멀티 플랫폼 소프트웨어 개발을 위한 대화형 도구의 설계 및 구현

Design and Development of an Interactive Tool for Developing Multi-platform Software

최 진 우* 황 선 태** 우 종 우***
Jin-Woo Choi Sun-Tae Hwang Chong-Woo Woo

요 약

새로운 하드웨어 및 운영 체제의 지속적인 출현으로 인해 멀티 플랫폼에서 이식이 가능한 포터블 프로그래밍에 대한 중요성이 더욱 부각되고 있다. 또한 최근 컴퓨터 및 컴퓨터 통신의 발달로 인해 서로 다른 플랫폼으로의 접근 기회가 증가함에 따라 소프트웨어 개발자는 초기 제작 단계부터 여러 가지 플랫폼을 염두에 두고 개발해야 하는 등 소프트웨어 개발 부담이 가중되고 있다. 기존의 GNU 시스템 도구 등에서는 소스 코드의 일부 configuration을 자동으로 생성하게 하고 새로운 플랫폼을 향한 지침서 등을 제시하고 있지만 소프트웨어에 이식성을 부여하기 위한 실질적인 코드는 소스 레벨에서 개발자에 의해 직접 입력되어야 하는 한계가 있다. 본 논문에서는 멀티 플랫폼 소프트웨어 개발을 위한 대화형 의 설계와 구현에 관하여 기술한다. 이 도구는 다양한 유닉스 계열 시스템에 관한 숙련된 지식이 없는 개발자들도 손쉽게 멀티 플랫폼 소프트웨어를 개발할 수 있도록 도울 수 있게 설계되었다. 또한 이 도구는 기존 GNU의 autoconf, autoheader, automake와 같은 시스템 툴을 사용하며, 개발자와 상호작용에 따라 적합한 장소에 포터블 코드를 삽입할 수 있다.

Abstract

A continuous development of new hardware and operating systems brings the importance of portable programming, which can be imported into multi-platform environment. Also, the recent development of computer communication technology brings us many choices of accessing into the other platforms, which also becomes a burden for the programmers who should consider developing software fitting into diverse platforms. The present GNU system tools creates a part of the source code automatically, and suggests a direction towards a new platform, but the source code must be prepared by the programmer at the source level. In this paper, we describe design and implementation of an interactive tool for developing multi-platform software. The tool is designed to help a programmer to build a portable program for multi-platform, even though the programmer does not know very well about various UNIX platforms. And, the tool uses existing GNU system tools, such as autoconf, autoheader, automake and etc, and is able to insert a portable code into a right position by interacting with a programmer.

1. 서 론

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.

이런 요구는 여러 가지 방법으로 해결될 수 있는데 실행 파일만을 가지고 있을 때 해당 플랫폼을 흉내내주는 가상 레이어의 에뮬레이션 기법[3,4]에서부터 소스를 가지고 있다는 가정 하에 포팅을 하는데 그 절차를 어느 정도 자동화하고 도와

* 정회원 : 국민대학교 전산과학과 박사과정
jwchoi@cs.kookmin..ac.kr
** 비회원 : 국민대학교 컴퓨터학부 교수
sthwang@kookmin..ac.kr
*** 정회원 : 국민대학교 컴퓨터학부 교수
cwwoo@kookmin.ac.kr

주는 도구[5,6]의 사용까지 다양하다. 또한 소프트웨어의 제작 단계에서부터 다양한 플랫폼에서의 실행을 가정할 수 있는데 아예 멀티 플랫폼을 지원하는 라이브러리[7,8]를 사용하거나 아니면 해당 플랫폼에 대한 지식을 충분히 가지고 제작해야 한다. 이 때 여러 가지 지침서[9,10]를 참고 할 수도 있다. 하지만 소스 코드에 이식성을 부여하기 위해서 최종 작업하는 단계에서 개발자를 돕는 개념은 포함되어 있지 않다.

본 논문에서는 기존 시스템 도구 및 지침서 등을 총괄하여 멀티 플랫폼에 대한 별반 지식이 없는 개발자라도 이식성이 높은 소프트웨어의 제작이 가능하도록 돕는 도구의 개발 및 이를 활용하기 위한 멀티 플랫폼 소프트웨어 개발에 대해서 논한다. 다음 2절에서는 관련된 기존 기술과 도구들에 대하여 언급하고, 3절에서는 개발 또는 포팅 시 잠재적인 문제점을 제시한다. 4절에서 위에서 전술한 문제점 해결을 위한 종합 개발 도구의 각 모듈별 설계, 5절에서 구현, 그리고 마지막으로 6절에서 결론 및 향후 연구에 대하여 기술한다.

