관하여서도 개발자의 주의를 요할 수 있다. 즉 바이트 정렬의 있어서 개발자에게 주의를 기울일 수 있도록 시스템은 문제의 여지가 있는 사용된 구조체(structure) 혹은 공용체(union)를 다른 형태(반전, 컬러링)로 제시하며, 이에 따른 지침서를 제공할 수 있다.

개발 에디터의 기능은 시스템 초기화 시 목적지 플랫폼 환경에 요구되는 정보들로 설계된 XML 명세서를 개발 도구 안으로 로드함으로써 개발자에게 지침서를 제공하게 된다. 예를 들어 문제의 여지가 있는 특정 라이브러리 콜과 시스템 콜의 사용 시 또는 사용되는 함수의 의미가 각각의 플랫폼에서 상이한 경우 편집 중인 소스를 재구성하도록 개발자에게 지침서를 제공하거나 또는 개발도구 안의 에디터 스스로가 소스를 재구성하게 된다. 이러한 일련의 수행 과정은 에디터의 핵심 모듈인 검증 모듈, 전처리 모듈과 설정 모듈의 상호 작용을 통해 수행하게 된다[그림 2].



[그림 2] 핵심 모듈과 시스템 도구들과의 연동

시스템 자동화 빌더 기능은 개발의 최종 단계에서 소프트웨어 패키지의 컴파일 환경 설정에 관여하는 기능이다. 최종 생산된 패키지는 개발이 이루어진 플랫폼과 실제 실행이 이루어질 목적지 플랫폼과는 구별된다. 만일 해당 플랫폼에서의 실행은 성공적임에 반하여 개발 플랫폼에서 그렇지 않다면 멀티플랫폼 소프트웨어에 위배된다. 그러므로 개발 플랫폼에서 실행의 성공이 우선되어야 한다. 그 이유는 목적지 플랫폼에서의 오류 발생 시는 소스의 로직이 아닌 플랫폼에 의존적인 부분에 문제점이 존재함을 입증하는 것이기 때문이다. 실제로 이러한 예는 각 플랫폼에 따라 포함되어야 할 시스템 파일들의 경로가 구별되어야 할 경우를 들 수 있으며 이때 시스템은 시스템 구축 도구를 사용하여 이를 검증한 뒤 소프트웨어 패키지의 재구축 시 요구되는 스크립트 파일의 작성 시 재구성의 자동화를 제공한다.

나머지 기능으로써 프로젝트 관리의 기능은 다음과 같다. 기본적으로 새로운 프로젝트 아래서 새로운 멀티플랫폼 코드를 작성하는 기능이 있고 기존의 소스 파일들을 새로운 프로젝트로 묶어서 입력하여 멀티플랫폼 코드로 전환하는 기능을 제공한다. 프로젝트에 포함된 파일들의 경우 시스템은 하나의 파일 수정으로 인한 변경이 해당 파일에만 국한되지 않고 포함된 파일들 모두에게 변경을 반영 가능하며, 기존 소프트웨어의 포팅 시 시스템은 진행에 관련하여 통계 수치를 제공한다. 즉 발견된 문제와 이를 수정한 개수, 미수정한 개수 등과 관련한 모든 진행에 관계된 통계치를 위한 'Metaconfig'를 제공하여, 이들의 위치로 개발자를 직접 이동시킨다. 마지막으로 진행 중인 프로젝트의 유지를 위하여 별도의 프로젝트 디렉토리를 구분하여 관리를 용이하게 하였다.

최종 완성된 소프트웨어 패키지를 다양한 플랫폼에서 테스트 가능하도록 하였으며(cross compile), 만일 성공적이지 못하다면 'Metaconfig'에 기록된 피드백 정보(error, history 등)를 가지고 개발 싸이클(Development cycle)을 순환하여 수정할 수 있도록 설계하였다.

5. 결론

본 논문에서는 멀티플랫폼 소프트웨어를 개발하는 단계에서 고려해야 할 사항들을 논하고 이를 반영할 수 있는 개발 도구의 요건을 서술하였다. 이 개발 도구는 기존의 GNU 시스템 도구와 연동하면서 소프트웨어 개발자에게 멀티플랫폼 소프트웨어 제작에 필요한 지식을 제공하고 개발 중인 소프트웨어에 대화식으로 적용할 수 있으며 완전한 해법을 줄 수 없는 사항들은 일반적으로 개발자가 개발 시 행하는 일들을 도울 수 있는 기능들을 첨가하였다. 필요한 지식은 전문가로부터 얻거나 잘 알려진 멀티 플랫폼 소스 코드에서 추출하여 반영될 수 있도록 설계되었다. 현재 구현에는 Python 이 이용되고 있으며 개발 도구의 핵심 기능인 검증 모듈. 전처리 모듈과 설정 모듈의 프로토타입이 구현되어 있으며 Smart Editor 와 통계 모듈 등이 구현 중에 있다.

참고문헌

[1] LDP, "Linux", http://www.linuxdoc.org
[2] COMPAQ, "Tru64TM UNIX on Compaq Documentation Overview", April 2000, http://tru64unix.compaq.com/faqs/publications/base_doc/DOCUMENTATION/V50A_HTML
[3] M. Welsh, "Porting Applications to Linux*", January http://www.cs.berkeley.edu/~mdw/linux/porting-linux/html/paper.html
[4] E. Zadok, "Autoconfiscating Amd: Automatic Software Configuration of the Berkeley Automounter", http://www.cs.columbia.edu/~ezk/research/amu/autoconfiscating_amd.html
[5] COMPAQ, "Sun Solaris to Compaq Tru64 UNIX Porting Guide", Oct 1999, http://www.tru64unix.compaq.com/faqs/publications/porting/HTML/solaris.html
[6] A.Dolec, A. Lemmke, D. Kepple, "Notes On Writing Portable Programs In C", June 1990, http://www.cs.umd.edu/users/cml/cstyle/portableC.pdf
[7] D. MacKenzie, D. Elliston "Autoconf", http://www.gnu.org/manual/autoconf/html_mono/autoconf.html
[8] D. Mackenzie, T. Tromey, "GNU Automake", http://www.gnu.org/manual/automake/html_mono/automake.html
[9] G.Matzigkeit, "GNU Libtool", http://www.laria.u-picardie.fr/docs/gnu/libtool/libtool_toc.html
[10] M. S. Richard, Mc. Roland "GNU make", http://www.gnu.org/manual/make-3.79.1/html_mono/make.html.gz
[11] 최진우, 윤영태, 이진석, 이정국, 우종우, 황선태, 2000, "멀티플랫폼 소프트웨어 제작을 위한 개발 도구의 설계", 인터넷 정보학회 추계 학술발표논문집, pp 314-318