2002년 한국정보처리학회 춘계학술발표논문집 제9권 제1호

Qplus 타겟 빌더: 임베디드 리눅스 툴킷

강우철*, 윤희철**
*한국전자통신 연구원(ETRI)
**한국전자통신연구원(ETRI)
e-mail : wchkang@etri.re.kr, hcyun@etri.re.kr

Qplus Target Builder: An Embedded Linux Development Toolkit

Woo-Chul Kang*, Hee-Chul Yun**
*Embedded S/W Team, Electronics and Telecommunications Research Institute
**Embedded S/W Team, Electronics and Telecommunications Research Institute

요 약

임베디드 리눅스의 개발은 커널과 각 시스템 응용들을 자신이 원하는 플랫폼에 맞추어 일일이 수작업으로 설정, 컴파일, 타겟에의 인스톨 과정을 거쳐야 한다. 이것은 실제 타겟에 필요한 응용 개발에 앞서 많은 개발 시간을 소요하게 하며, 또한 그 과정 중 많은 시행착오의 반복을 야기하게 한다. Qplus 타겟 빌더는 임베디드 리눅스 개발 과정 중 설정 가능한 사항들을 옵션화하여 개발자가 통합된 환경에서 커널과 응용을 세세하게 설정할 수 있게 하며, 각 옵션들간의 의존성을 자동으로 검사함으로써 타겟에 적합한 설정을 할 수 있게 도와준다. 또한 타겟에 적재(deployment)될 수 있는 형태로 루트파일 시스템을 자동 생성하고 라이브러리 최적화를 통해 사이즈를 최적화시킨다. Qplus 타겟 빌더는 인터렉티브한 타겟 시스템 설정과 자동화된 타겟에의 적재로 임베디드 리눅스의 개발 시간을 단축시켜 준다.

1. 서론

최근 임베디드 시스템의 운영체제로써 리눅스가 큰 관심의 대상이 되고 있다. 시스템 엔지니어의 50% 이상이 자신이 개발중인 시스템에 탑재될 운영체제로 리눅스를 고려하고 있다는 조사는 이를 잘 보여준다.[7]. 이는 리눅스가 소스코드가 공개된 점 외에도 여러 디바이스에 대한 지원, 안정성(reliability), 견고성(robustness), 네트웍킹과 인터넷에 대한 뛰어난 지원등을 장점으로 들 수 있기 때문이다. 그러나 리눅스를 임베디드 시스템에 사용함에 있어서의 여러 장애점도 있는데, 그중 하나가 편리한 개발 환경이 없다는 점이다. 특히 리눅스는 타겟에서 수행될 응용을 개발하기 위해서는 타겟의 기본 동작환경(타겟 루트 파일 시스템)과 개발 환경(네트웍, NFS,DHCP 등의 설정)을 구성해 주어야 하며 이러한 일은 타겟 시스템의 응용 개발과정과 타겟에의 최종 적재 과정에서 서로 조금씩 다르게 반복적으로 수행되어야 한다. 이러한 시스템 설정과 타겟에의 적재 과정의 반복은 툴의 도움없이 개발자의 경험에 따른 수작업으로 이루어 지게 되며 이는 시스템 구축 과정에서 많은 시행착오를 유발하게 된다. 이는 시장화 시간(Time-To-Market)이 중요시되는 상품화에 있어서 치명적인 단점으로 작용한다. 이로 인해 임베디드 리눅스 개발자의 69%는 개발 툴킷의 필요성을 호소했다[7].

Qplus 타겟빌더는 임베디드 리눅스 개발 과정중 설정 가능한 사항들을 옵션화시켜 개발자가 통합된 환경에서 커널과 응용을 설정할 수 있게 하며, 각 옵션들간의 의존성을 자동으로 검사함으로써 타겟에 적합한 설정을 쉽게 할 수 있도록 도와준다. 또한 이러한 설정 결과를 컴파일해서 최종 타겟에 적재될 시스템 이미지를 생성해주고 라이브러리 최적화를 통해 그 사이즈를 최적화 시켜준다.

2. 관련 연구

임베디드 리눅스 개발 툴킷에 대한 이러한 요구로 인해 개발 과정을 자동화 시켜주는 몇몇 상용 툴킷이 개발 되었다.

