이동단말기 환경에서 응용프로그램 로더와 동적 링커 개발

김유일* 이원재*O 한환수* 이재호+ 김선자+
한국과학기술원 전자전산학과 전산학전공*, 한국전자통신연구원 무선인터넷 플랫폼팀+
{youil.kim, wonjae.leeO, hwansoo.han}@arcs.kaist.ac.kr {bigleap, sunjakim}@etri.re.kr

Developing application loader and dynamic linker for mobile devices

Youil Kim* Wonjae Lee*O Hwansoo Han* Jaeho Lee+ Sunja Kim+
Div of Computer Science, Dept of EECS, KAIST*, Electronics and Telecommunications Research Institute+

요 약

응용프로그램 로더와 동적 링커는 응용프로그램이 실제로 실행되기까지의 준비 과정을 담당하는 프로그램들이다. 최근의 WIPI 표준은 휴대폰에 새로운 응용프로그램을 전송 받아 수행할 수 있는 환경을 제안하고 있지만, 휴대폰과 같은 제한된 이동단말기에서 사용할 만한 로더와 동적 링커는 공개되어 있지 않다. 본 논문에서는 이동단말기 환경에서 응용프로그램 로더와 동적 링커를 개발하는 과정을 설명한다. 개발에 참가한 연구진의 경험을 소개함으로써 이후의 비슷한 환경에서 로더와 링커를 개발하려는 연구자들에게 중요한 참고자료로 활용될 수 있을 것이다.

1. 서론

무선인터넷은 점차 우리 실생활에 필수적인 통신 수단으로 자리잡고 있다. 이러한 무선인터넷의 확산에도 불구하고, 기존의 제한된 컨텐츠 접근, 속도 한계, 이동통신사 간의 컨텐츠 호환성 등 해결되지 않은 문제점들이 많이 지적되었는데, 표준화는 이와 같은 문제점을 극복하기 위한 효과적이고 근본적인 해결책이다. 국내에서는 이동통신 3사와 전파연구소, ETRI 등을 주축으로 한 전문가와 개발자들이 2001년 7월부터 1년여 간 국내 모바일 플랫폼 표준 개발 계획을 추진하여 WIPI(Wireless Internet Platform for Interoperability)[2, 3]를 만들었다.

WIPI 표준이 제안하는 환경은 네이티브 코드로 컴파일 된 응용프로그램을 휴대폰에 전송 받아 수행하는 것이 가능한 환경이다. 이와 같은 환경에서는 보다 복잡한 운영체제에서나 필요했던 응용프로그램 로더와 동적 링커가 필요하다. 공개되어 있는 로더와 동적 링커 - 대표적으로 리눅스의 로더와 동적 링커를 휴대폰에 그대로 이식하는 방향은 적합하지 않은 것으로 보인다. 우선 기존의 코드는 너무 복잡하여 이동단말기와 같은 제한된 환경에서 사용할 만한 효율적이고 작은 코드로 컴파일하기 어려웠다. 또한 전체적인 구조도 다소 변경되어야 했다.

본 논문에서는 이동단말기 환경에서 응용프로그램 로더와 동적 링커를 개발하는 과정을 설명한다. 본 연구진이 개발 중에 겪은 어려움과 해결 과정을 소개하여, 이후의 연구자들에게 참고자료로 활용되는 것을 목적으로 한다.

2. 실행 파일 형식

어떤 형식의 실행 파일을 대상으로 하느냐에 따라 응용프로그램 로더와 동적 링커의 구현 내용이 달라진다. 본 연구진은 현재 리눅스 환경에서 가장 많이 사용되는 실행 파일 형식인 ELF(Executable and Linking Format)를 사용하기로 결정했다. ELF가 정의하는 실행 파일 포맷은 매우 유연하여, 불필요한 정보를 제거하여 더 작은 실행 파일을 만들 수도 있고, 필요에 따라 새로운 정보를 추가할 수도 있다[4]. 본 연구에서 사용한 이동단말기는 ARM7TDMI 코어를 사용한 프로세서를 탑재했는데, ARM 기반 환경을 위한 ELF 규격[5, 6]이 잘 정리되어 있고, 대부분의 컴파일러 환경이 ELF를 잘 지원한다는 것이 ELF를 선택하게 된 이유이다.

