*테스트가 가능한 부분적으로 완성된 실행파일을 통한 개발기한 단축에 관한 연구

조재현*, 유혁**
고려대학교 컴퓨터전파통신공학과
e-mail : {jhjo,hxy}@os.korea.ac.kr

A Study for Shortening Development Time through Partially Implemented Test Software

JaeHyun Jo*, Chuck Yoo**
Dept. of Computer and Radio Communication Engineering, Korea University

요 약

모바일 환경에서 새로운 하드웨어를 장착한 모델을 시장에 적기에 출시하는 것이 중요하다. 그러나 새로운 하드웨어 테스트와 관련이 없는 모듈의 인터페이스 변경에 의한 unresolved symbol 에 의한 링크에러로 인해 테스트 실행파일의 생성이 늦어져 새로운 하드웨어의 검증이 늦어지고 이로 인해 전체적인 개발일정이 지연 되는 불합리한 경우가 발생 하는 경우가 많았다. 본 논문은, 새로운 하드웨어 검증과 관련이 없는 모듈의 unresolved symbol 을 수정하지 않은 상태에서 실행 파일을 생성할 수 있도록 하는 링커를 제안하고, 테스트가 가능한 최소한의 모듈만 가지고도 unresolved symbol 에 실제 의미 있는 접근이 발생 하기 전까지 하드웨어 검증이 가능하게 함으로서 전체 개발기한을 단축 할 수 있는 방법을 제시한다.

1. 서론

모바일 환경에서 새로운 CPU 나 주변장치를 장착한 하드웨어를 적기에 시장에 출시하기 위한 경쟁이 치열하다. 이를 위해서 항상 요구되는 것이 개발기한의 단축이다. 보통 새로운 하드웨어를 사용하는 프로젝트의 소프트웨어 개발 단계에서 처음으로 해야 하는 일이 새로운 하드웨어를 적용한 보드가 정상적으로 설정이 되어 동작하는지 확인하며 부팅을 시켜 보는 것이다. 부팅과정을 통해 새로운 하드웨어 설정이 정확한지, 기본적인 동작은 정상적으로 이루어 지는지 검증을 하게 된다. 그러나 보통 모바일 환경에서는 여러 모듈을 정적링크(static link)하여 만들어 지는 하나의 이미지로 실행파일이 구성되는 경우가 대부분이다. 따라서 부팅 및 하드웨어 테스트와 상관 없는 모듈이더라도 필요한 함수나 변수가 정의되지 않으면, 즉 unresolved symbol 을 가지게 되면 링크단계에서 에러가 발생하고 이로 인하여 실행파일을 만들 수 없게 되어 새로운 하드웨어를 검증할 수 없게 된다. 필요한 것은 오직 부팅 과정을 통하여 새로운 하드웨어가 정상 동작하는지 확인 하는 일인데, 이와 전혀 상관이 없는 자바, 브라우저 모듈의 unresolved symbol 에 의해 실행파일을 만들 수 없게 되어 새로운 하드웨어 검증 작업이 지연되는 불합리한 상황이 발생 하게 된다. 이러한 상황은 결국 적기에 제품 출시를 지연시키는 상황으로 이어진다.

개발 초기에 테스트 실행파일 생성이 어려운 이유를 살펴보자. 보통 새로운 제품은 새로운 CPU 나 주변장치를 적용하는 경우가 대부분이고, 이 때 모델의 특성에 따라 링크되는 소프트웨어 모듈 구성이 달라지고 이에 따라 모듈간의 인터페이스가 변경이 된다. 또한 CPU 가 변경 되는 경우에 새로운 CPU 에 맞는 컴파일 옵션으로 각 모듈을 새로 빌드하여 취합한 후 링크 과정을 통해 실행파일을 만들어야 한다. 이와 같이 수많은 모듈의 추가삭제와 모듈간 인터페이스 변경, 빌드 옵션 변경으로 인하여 모듈을 취합하여 바이너리 만드는 과정에서 수 많은 unresolved symbol 에 의한 링크에러 발생이 필연적이다.

문제를 더 어렵게 하는 것은 모바일 소프트웨어는 하나의 회사에서 모든 소프트웨어를 개발하는 것이 아니기 때문에, 모듈을 제공하는 모든 회사에서 빌드 옵션 및 호출 관계 등을 정리하여 링크에러가 발생하지 않아야 실행파일을 생성할 수 있다는 데에 있다. 따라서 새로운 하드웨어를 적용한 모델의 경우에 관련된 회사들과의 조율이 필요하기 때문에 테스트 바이너리 생성에 오랜 시간이 걸리는 것이 보통이다.