대표적인 예로 RedHat 의 ELDS[5], Montavista 의 HardHat[4], Lineo 의 Embedix[1][3]를 들수 있다. ELDS 와 HardHat 는 일반 데스크탑 리눅스에서 사용되는 패키지 인스톨 방식과 비슷한 방식으로 타겟에 적재될 응용들을 선택하면 그 응용들과 커널을 모아 타겟에 적재 가능한 이미지를 만들어 주는 도구를 제공한다. 하지만 이들은 커널의 경우 기본 커널 설정 툴을 사용해서 설정 컴파일하며 응용패키지의 경우 응용의 성질에 따른 다양한 설정은 할 수 없고 단지 인스톨 여부만 결정할 수 있다. 또한 커널과 응용패키지를 따로 설정함으로써 둘 간의 의존성기술이 불가능하다 [2][4][5]. Embedix 는 커널과 응용의 설정가능한 사항들을 옵션화하고 옵션간의 의존성등의 정보를 ECD(Embedix Component Description) 파일에 기술할수 있으며 또한 뛰어난 유저 인터페이스를 제공한다. 커널과 응용을 같은 레벨에서 설정하기 위해 커널 옵션까지 ECD 로 기술하는 하는 방식을 사용하고 있다. 그러나 리눅스 커널의 방대한 설정 정보를 모두 ECD 로 바꾸는 것은 매우 힘들고 시간이 걸리는 작업이며 ECD 로의 변환 과정에서 설정 정보가 본래 커널 개발자들의 의도와는 다르게 변질될 가능성이 크다는 문제점을 가지고 있다. [2][3].

3. Qplus 타겟빌더

임베디드 리눅스 툴킷은 1) 사용자가 커널과 기본 시스템 응용, 자신이 개발한 응용을 다양하고 쉽고 설정하고 타겟에 적재 할 수 있도록 도와주어야 하며, 2) 적재되는 최종 이미지는 타겟의 하드웨어에 적합한 사이즈로 최적화되어야 한다.

Qplus 타겟빌더는 1),2)의 요구를 만족시키기 위해서 다음과 같은 기능을 제공한다.

- ✓ 커널과 응용 패키지의 설정가능 사항의 옵션화를 통한 미세설정의 지원
- ✓ 각 옵션간의 의존성 자동 검사
- ✓ 커널과 응용의 동시 설정을 통한 커널과 응용간의 의존성에 대한 자동검사
- ✓ 타겟에 적재될 파일들의 리스트를 설정 가능하게 하고 라이브러리의 사이즈를 최적화 시킴으로써 최종 적재 이미지의 사이즈를 줄임
- ✓ 유저가 쉽게 설정을 하게 도와주는 GUI 방식의 유저 인터페이스
- ✓ 타겟에 적재될 루트파일시스템 이미지의 자동 생성
- ✓ 타겟 초기화 스크립트와 네트웍 주소, 유저정보와 같은 기본 설정사항의 자동 생성

그림 1>은 Qplus 타겟 빌더에서 각 패키지의 옵션이 개발자가 볼수 있는 설정가능한 형태로 변환되고 설정된 후 그에 따라 컴파일 되고 최적화 과정을 통해 타겟에 최종 적재되는 전체과정을 보인다.

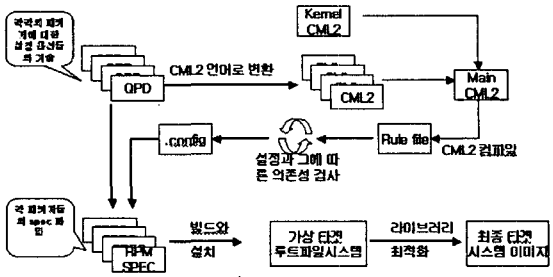

그림 1> Qplus 타겟빌더의 전체 동작 순서

QPD 파일과 옵션의 기술(description)

Qplus 타겟 빌더는 각 응용에서 옵션화 할 사항들을 QPD(Qplus Package Descriptor)에 기술한다. 컴파일에 영향을 미치는 것이나 타겟에 인스톨 할 파일들의 집합등이 옵션 가능한 사항이다. 각 컴파일 옵션이나 파일들은 해당 옵션의 선택여부에 따라 컴파일되고 최종 타겟 이미지에 적재되는 것이다. 아래는 tinyx 라는 패키지의 하위 옵션중 하나인 Xnv 옵션의 예시이다.

```
%option tinyx/Xnv
%%prompt  Xnv server (for nVidia GeForce2MX  cards)
%%require 'UNIX'==y
%%files  /usr/X11R6/bin/Xnv
```