2. 관련 연구

2.1 GNU 시스템 도구

이식 가능한 프로그램의 개발을 위한 대표적인 시스템 도구들로는 GNU의 autoconf, autoscan, autoheader [11], automake[12], libtool[13] 등을 들 수 있다. Autoconf의 입력파일은 소스 패키지 내의 프로그램들의 검증, 컴파일 시 링크되어질 라이브러리들의 검증, 헤더파일들의 검증, 타입의 정의 및 구조체와 컴파일 환경의 검증, 라이브러리 함수들의 존재 여부 검증 등을 담당하는 부분으로 구성되어 있다. Automake는 이식 가능한 프로그램 생성에 필요한 어떠한 정보도 가지고 있지 않지만 소스 패키지의 설치에 대한 보다 상세한 정보를 가지고 있는 입력파일로 사용함으로써 보다 용이하게 Makefile[14]을 생성할 수 있다. 이들 자동화 도구들의 사용에 앞서, 우선 개발자에 의해 작성된 소스 파일을 비롯하여 기타 환경에 관여하는 파일들을 작성해야만 한다.

2.2 유닉스 윈도우 간 소프트웨어 이동

기존 윈도우 코드들을 가능한 수정하지 않고 작업하려면 윈도우의 API를 구현 할 수 있는 라이브러리가 필요하며, 이러한 대표적인 라이브러리로 Wine을 들 수 있다[3]. Wine 사용에 관한 대표적인 용례들은 다음과 같다.

◎ 윈도우의 실행 파일을 리눅스 상에서 다시 컴파일 하지 않고 바로 실행하도록 하는 프로그램 로더(loader), 즉 에뮬레이터(emulator)를 이용하는 방법
◎ 윈도우 프로그램 소스 코드를 리눅스에서 컴파일 하는 데에 사용되는 라이브러리(library)를 이용하는 방법이다.

첫째 용례는 리눅스에서 윈도우의 실행파일을 바로 실행 가능하지만 리눅스 시스템의 특성들을 충분히 활용하기는 어려운 문제점이 존재한다. 둘째 용례는 윈도우 라이브러리의 API의 기능들을 구현할 수 있기 때문에 리눅스 환경에 적합한 프로그램 작성이 가능하다. 특히 Wine 라이브러리가 X윈도우의 Xlib 라이브러리처럼 공개되어 있기 때문에 특정한 응용프로그램을 리눅스에서 컴파일 할 때 필요에 따라 개발자가 Wine 라이브러리를 수정할 수 있다는 장점이 있다. 그러나 이들 도구의 이용은 멀티 플랫폼 소프트웨어 개발 시 소요시간을 단축시키고 다소 자동화 기능을 제공하지만 다음과 같은 문제점을 가지고 있다.

◎ 이러한 시스템 도구들은 그 기능과 옵션들의 지나친 세분화로 인해 이와 관련한 지식의 습득 또한 개발자에게는 큰 장애가 될 수 있다.
◎ 소프트웨어 패키지 구축 시 시스템 도구들은

일련의 절차를 따라 사용하여야 하는데, 미숙한 개발자에게는 그 사용이 어렵다.

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

◎ 실제 개발을 담당할 개발자들에게는 시스템 프로그래밍 지식이 부족하며, 개발자들 스스로 목적 플랫폼의 시스템 문서를 참조하여 개발하기 어렵다.

본 논문에서는 이러한 문제점들을 보완하고, 또한 다양한 유닉스 계열의 시스템 프로그래밍에 관한 숙련된 지식이 없는 개발자들도 손쉽게 멀티 플랫폼 소프트웨어를 개발할 수 있는 멀티 플랫폼 소프트웨어 개발 도구의 개발을 연구를 목표로 하여 시스템을 설계하였다.