본 논문은 새로운 하드웨어 검증과 관련이 없는 모듈의 unresolved symbol 을 수정하지 않은 상태에서 테스트가 가능한 최소한의 모듈을 통해 실행 파일을 생성할 수 있도록 하는 링커를 제안하고, unresolved symbol 에 실제 의미 있는 접근이 발생 하기 전까지

* 이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2010-0029180)

하드웨어 검증이 가능하게 함으로써 전체 개발 기한을 단축 할 수 있는 방법을 제시한다.

2. 관련연구

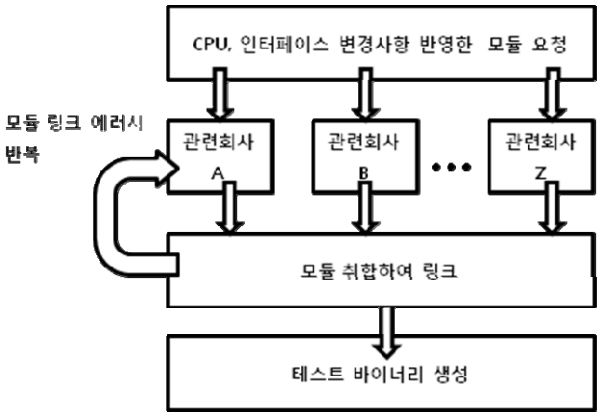
테스트와 관련 없는 모듈의 링크 에러로 인해 테스트 실행파일을 만들 수 없는 문제를 해결하는 방법에는 크게 두 가지 방식이 있다. 동적링크(dynamic link)를 이용하는 것과 링커차원에서 unresolved symbol 을 링크에러로 처리하지 않고 지정한 default symbol 로 연결해 줌으로 써 실행파일을 만들어 주는 것이다.

동적링크를 사용하면 테스트와 관련이 없는 모듈이 라이브러리 형태로 분리되기 때문에 테스트 실행파일을 만들 수 있고, 테스트와 관련 없는 모듈이 호출되지 않으면 동적링킹이 수행되지 않기 때문에 테스트를 진행 할 수 있다. 그러나 REX, Nucleus 같은 임베디드 환경에서 주로 사용되는 실시간 운영체제에서는 동적링크 기능을 지원하지 않는다. 개발 업체에서 실시간 운영체제를 수정하여 동적링크 기능을 구현하는 것도 생각해 볼 수 있지만, 해당 운영체제는 칩 벤더에서 BSP(Board Support Package)로 제공하기 때문에 동적링크 기능을 구현하였다고 하더라도, 칩 벤더의 운영체제의 코드변경에 따라 구현을 수정해 주어야 하는 등 유지 관리에 어려움이 있다.

다른 방법으로 링커에서 지원하는 방법이 있는데 임베디드 환경에서 주로 사용되는 ARM 링커에서 테스트 실행파일 생성에 대한 요구를 파악하고 부분적으로 완성된 실행파일 지원을 위한 옵션을 제공하고 있다. ARM 링커의 –unresolved 옵션을 통해 임의의 unresolved 함수를 지정한 default 함수로 링킹해 주도록 하여 unresolved 함수를 가지더라도 실행 파일을 만들 수 있도록 하고 있다. 그러나 모든 unresolved 함수를 특정 하나의 전역함수에 링크하면서 unresolved 함수와 연결된 주소간에 N:1 의 관계가 성립하게 된다. 따라서 unresolved 함수로 연결된 default 함수에 진입하였을 때 진입한 주소를 보고 어떤 unresolved 함수가 호출된 것인지 알 수 없는 문제가 발생한다. 어떤 unresolved 함수가 호출되었는지 알 수 있어야 해당 함수가 테스트 수행에 필요한 함수인지 판단을 하게 되고, 이를 근거로 테스트의 유효성을 보장 할 수 있는데 호출된 함수의 이름을 알 수 없기 때문에 테스트 중 unresolved 함수가 호출되면 그 테스트의 유효성을 보장할 수 없게 된다. 또한 모듈간의 인터페이스 변경으로 인해 전역변수가 unresolved symbol 문제를 일으키는 경우가 많은데, 전역변수의 경우에는 default symbol 로의 연결을 지원하지 않아 실효성이 떨어진다.