각 옵션은 %%prompt, %%desc, %%requires, %%provide, %%files, %%build_vars 등의 성질을 가질 수 있는데 %%require, %%provide 는 이 옵션이 설정될 때 수행될 의존성을 기술하며, %%files 는 옵션이 선택될 경우 타겟의 루트 파일시스템에 인스톨 될 파일들의 리스트를 기술한다. %%build_vars 는 컴파일 옵션 등을 바꾸고 싶은 경우에 사용된다. 또한 옵션들은 옵션간의 계층 구조를 '/'를 통해서 표시한다. %option /tinyx/Xnv 는 Xnv 가 tinyx 라는 옵션의 하위 옵션임을 나타낸다.각 응용 패키지에 대해 QPD 파일에 옵션들을 통해 컴파일 방식, 인스톨 할 파일리스트, 그리고 옵션과 옵션간의 의존성 기술을 하게 되는 것이다.

커널과 응용 설정간의 의존성의 기술

커널과 여러 시스템 응용 프로그램들은 서로 연관을 가지는데 예를 들어 busybox 패키지의 mount 명령어의 loop 디바이스에 대한 지원을 동작시키기 위해서는 커널의 BLK_DEV_LOOP 옵션이 설정되어야만 한다. 이런 예는 커널과 응용 사이에 양방향으로 존재한다. 이러한 이유로 커널과 응용은 서로 같이 설정되어야 하며 그래야만 둘간의 의존성을 제대로 검사할 수 있게 된다.

리눅스 커널(버전 2.5.x)의 기본설정 도구는 CML2[6] 이다. CML2 는 커널에서 사용하는 매크로 심벌들을 정의하고 이에 대한 룰을 기술하면 각 심벌들의 값을 사용자가 설정할 때 룰에 맞게 설정이 되었는 지를 검사해주고 이에 따라 커널에서 사용할 심벌들의 선택여부를 만들어 주는 시스템이다. 현재 커널

에서 설정해야 하는 심벌은 약 1750 개 이다. 사용자가 이렇게 많은 선택사항을 한번에 접하면 무얼 어떻게 설정해야 하는지를 알 수 없게 된다. 따라서 CML2 는 각 심벌간에 계층구조를 줄 수 있고 설정된 심벌의 값에 따라 다른 심벌들을 점차적으로 보여 줌으로서 설정을 쉽게 할 수 있도록 도와준다.

리눅스 커널은 공개 소프트웨어의 특성상 수시로 새로운 기능이 추가되고 있으며, 이에 따라 커널 설정 옵션들도 그에 따라 계속 변하고 추가되고 있다. 앞서 언급한바와 같이 Embedix 처럼 ECD 파일에 커널의 옵션을 기술하는 것은 커널이 바뀔 때마다 매번 ECD 에 수 메가 바이트에 달하는 양의 정보를 업데이트해야 하는 어려움이 있다. 따라서 커널의 설정은 기본 설정 툴인 CML2 를 사용하게 해야 하는 것이 관리를 수월하게 하는 장점을 가진다.

리눅스 커널의 기능확장에 따른 옵션 기술화일(Qplus 타겟빌더의 QPD 와 Embedix 의 ECD 파일에 해당)의 관리를 수월하게 하고, 커널과 응용을 동시에 설정함으로써 둘 간의 의존성을 자동으로 검사하기 위해서 Qplus 타겟 빌더는 각 응용의 옵션이 기술된 QPD 파일을 CML2 언어로 변환하고, 변환된 각 응용의 QPD 파일과 커널의 CML2 구조를 루트메뉴의 하위 메뉴로써 등록시켜 함께 설정할 수 있도록 한다. 변환시 QPD 파일의 각 옵션이름은 CML2 룰의 심볼로 변환되며, 옵션간의 계층구조는 CML2 의 메뉴간의 계층구조로 변환되어 표현된다. 각 옵션이 가지는 성질은 CML2 로 변환되지 않으며 최종 설정이 일어난 뒤 옵션이 선택된 경우에만 적용되게 된다.

Font End

앞서의 과정을 통해 생성된 커널과 응용의 통합 CML2 룰 파일은 Qplus 타겟 빌더의 Front-End 에 의해 보여지고 이를 통해 유저는 각종 옵션을 선택하게 된다. 그림 2>는 타겟빌더의 실행 모습을 보인다.

유저가 각 옵션을 선택하면, 그에 대한 설명과 현재 상태가 표시되며, 유저는 이를 보아가면서 각 옵션을 선택하게 된다.