3. 멀티 플랫폼 소프트웨어 개발

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

◎ 다수의 특정 목적 플랫폼을 향한 소스 코드 내부에는 조건부 코드들이 존재해야 하며, 이들로 둘러싸인 부분에 원하는 코드를 삽입할 지식이 요구된다. 동시에 완성된 소프트웨어 패키지를 목적 플랫폼에서 구축 시 Makefile에 관련된 총체적인 지식이 요구된다.

◎ 목적 플랫폼에 따라 이미 설치되어 있는 시스템 파일들(#include)의 위치가 다르거나 혹은 존재하지 않을 수 있으므로 개발자에게는 이들의 위치 정보에 대한 지식이 요구된다.

◎ 플랫폼 또는 제공되는 라이브러리에 따라 함수의 정의가 존재하지 않을 가능성이 있으므로, 모든 플랫폼에 의존적인 라이브러리 함수의 정의에 대한 정확한 지식이 요구된다.

◎ 플랫폼에 따라 동일 함수라도 그 의미가 서로 다른 모든 경우의 지식이 요구된다. 예를 들면 SVR4와 BSD의 경우 함수 signal의 사용 시, SVR4 상의 signal은 신뢰할 수 없는 신호들만을 제공하는 전형적인 UNIX V7 신호들과 동일한 반면, BSD 상에서의 signal은 신뢰할 수 있는 신호로써 시스템 호출의 재시작을 제공한다[3].

◎ 각 플랫폼에 따른 바이트 정렬 또한 고려되어야 한다. 예로써 SPARC 시스템 구조는 빅 엔디안(Big Endian)이고, Alpha와 Intel x86 시스템 구조는 리틀 엔디안(Little Endian)이므로 코드 작성 시 이러한 지식이 요구된다.

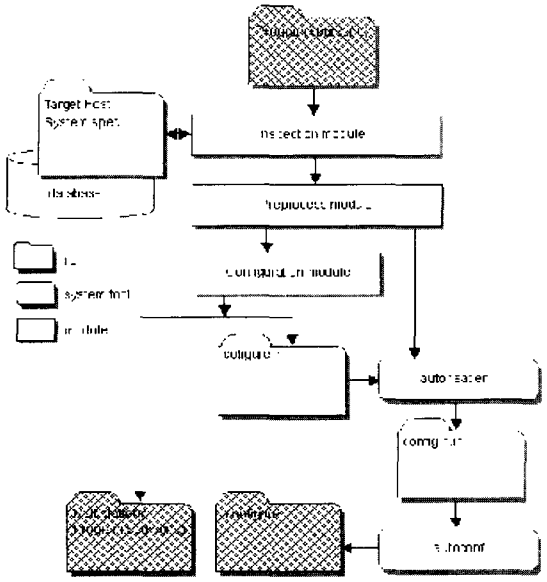
◎ 다양한 플랫폼에서의 소프트웨어 패키지 구축 시, 각 플랫폼에서의 사용 가능한 컴파일러의 선택과 제공되는 라이브러리의 사용 시 필요한 옵션들, 또는 설치(install) 시 경로 설정들과 같은 모든 환경 정보에 대한 지식이 개발자에게 요구된다.

◎ 대부분의 경우 ANSI C에서는 문제를 삼지 않지만, 그렇지 않은 경우에서는 워드(word)의 길이, 타입들의 범위에 관한 최소 값 또는 최대 값 등이 문제를 발생할 경우도 존재한다.

이러한 다양한 문제점을 고려하더라도 실제 플랫폼에서의 실질적인 수행 또는 구축 시에는 개발자에 의해 예기치 못한 많은 문제점이 발생할 수 있으므로 지속적으로 소프트웨어 개발 싸이클을 통한 완전한 제작이 필요하다.

4. 종합 개발 도구의 설계

멀티 플랫폼 소프트웨어 개발 도구는 대화식으로



(그림 1) 종합 개발 도구의 전체 흐름도

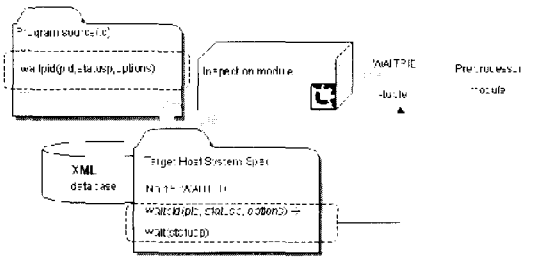
이루어지며 워드프로세서에서 맞춤법 검사 시, 시스템의 제안에 따라서 새로운 단어로 치환해 나가는 것과 유사한 개념으로 설계되었다.