3. 기존 테스트 실행파일 생성과정의 문제점

기존 방식에서는 아래 그림 1 에서 보듯이 테스트와 관련이 없더라도 실행파일에 포함되는 모듈을 개발하는 모든 회사에서 링크에러를 수정해 주어야 테스트 바이너리 생성이 가능하고, 이 링크에러를 수정하는 과정에서 개발지연이 발생한다.



(그림 1) 기존 테스트 바이너리 개발 방식

4. 테스트가 가능한 부분적으로 완성된 실행 파일을 통한 개발기한 단축

칩 벤더가 BSP 형태로 제공하는 실시간 운영체제에서 동적링크 기능을 칩 사용 회사에서 구현하는 것은 구현 및 유지보수에 측면에서 많은 비용이 들기 때문에 좋은 선택이 아니다. 링커를 통해 unresolved symbol 을 가지더라고 테스트를 위한 실행 파일 생성을 지원 하도록 하고, 테스트 과정에서 unresolved symbol 에 접근이 발생 한 경우에 해당접근이 테스트의 유효성에 영향을 미치는지를 개발자가 정확히 판단 할 수 있도록 정보를 제공해 주는 형태가 되어야 한다.

본 논문은 이와 같은 요구사항을 만족시키기 위해 새로운 링커옵션을 정의하여 부분적으로 완성된 실행파일을 생성하고, abort handler 를 통하여 어떤 unresolved symbol 에 접근 했는지를 개발자가 알게 하여 테스트 유효성을 판단할 수 있도록 하는 방안을 제시한다.

4-1. 테스트가 가능한 부분적으로 완성된 실행 파일을 위한 링커옵션 정의

기존 ARM 링커는 부분적으로 완성된 실행 파일생성을 지원하지만 모든 unresolved symbol 을 하나의 default symbol 에 연결 함으로서 unresolved symbol 에 접근했을 때 어떤 unresolved symbol 에 접근이 되었는지 알 수 없는 문제가 있다. 이 문제를 해결하기 위해 unresolved symbol 을 지정한 하나의 default symbol 이 아닌 각각의 고유한 주소 값에 연결해 주어야 한다. 즉 unresolved symbol 과 연결된 주소 값이 1:1 관계가 되어야 한다.

이를 위한 링커 옵션을 다음과 같이 정의한다.

```
linker –unresolved_variable = address
       –unresolved_ function = address
```

해당 옵션은 unresolved 변수와 unresolved 함수를 지정된 주소에 4byte 만큼씩 차례로 증가하면서 연결하여 각각의 unresolved symbol 이 고유한 주소를 가지도록 한다.

이를 위하여 아래 기술한 기존 링커의 symbol resolution 과정의 4) 단계를 4`)단계로 다음과 같이 변경한다.

링커의 symbol resolution 과정

1) 링커의 명령어 라인으로부터 .o 파일과 .a(라이브러리)파일을 읽어 들인다.

2) 1)의 과정 동안 unresolved reference 의 리스트를 유지한다.

3) 새로운 .o 또는 .a 파일을 만날 때 마다 unresolved reference 를 resolve 한다.

4) 더 이상의 .o 파일과 .a 파일이 없는데 unresolved reference 가 있는 경우에 링크에러처리 한다.

여기에 4)번 과정을 4`)와 같이 변경 한다.

4`) 더 이상의 .o file 과 .a 파일이 없는데 unresolved reference 가 있는 경우에 지정한 default symbol 주소에 연결하고 다음에 연결할 default symbol 주소를 4byte 만큼 증가한다.

제안한 링커 옵션을 통해 모듈이 unresolved symbol 을 가지더라도 unresolved symbol 에 고유한 주소를 할당한 실행파일을 생성할 수 있고, 테스트 중 특정한 unresolved symbol 에 접근한 경우에 할당된 고유주소를 통해 어떤 unresolved symbol 에 접근한 것인지 알아 낼 수 있다.