프로그램의 목적에 따라, ELF 파일은 두 가지 관점에서 바라볼 수 있도록 되어 있다. 어셈블러나 정적 링커는 섹션 헤더 테이블을 통해 파일을 섹션들의 모음으로 보게 되고, 로더나 동적 링커는 프로그램 헤더 테이블을 통해 파일을 세그먼트의 모음으로 바라보게 된다. 파일을 섹션 단위로 처리하는 경우가 더 자세한 정보를 얻을 수 있지만, 로더나 동적 링커의 관점에서는 섹션별 정보를 구분하는 것 보다는 섹션들의 모음인 세그먼트 단위의 정보를 가지고 읽기전용, 읽기/쓰기 접근 권한을 가지도록 메모리에 로드하도록 처리하는 것이 더 빠른 속도를 얻을 수 있기 때문이다.

2.1. ELF 헤더

ELF 파일은 항상 ELF 헤더로 시작한다. ELF 파일 내에서 고정된 위치를 가지는 정보는 ELF 헤더뿐이다. ELF 파일을 다루는 프로그램은 가장 먼저 ELF 헤더를 읽어 파일의 전체적인 구조와 해석 방법을 파악해야 한다. ELF 헤더는 ELF 버전, 시스템 정보, 그리고 섹션 헤더 테이블과 프로그램 헤더 테이블의 위치 및 크기 등의 정보를 담고 있다.

2.2. 심볼 테이블(symbol table)

ELF 파일은 심볼 테이블을 포함한다. ELF 파일에 나타나는 심볼의 대표적인 예는 코드에서 사용하는 함수의 이름이다. 심볼 테이블은 재배치와 링킹 과정에서 사용되며, 코드가 수행되는 중에는 사용되지 않는다.

2.3. 재배치 테이블(re-location table)

재배치란 코드와 데이터의 기반 주소(base address)가 달라지는 경우에 관련된 코드와 데이터를 적절히 수정하여 코드가 올바르게 수행될 수 있도록 해 주는 작업을 말한다. ELF 파일에 포함되는 재배치 테이블은 재배치를 수행할 주소와 재배치 방법을 나열한 것이다. ELF 파일은 정적 링커가 사용하는 재배치 테이블과 동적 링커가 사용하는 재배치 테이블, 이렇게 두 종류의 재배치 파일을 포함할 수 있다.

2.4. 동적 세그먼트(dynamic segment)

동적 세그먼트는 심볼 테이블, 심볼의 해시 테이블, 재배치 테이블 등, 동적 링커가 실행 파일에서 필요한 정보를 빠르게 얻고, 코드와 데이터를 적절히 수정하는데 필요한 정보를 담고 있다. 동적 세그먼트는 코드가 수행되는 중에는 사용되지 않지만, ELF 로더와 동적 링커의 일반적인 동작은 응용프로그램을 메모리에 적재할 때 동적 세그먼트도 메모리에 적재하여, 동적 링킹 과정에서 동적 세그먼트의 정보를 빠르게 얻을 수 있도록 하는 것이다.

3. 크로스 컴파일러 환경

크로스 컴파일러란 컴파일러가 수행되는 시스템과 다른 시스템에서 수행될 수 있는 코드를 생성하는 컴파일러를 말한다. 이동단말기 위에서 직접 컴파일러를 수행하는 것은 곤란하므로 일반적으로 크로스 컴파일러를 이용하여 이동단말기에서 수행할 응용프로그램을 생성한다.

본 연구진은 GCC(GNU Compiler Collection)와 GNU 바이너리 유틸리티, 그리고 Newlib[7]를 이용하여 ARM 기반 시스템을 위한 크로스 컴파일러 환경을 구축했다. GCC는 많이 사용되는 상용 컴파일러인 ADS(ARM Developer Suite)에 못지 않은 성능을 가진 것으로 알려져 있다. 응용프로그램은 다음과 같은 형태로 컴파일하여 실험하였다.

3.1. 시동 코드 제거

특별한 옵션 없이 프로그램을 컴파일하면, C 런타임 시동 코드(Newlib를 이용하는 경우 crt0.o)라고 불리는 코드가 실행 파일에 추가된다. 그러나 추가되는 시동 코드가 하는 일들은 일반적인 UNIX 환경에서 필요한 부분으로, 본 연구진이 사용한 이동단말기 환경에서는 불필요한 작업이므로 시동 코드를 실행 파일에 포함하지 않도록 컴파일하였다.

3.2. Thumb 인스트럭션

ARM 프로세서는 32 bit로 이루어진 ARM 인스트럭션과 16 bit로 이루어진 Thumb 인스트럭션, 이렇게 두 가지 인스트럭션을 수행할 수 있다. 평균적으로 Thumb 인스트럭션을 사용하는 코드는 크기가 감소하지만 수행 속도가 다소 느려지는데, 코드의 크기가 감소한다는 장점 때문에, 메모리가 제한된 소형 이동단말기에 들어가는 코드는 대부분 Thumb 인스트럭션으로 컴파일되고 있다.