그림 1과 같이 시스템은 핵심 모듈인 검증 모듈, 전 처리 모듈, 설정 모듈의 상호 작용과 GNU 시스템 스크립트들과의 연동을 통해 이식 가능한 소스를 생성한다. 최종 작업이 완성된 뒤에는 configure 실행 스크립트 파일과 Makefile, 그리고 프로젝트 정보를 가지고 있는 메타파일(metafile)을 부수적으로 생성한다[15].

4.1 검증 모듈 (Inspection Module)

검증 모듈은 개발자가 코드를 입력하는 과정에서 목적 플랫폼에서 문제를 야기할 만한 부분을 검증하는 부분으로써, 개발자가 사용 중인 특정 코드와 목적 플랫폼의 명세서를 검색하여 개발자에게 지침서를 제공하게 된다.

그림 2에서 검증 모듈은 개발자의 입력을 주시하여 검증이 요구되는 문자열(특정 함수 또는 헤더 파일 이름 등)이 입력되었을 경우 데이터베이스를 참조하여 지침서의 내용을 추출해 낸다. 그



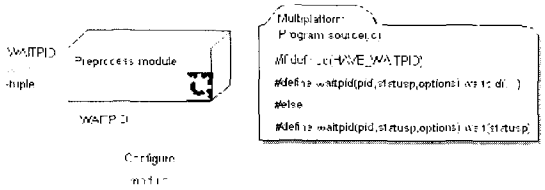
(그림 2) 검증 모듈

리고 이 추출 정보를 별도의 대화 상자를 통해 출력함으로써 개발자로 하여금 필요한 조처를 취하도록 한다. 검증 모듈을 메타 파일의 내용을 참조하여 이미 해결된 문제인지를 판단하고, 해결된 부분에 대해서는 어떠한 메시지도 출력하지 않는다. 위의 그림과 같이 waitpid에 대해서 해결이 된 상태인 경우에는 메타 파일 내에 waitpid에 관한 정보가 기록 되며, 이후 개발자의 waitpid의 입력이 있을시 또다시 지침서를 제공하지 않는다.

검증 모듈로의 입력을 준하여 참조하는 데이터베이스는 XML 구조로 문서화된다. XML 표준을 따름으로써 시스템에서 얻는 장점은 개발 테크닉을 문서화 할 수 있으며, 문서의 논리적 구조를 참조하여 이를 서로 공유 할 수 있다. 또한 동일한 문서를 개발자의 목적에 맞게 정보를 재가공 가능하며, 다양한 지식을 가지고 있는 개발자들 간 웹을 이용하여 지식을 교환하기에 편리하다.

4.2 전 처리 모듈 (Preprocess Module)

전 처리 모듈은 검증 모듈로부터 얻은 지식을 삽입하는 모듈로써 편집 중인 소스를 재구성하게 된다. 이때 전 처리 모듈의 입력은 명시된 태그와 테이블의 레코드를 하나의 쌍으로 구성한다(예: (waitpid, wait)). 이를 기준으로 전 처리 부분을 해당 헤더파일이나 소스 내용 중의 가장 적절한 부분에 기록하여 재구성하게 된다. 아래 그림 3은 waitpid 함수의 사용을 가정한 전 처리 모듈의 동작 과정으로써 검증 모듈의 수행 결과인 지침서의 제공을 나타낸다.

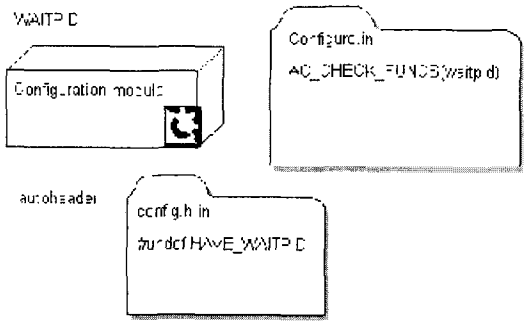


(그림 3) 전처리 모듈

4.3 설정 모듈 (Configuration module)

설정 모듈은 전 처리 모듈로부터 명시된 검증 태그들로 설정 파일(configure.in)을 구성한다. 그림 4에서 설정 파일은 autoconf의 입력 파일이 된다. 설정 모듈은 configure.in을 재구성할 뿐만 아니라 메타 파일을 재구성하게 된다. 전 처리 모듈로부터 입력되는 태그를 설정 파일 내 적절한 위치에 기록함과 동시에 메타 파일에도 이러한 변화를 반영하게 된다.