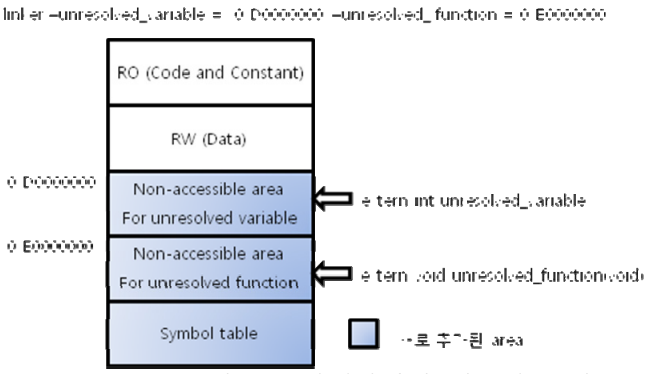
4-2. Unresolved symbol 접근 시 처리법

테스트 실행파일 수행 중에 unresolved symbol 에 접근하는 경우에 default 로 지정한 주소영역의 의미 없는 값이나 코드를 실행하여 오동작할 수 있다. 따라서 unresolved symbol 에 접근이 발생하는 경우에 예외상황임을 개발자에게 알려 해당 변수를 접근하는 것이 테스트에 주는 영향을 판단하도록 하여 테스트를 계속 진행할지 멈출지를 선택할 수 있게 해야 한다. 따라서 unresolved symbol 이 링크되는 주소영역을 non accessible 영역에 위치시켜 해당 주소에 접근발생시 a abort 에 의한 예외가 발생 하도록 한다. Linker descript script 통해 unresolved symbol 이 위치하는 주소영역을 분리하여 설정하고 MMU page table 속성을 통해 해당 영역을 non-accessible 영역으로 설정 함으로서 해당 영역에 접근 시 abort 를 발생시킨다. 또한 ELF 의 symbol table 정보를 메모리에 포함하여 unresolved symbol 에 접근한 경우에 할당된 고유한 주소 값을 키 값으로 사용하여 실제 어떤 변수나 함수에 접근했는지를 알 수 있게 한다. 이 정보는 unresolved symbol 에 접근한 경우에 이 변수나 함수가 테스트에 영향을 주는 것인지를 판단하여 테스트를 계속할지 멈출지를 판단하는 기준으로 사용된다. 예를 들어, 테스트 중에 브라우저 모듈의 InitBrowser()라는 초기화 함수에 접근이 발생한 것을 예외를 통하여 알게 되면 개발자는 이 함수가 수행되지 않더라도 테스트에 영향이 없다고 판단하여 테스트를 계속 진행할 수 있다.

예외적으로 배열이나 구조체와 같이 4byte 를 넘는 변수를 arVariable[9]과 같이 base 주소에 offset 을 더한 형태로 접근하는 경우에는 실제 접근한 변수가 아닌 다른 변수에 접근한 것으로 잘 못된 정보를 줄 수 있다. 왜냐하면 본 논문에서 제안한 링커에서 unresolved 변수의 정확한 크기를 알 수 없어 4byte 로 가정했기 때문이다. 따라서 unresolved 변수에 접근 하는 경우에는 어느 함수에서 어떤 unresolved 변수를 접근 했는지를 나타내어 접근된 unresolved 변수가 해당 함수에서 접근할 수 있는 변수인지를 판단해야 한다. 때에 따라서는 개발자가 소스코드와 비교해 봐야 실제 어떤 unresolved 변수를 접근 했는지 알 수 있다.

그러나 보통 모듈간에 인터페이스는 함수로 이루어져 있고 전역변수를 직접 참조하는 경우는 드물기 때문에 이러한 예외적인 경우가 크게 문제가 되지 않는다.

아래 그림 2 가 테스트를 위해 제시한 링커옵션을 통해 만들어진 부분적으로 완성된 실행파일의 메모리 구성이다.