설정은 항목은 다음과 같이 주요 3 가지 부분으로 구성 되어 있다.

- ✓ 리눅스 커널
- ✓ 응용 패키지
- ✓ 타겟 환경(타겟의 네트웍 주소, 기본 사용자 설정, 부팅 방식 결정)

리눅스 커널 부분은 배포되는 리눅스 커널의 CML2 룰을 그대로 표시하고 있으며 용용 패키지 부분은 등록된 각 응용의 QPD 파일의 옵션 사항들이 CML2 로 변환되어 보여지는 것이다. 타겟 환경부분은 타겟이 수행되기 위해 필요한 기본 정보인 네트웍 주소, 기본 유저의 등록, 부팅방식과 라이브러리 최적화 여부등을 설정할 수 있는 부분이다.

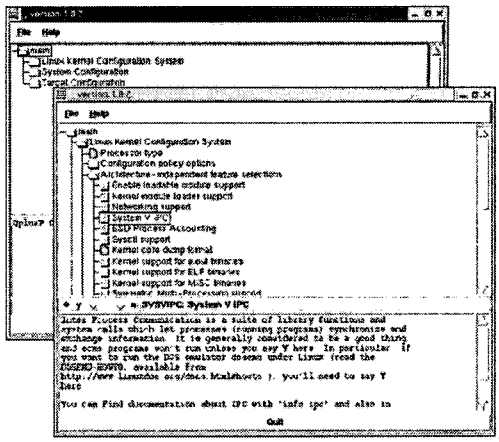

그림 2> Qplus 타겟 빌더의 수행 모습

위의 타겟빌더의 GUI 는 한번에 한가지 메뉴만 사용자에게 보여지고 설정 가능한 CML2 의 기본 Front-End 를 한 화면에서 트리 형태로 위 세가지 부분의 각 옵션의 모습을 전체적으로 보아가면서 설정할 수 있도록 개선한 것이다. 이를 통해 유저는 각 옵션을 버튼클릭만으로 설정해 나갈 수 있다.

의존성(dependency)의 기술과 변환

각 옵션의 값을 설정할 때 QPD 파일과 커널의 CML2 파일에 기술된 룰에 따라 제대로 설정이 되었는지 매번 검사가 일어난다. QPD 파일 각 옵션의 %%require 필드에 각 옵션이 필요로 하는 의존성 규칙을 명시할 수 있는데, 다른 패키지의 옵션과 커널 심볼의 설정에 관한 규칙이 기술된다. 예를 들어 procps 가 ncurse 라이브러리를 필요로 하고 busybox 패키지의 ps 명령은 함께 인스톨 되지 말아야 할 경우 아래와 같이 %%require 문을 추가하면 된다.

```
%package procps
%%require ncurse==y and busybox/ps==n
...
```

이처럼 %%require 에 기술된 의존성 룰은 다음과 같은 CML2 의 의존성 규칙으로 자동 변환된다.

PKG_PROCPS_OPT implies PKG_NCURSE_OPT==y and PKG_BUSYBOX_OPT_PS ==n

이렇게 변환된 CML2 의존성 규칙은 CML2 엔진에 의해 심벌 값이 변할 때마다 매번 체크되고 침범 여부를 표시하게 된다. 의존성 규칙에 어긋나는 설정이 이루어질 경우 설정에 따른 부작용(side-effect)을 모두 원래대로 되돌리게(roll-back) 된다. 이와 같은 의존성 자동 체크는 수많은 옵션을 설정함에 있어서 리눅스에 경험이 없는 개발자는 물론 고급 개발자들에게도 많은 도움을 주게 된다.

설정결과의 저장과 빌드

이렇게 개발자에 의해 설정된 결과값은 *'심벌=[y/n]'* 의 형식으로 기록되는데 커널은 이를 *'#define 심벌 [1/0]'*의 형태로 변화해서 컴파일에 이용하며, 각 응용 패키지는 컴파일에 영향을 미치는 설정(%%build_vars 필드를 가지는 옵션이 바뀐경우)이 바뀐 경우만 새로 컴파일하고 그렇지 않고 타겟에 깔릴 파일 리스트등의 필드만 가지는 옵션만 선택된 경우는 컴파일은 하지 않고 가상의 타겟의 루트 파일시스템을 만들 때 선택된 옵션에 기술된 파일리스트에 따라 인스톨 할 파일을 결정한다.