메타 파일은 프로젝트 파일 존재 시에는 설정 파일의 정보와 프로젝트 파일의 정보 모두를 반영하며, 프로젝트 파일이 존재하지 않은 경우에는 설정 파일의 정보를 그대로 반영하게 된다.

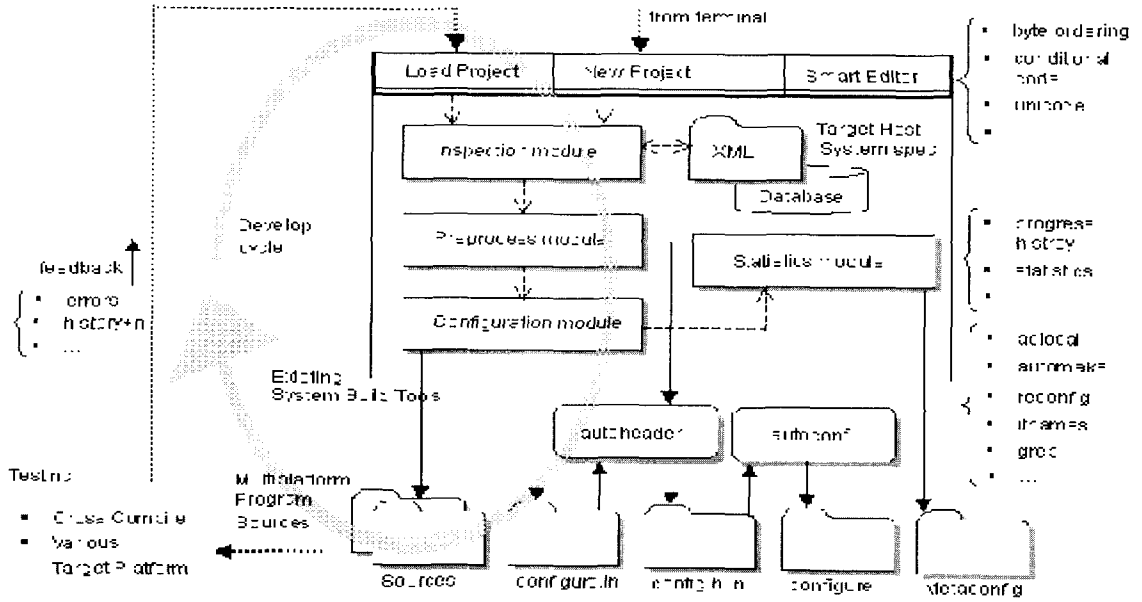


(그림 4) 설정 모듈

4.4 핵심 모듈들과 시스템 도구와의 연동

시스템은 세 개의 핵심 모듈과 진행 중인 프로젝트의 유지를 위한 통계 모듈(Statistics module)로 설계되었다(그림 5 참조). 제안된 멀티 플랫폼 소프트웨어 개발 도구는 개발자에게 크게 스마트 에디터, 개발 에디터, 시스템 자동화 빌더, 프로젝트 관리 등 네 가지 기능들을 제공한다.

첫째, 스마트 에디터(smart editor)의 기능은 개발자에게 유틸리티 메뉴를 제공하는 기능이다. 개발되는 소스 코드는 많은 조건부 코드로 인하여



(그림 5) 시스템 설계도

직접 검색 시 상당한 어려움이 존재한다. 또한 중첩된 조건부 코드는 개발의 해독을 어렵게 한다. 이에 시스템은 시스템 보완 도구들을 이용하여 조건부 코드들만의 위치만을 정확히 검색하여 개발자에게 제시할 수 있다(예를 들어 ifnames, grep 등). 또한 시스템이 인식하기에 곤란한 개발자 오류의 범주에 속하는 바이트 정렬, 조건부 코드, 유니코드 등에 관하여서도 개발자의 주의를 요할 수 있도록 반전(block), 색채 효과(coloring)와 같은 다른 형태로 제시하고, 이에 따른 지침서를 제공할 수 있다.