본 연구에서 사용한 이동단말기의 운영체제가 Thumb 인스트럭션으로 이루어져 있었으므로, 자연스럽게 응용프로그램도 Thumb 인스트럭션으로 컴파일하여 사용하였다. 물론 ARM 인스트럭션으로 이루어진 응용프로그램을 수행할 수 없는 것은 아니다. 프로세서의 인스트럭션 모드를 전환하는 코드를 응용프로그램의 시작과 끝에 추가하면 되는데, GCC의 경우 이러한 코드를 자동으로 추가하는 옵션(interwork 관련 옵션)을 가지고 있어, 본 연구진은 실제로 ARM 인스트럭션으로 이루어진 응용프로그램도 수행시킬 수 있었다.

4. 이동단말기 상에서 로더와 동적 링커의 동작

로더는 실행 파일에 나타나는 코드와 데이터를 메모리에 복사하고 코드를 수행하는데 필요한 메모리를 확보함으로써 응용프로그램을 수행할 준비를 한다. 코드를 수행하는데 필요한 메모리란 초기화 되지 않은 전역 데이터(BSS)를 위한 공간과 스택을 의미한다. 본 연구에서 사용한 이동단말기 환경은 동적 메모리 할당(malloc)/해제(free)를 지원하지 않았으므로, 힙에 대해서는 고려하지 않았다.

이동단말기를 위한 로더를 설계할 때 고려해야 할 점은, 일반적으로 이동단말기 환경은 가상 메모리를 지원하지 않는다는 것이다. 리눅스 환경에서 로더의 동작이 단순한 이유는 코드와 데이터를 적재될 메모리 영역의 (가상) 주소가 이미 정해져 있기 때문이다. 컴파일러가 생성하는 보통의 코드는 적재되는 주소가 달라지면 올바로 수행되지 않는다. 적재되는 주소가 달라지는 경우에 코드가 올바로 수행될 수 있도록, 코드에 나타나는 주소를 적절히 수정하는 작업을 재배치(relocation)라고 부르는데, 보통의 코드라면 재배치를 거쳐야 이동단말기 환경에서 올바르게 수행될 수 있다.

Position-independent-code (PIC)로 컴파일한 코드는 임의의 주소에 적재되어도 올바르게 수행될 수 있다. PIC는 분기 명령이 Program Counter (PC)에 상대적인 주소를 사용하고, 전역 데이터를 읽거나 쓸 때는 항상 Global Offset Table (GOT)를 통해 접근한다는 특성을 가진다. GOT의 사용은 PIC로 컴파일한 코드의 크기를 증가시키고, 수행 속도를 느리게하는 단점이 있지만, 재배치의 편의성 때문에 일반적으로 공유 라이브러리의 코드는 PIC로 생성되어 진다.

로더가 재배치를 하도록 할 것인지, PIC를 사용할 것인지는 수행시간과 개발시간을 고려한 선택에 달려있다. PIC의 수행 속도가 다소 느리다는 점을 지적했지만, 로더가 재배치를 하게 되면 응용프로그램의 적재 시간이 오래 걸린다. 또한 로더가 재배치를 하기 위해서는 실행 파일에 재배치 정보를 남겨 두어야 하는데, 일반적으로 정적 링커는 최종 실행 파일에 재배치 정보를 남기지 않는다. 사용하는 정적 링커가 실행 파일에 재배치 정보를 기록하는 기능을 제공하는지도 고려해서 어떤 방법으로 재배치 문제를 해결할지 결정해야 한다. 본 연구에서는 재배치의 편의성을 우선하여 PIC를 사용하기로 결정했다.

동적 링커는 공유 라이브러리를 사용하는 경우에 필요한 처리를 하는 프로그램이다. 공유 라이브러리를 사용하고 있어, 동적 링커의 도움이 필요한 응용프로그램은 실행 파일 내부에 동적 세그먼트라고 불리는 정보를 가지고 있다. 동적 세그먼트는 기본적으로 응용프로그램이 필요로 하는 공유 라이브러리의 목록을 가지고 있다. 동적 링커는 필요한 공유 라이브러리가 메모리에 적재되어 있지 않은 경우, 먼저 필요한 공유 라이브러리를 메모리에 적재해야 한다.