(그림 2) 테스트 실행파일의 메모리 구성

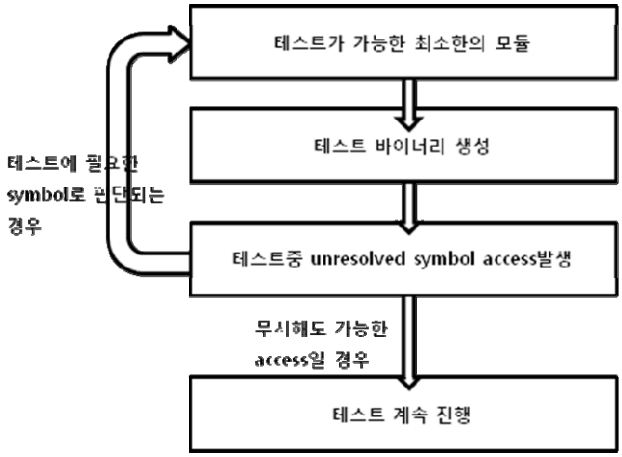
테스트 파일 실행 중 unresolved symbol 에 접근이 발생하면 접근영역이 read/write 가 불가능 하기 때문에 unresolved 변수에 접근한 경우에 data abort 가 unresolved 함수에 접근한 경우 prefecth abort 가 발생한다. 따라서 data/prefetch abort handler 에서 symbol table 을 분석하여 실제 어떤 symbol 에 접근했는지를 개발자에게 보여주고, 테스트를 계속할지 멈출지 선택 할 수 있도록 한다 테스트를 계속 진행 하는 것을 선택하면 unresolved symbol 을 접근한 다음 명령으로 진행하도록 한다. 따라서 data/prefetch abort handler 는 각각 다음 그림 3 의 의사코드와 같이 작성되어야 한다.

```
void Data_Prefetch_Abort_Handler(int pc, int accessed_address)
{
  1.if (accessed_address가 unresolved symbol area에 속하면 )
  {
    2. accessed_address 값을 가지고 symbol table을 통해 어떤
       unresolved symbol에 access가 발생 했는지를 알아 낸다.

    3.해당 unresolved symbol에 access가 발생 했음을 화면에 표시하고
      개발자로 부터 테스트 바이너리의 실행을 멈출지 계속 진행할지를
      판단하게 한다.

      if (계속 진행 선택)
        pc 다음의 명령어로 분기한다.
      else
        실행을 멈춘다.
  }
}
```

(그림 3) data /prefetch abort handler 의 의사코드

4-.3 테스트가 가능한 부분적으로 완성된 실행 파일을 통한 테스트 방식

그림 4 는 본 논문이 제안하는 테스트가 가능한 부분적으로 완성된 실행 파일을 통한 테스트방식을 나타낸다.



(그림 4) 제안하는 테스트 바이너리 개발 방식

본 논문이 제안 하는 방식은 테스트가 가능한 최소한의 모듈만 갖추어져 있으면 테스트 바이너리를 생성할 수 있고, 해당 바이너리를 통하여 원하는 테스트를 수행 할 수 있다. 테스트 중에 unresolved symbol 에 접근이 발생 하는 경우에 어느 symbol 에 접근했느냐에 따라 테스트를 계속 진행할지, 테스트를 중단하고 다시 unresolved symbol 문제를 해결하여 다시 테스트 바이너리를 만들어야 할지를 결정 할 수 있다.

테스트 중 unresolved symbol 에 접근이 발생 하는 경우라도 이것이 테스트 중 사용하지 않는 모듈의 초기화에 관련된 접근이라고 하면 무시하고 넘어가도 테스트에 전혀 지장을 주지 않을 것이다. 또한 테스트에 필요한 unresolved symbol 에 접근하는 경우라 해도 unresolved symbol 에 접근하는 시나리오를 피해서 테스트를 하면서, 병렬로 unresolved symbol 을 가진 모듈의 인터페이스를 정리 할수 있기 때문에 개발시간 절약측면에서 종래에 방법에 비해 훨씬 효율적이다.

5. 결론

모바일 환경에서 경쟁력을 가지기 위해 시장 적기 출시를 위한 개발시간 단축이 중요하다. 본 논문은 테스트가 가능한 최소한의 모듈만 갖추어진 상태에서 실행파일 생성을 통해 필요한 테스트가 가능하게 하는 방법을 제시함으로써 테스트와 관련 없는 모듈의 문제로 인하여 테스트 실행파일이 생성이 되지 않아 테스트를 진행 할 수 없는 문제의 해결 방안을 제시하였다.

참고문헌

[1]Randal E. Bryant and David O'Hallaron, "Computer Systems A Programmer's Perspective", Prentice Hall, New Jersey, 2003

[2]John R. Levine, "Linkers and Loaders", Morgan Kaufmann, San Francisco,1999

[3]ARM Linker Reference
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0493a/CHDEHBDG.html

[4] Executable and Linkable Format(ELF).Version 1.1 TIS Committee