둘째, 개발 에디터의 기능은 시스템 초기화 시 목적지 플랫폼 환경에 요구되는 정보들로 설계된 XML 명세서를 개발 도구 안으로 로드함으로써 개발자에게 구조화된 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.

셋째, 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 개발 플랫폼에서 실행이 실패 한다면 멀티 플랫폼 소프트웨어에 위배된다. 따라서 시스템은 시스템 구축 도구를 사용하여 이를 검증한 뒤 소프트웨어 패키지의 재구축 시 요구되는 스크립트 파일의 작성 시 재구성의 자동화를 제공한다.

넷째, 프로젝트 관리 기능은 새로운 프로젝트 아래서 새로운 멀티 플랫폼 코드를 작성하는 기능과 기존의 소스 파일들을 새로운 프로젝트로 묶어서 입력하여 멀티 플랫폼 코드로 전환하는 기능을 제공한다. 프로젝트에 포함된 파일들의 경우 시스템은 하나의 파일 수정으로 인한 변경이 해당 파일에만 국한되지 않고 포함된 모든 파일들에게 변경을 반영 하며, 기존 소프트웨어의 포팅 시 시스템은 진행에 관련한 통계 수치를 제공한다. 즉 발견된 문제와 이를 수정/미수정한 개수, 이전 작업 라인 번호 등 모든 이전 진행에 관계된 통계치의 추후 사용을 위하여 'Metaconfig' 를 제공한다. 진행 중인 프로젝트의 유지를 위해서는 관리의 용이상 별도 디렉토리로 구분하였다.

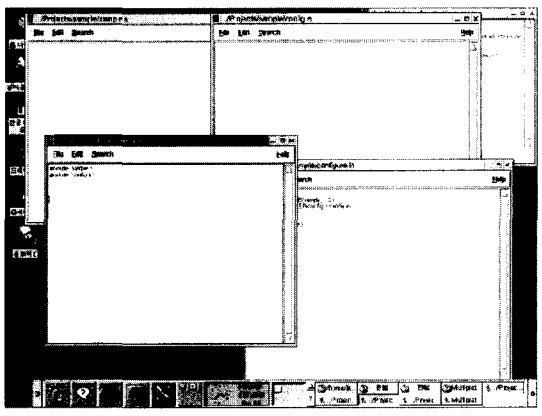
최종 완성된 소프트웨어 패키지를 다양한 목적지 플랫폼에서의 테스트를 위해 크로스 컴파일(cross compile)이 가능하도록 하였으며, 만일 성공적이지 못하다면 'Metaconfig'에 기록된 피드백 정보(error, history 등)를 가지고 개발 싸이클(Development cycle)을 반복 순환하여 수정할 수 있도록 설계하였다.

5. 종합 개발 도구의 구현

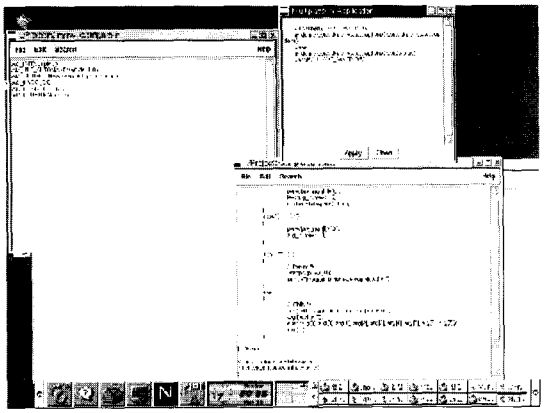
개발 도구는 시스템 비 의존적 실행이 가능하도록 Python 2.0과 Tcl/Tk 8.3을 이용하여 구현하였다.

5.1 초기 환경

초기 실행 시에는 그림 6과 같이 소스와 헤더 파일이 생성되며, 기본 골격을 갖춘 파일 configure.in과 autoheader의 수행결과로 파일 config.h.in이 생성된다.