타켓에의 적재(deployment)와 라이브러리의 최적화

Qplus 타겟빌더는 이렇게 생성된 가상의 타겟 루트 파일시스템에서 공유 라이브러리를 최적화 할 수 있도록 하고 있는데, 이는 파일과 심벌단위의 최적화를 동시에 사용할 수 있다.

파일단위의 최적화는 공유 라이브러리중 현재 타겟에 적재 될 응용과 다른 라이브러리들이 필요로 하는 라이브러리 파일만 추려내서 적재하는 것이며, 심벌단위는 한 단계 더 나아가 선택된 공유 라이브러리들에서 각 응용과 라이브러리가 실제로 사용하는 심벌만 추려내서 새로운 라이브러리를 만들어 내는 방식이다.

타겟에 적재되는 응용이 제한적인 경우 이러한 라이브러리 최적화를 통해 최종 타겟에 적재되는 이미지의 사이즈를 줄일 수 있다.

4. 평가

Qplus 타겟빌더 툴을 사용해서 얼마만큼 작업이 편해지고 단축되는가를 수치적으로 측정하는 것은 쉽지 않지만 ETRI 내에서 수행중인 홈서버에 탑재 될 임베디드 리눅스 시스템 개발에 있어서 Qplus 타겟 빌더를 사용한 결과 2 ~ 7 일 정도 소요되는 개발 환경 구축 시간이 툴의 사용으로 모두 자동화되었으며, 개발과정 중 반복적으로 나타나는 각 응용과 커널의 설정과 최적화 등도 모두 자동화되어 홈서버에 탑재될 전자북과 같은 응용의 개발자는 임베디드 리눅스에 대한 별 배경지식 없이 개발을 시작할 수 있었다.

또한 타겟에 탑재되는 라이브러리는 Qplus 타겟빌더의 자동화된 최적화 과정을 통해 그 사이즈를 줄일 수 있었는데 표 1>은 이를 보인다.

	일반	파일단위최적화	심벌단위최적화
파일개수	60 개	14 개	14 개
전체사이즈	1,425KB	1,256KB	674KB

표 1> 라이브러리 최적화를 통한 사이즈 감소

파일단위최적화를 통해 실제로 쓰이는 라이브러리 파일들만 추출한 결과 2Mbyte 정도의 사이즈를 줄었으며, 또한 그 중에서도 쓰이는 심벌들만 추출한 결과 추가적으로 6M 바이트의 라이브러리 사이즈를 줄일 수 있었다.

5. 결론

임베디드 리눅스 개발에 있어서 그 과정을 자동화하기위해 발표된 툴들은 커널과 응용간의 의존성의 기술, 편리한 유저 인터페이스등에서 아직 만족할만한 기능을 제공하고 있지 않다[3][4][5].

Qplus 타겟 빌더는 임베디드 리눅스 개발 과정중 설정 가능한 사항들을 옵션화시켜 개발자가 통합된 환경에서 커널과 응용을 세세하게 설정할 수 있게 하고, 각 옵션들간의 의존성을 자동으로 검사함으로써 타겟에 적합한 설정을 할 수 있게 도와준다. 또한 최종 타겟 이미지는 라이브러리 최적화 과정을 통해 최소의 Footprint 를 보이게 된다.

이러한 자동화된 타겟 시스템의 설정과 타겟에의 적재는 임베디드 리눅스의 개발 시간을 단축시키는 효과를 가지며 빠른 시간내에 임베디드 리눅스를 개발하고자 하는 많은 개발자들에게 도움이 될 것이다.

참고문헌

[1] "Embedix Users Guide":,
ftp://ftp.lineo.com/pub/docs/embedix/user_guide.pdf
[2] "A developer's review of the leading Embedded Linux toolkits",
http://www.linuxdevices.com/articles/AT8402180338.html
[3] "A developer's review of Lineo's Embedix SDK,"
http://www.linuxdevices.com/articles/AT3702738504.html
[4] "A developer's review of MontaVista's Hard Hat Linux SDK",
http://www.linuxdevices.com/articles/AT6698549967.html
[5] "A developer's review of Red Hat's Embedded Linux Developer Suite, "
http://www.linuxdevices.com/articles/AT9802104051.html
[6] "The CML2 Language:Constraint-based configuration for the Linux kernel and elsewhere",
http://www.tuxedo.org/~esr/cml2/cml2-paper.html
[7] "The 2001 Embedded Linux Market Survey",
http://www.linuxdevices.com/cgi-bin/survey/survey.cgi?view=results&id=12292000143241