(그림 6) 초기 실행



(그림 7) 최종 실행

5.2 소스 편집 시 가이드 환경

개발자의 소스 편집 시 문제의 개연성을 지닌 함수(예:waitpid)를 입력할 경우, 검증 모듈이 사용된 함수의 문제점을 포착함과 동시에 문제 함수에 관한 지침서를 대화 상자를 통해 표시하게 되며, 문제의 함수는 다른 색으로 표현된다(그림 7). 개발자는 지침서를 통해 사용된 함수가 왜 문제가 되는지 인지하여 직접 전 처리기 매크로를 작성할 수도 있으며, 에디터의 검증 모듈이 문제함수의 이름과 이에 관련된 데이터베이스의 레코드를 전 처리 모듈에게 전달하고, 전 처리 모듈이 자동으로 전 처리기 매크로를 문제의 함수가 사용된 함수의 상단에 삽입하여 문제를 해결할 수도 있다.

6. 결 론

본 논문에서는 멀티 플랫폼 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 멀티 플랫폼 소프트웨어 제작에 필요한 지식을 제공하고, 대화식으로 적용할 수 있으며, 완전한 해법을 줄 수 없는 사항들은 일반적으로 개발자가 개발 시에 행하는 일들을 도울 수 있는 기능들을 첨가하였다. 필요한 지식은 전문가로부터 얻거나 잘 알려진 멀티 플랫폼 소스 코드에서 추출하여 반영될 수 있도록 설계 및 구현하였다.

참 고 문 헌

[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] J. Wrage, "Wine-HOWTO-User Documentation", http://www.la-sorciere.de/Wine-HOWTO.ps.

[4] Willows Software, "The Willows Toolkit Technical White Paper", http://www.willows.com/whitepaper.html.

[5] Welsh, M.?Porting Applications to Linux*?, January, http://www.cs.berkeley.edu/~mdw/linux/porting-linux/html/paper.html.

[6] E. Zadok, "Autoconfiscating Amd: Automatic Software Configuration of the Berkeley Automounter", http://www.cs.columbia.edu/~ezk/research/amu/autoconfiscating_amd.html.

[7] Trolltech, "Qt Technical Overview", ftp://ftp.trolltech.com/qt/pdf/whitepaper.pd.

[8] G. J. Noer, "Cygwin: A Free Win32 Porting Layer for UNIX Applications", http://sources.redhat.com/cygwin/usenix-98/cygwin.html.

[9] COMPAQ, "Sun Solaris to Compaq Tru64 UNIX Porting Guide", Oct 1999, http://www.tru64unix.compaq.com/faqs/publications/porting/HTML/solaris.html.

[10] 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.

[11] D. MacKenzie, D. Elliston "Autoconf", http://www.gnu.org/manual/autoconf/html_mono/autoconf.html.

[12] D. Mackenzie, T. Tromey, "GNU Automake", http://www.gnu.org/manual/automake/html_mono/automake.html.

[13] G.Matzigkeit, "GNU Libtool", http://www.laria.u-picardie.fr/docs/gnu/libtool/libtool_toc.html.

[14] M. S. Richard, Mc. Roland "GNU make", http://www.gnu.org/manual/make-3.79.1/html_mono/make.html.gz.

[15] 최진우, 박상서, 이진석, 박성우, 이정국, 황선태, 우종우, "포터블 프로그래밍 도구를 활용한 멀티플랫폼 S/W 개발, 정보처리학회 춘계 학술발표집, pp. 1113~1116, 2001.

◐ 저 자 소 개 ◑

최 진 우

1998년 한성대학교 전산학과 졸업(학사)
2000년 국민대학교 대학원 전산과학과 졸업(석사)
2000년~현재 : 국민대학교 대학원 전산과학과 박사과정
관심분야 : 인공지능, ITS, 에이전트, 정보 보호
E-mail : jwchoi@cs.kookmin..ac.kr

황 선 태

1985년 서울대학교 전자계산기공학과 졸업(학사)
1987년 서울대학교 대학원 전자계산기공학과 졸업(석사)
1996년 맨체스터대학교 대학원 전산학과 졸업(박사)
1997년~현재 : 국민대학 컴퓨터학부 교수
관심분야 : 병렬처리, 시스템 소프트웨어, 그리드 시스템
E-mail : sthwang@kookmin..ac.kr

우 종 우

1978년 서울대학교 농생물학과 졸업(학사)
1983년 Minnesota State University at Mankato 전산학과 졸업(석사)
1991년 Illinois Institute of technology 전산학과 졸업(박사)
1994년~현재 : 국민대학교 컴퓨터학부 교수
관심분야 : 인공지능, 지능형 교육시스템, 에이전트, 정보보호
E-mail : cwwoo@kookmin.ac.kr