응용프로그램이 공유 라이브러리의 함수를 호출하는 경우, 실행 파일의 심볼 테이블에서 함수의 이름은 위치가 정의되지 않은 심볼로 나타난다. 재배치 과정에서 심볼 테이블을 사용하므로, 동적 링커는 재배치에 앞서 심볼 테이블의 정보를 완성해야 한다. 동적 세그먼트도 약간의 재배치 정보를 가지고 있다. 동적 링커가 수행하는 재배치는, 공유 라이브러리의 함수로 분기하는 주소를 기록해 주거나, GOT 항목에 코드가 적재된 영역의 주소를 더해주는 등의 일이다.

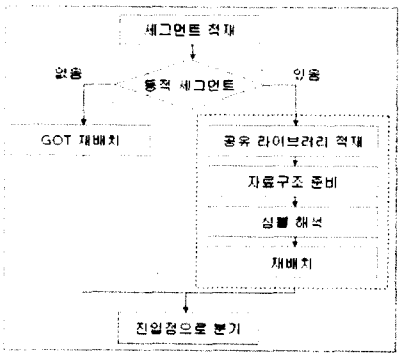
5. 이동단말기 상에서 로더와 동적 링커의 구현

PIC로 이루어진 응용프로그램을 메모리에 적재하는 방법은 비교적 간단하다. 로더는 실행 파일의 프로그램 헤더 테이블을 읽어, 메모리에 적재 가능한 세그먼트(loadable segment)들의 파일 상에서의 위치와 크기, 메모리 상에서의 크기를 파악하여

메모리를 할당하고 세그먼트를 메모리로 복사한다. 파일 상에서의 크기보다 메모리 상에서의 크기가 큰 세그먼트가 존재할 수 있는데, 초기화 되지 않은 전역 데이터 영역(BSS)이 대표적인 예이다. 구현한 로더는 응용프로그램을 적재할 때마다 메모리를 할당하는 방식으로 동작하지 않고, 미리 응용프로그램들을 위해 사용할 메모리 영역을 충분히 할당해 두고 이를 관리하는 방식으로 동작한다.

실행 파일 중에 동적 세그먼트가 존재하면, 동적 링커의 처리가 필요한 응용프로그램이라는 사실을 알 수 있다. 본 연구에서는 로더와 동적 링커를 통합하여 구현하였다. 동적 링커는 세그먼트들이 메모리에 적재된 이후의 처리를 맡는다. 동적 링커는 필요한 공유 라이브러리를 메모리에 적재하는 역할도 담당하는데, 구현한 동적 링커는 메모리에 존재하는 공유 라이브러리의 목록과 정보를 유지하면서 필요에 따라 공유 라이브러리를 적재한 후, 응용프로그램과 공유라이브러리들 간의 의존성 관계를 파악하여 순차적으로 심볼 해석과 재배치 작업을 수행한다.

<그림 1>은 시험적으로 구현한 로더 및 동적 링커의 구조를 개략적으로 나타낸 것이다. 로더 및 동적 링커를 C 언어로 구현하는 경우, 자료구조 준비 과정은 동적 세그먼트 내부의 심볼 테이블, 해시 테이블, 재배치 테이블 등을 사용하기 쉽도록 포인터를 적절히 설정하는 것으로 끝난다. GOT 재배치 과정은 보통 GOT의 각 항목에 적재된 코드의 기반 주소를 더해주는 작업이 되는데, 자세한 내용은 환경에 따라 달라질 수 있다.



<그림 1> 구현된 로더 및 동적 링커의 구조

6. 응용프로그램의 수행의 시작과 종료

이동단말기 환경에서 고려해야 하는 점 중의 하나는, 응용프로그램이 화면에 정보를 출력하거나 장치를 제어하기 위해서는 이동단말기 운영체제가 제공하는 함수를 호출할 수 있어야 한다는 점이다. 일반적으로 이동단말기의 운영체제는 응용프로그램에 의해 사용될 가능성을 전제하지 않고 구현되었지만, 이와 같은 호출을 가능하게 하는 몇 가지 방법을 생각할 수 있다.

가장 간단한 방법은 호출하려는 함수의 코드가 메모리 상에서 위치한 주소를 알아내어 분기하는 방법이다. 가상 메모리를 지원하지 않는다는 특성이 이런 방법을 가능하게 해 준다. 그리고 C 언어 프로그램의 특성을 이용해, 함수의 주소를 함수의 타입에 맞는 함수형 포인터로 변환하고 호출하면 된다. 또는 인라인 어셈블리어를 사용할 수도 있다. 물론 응용프로그램 개발자가 운영체제 내부의 함수 주소까지 알고 있어야 한다는 의미는 아니다. 설명한 구현 내용을 공유 라이브러리로 구현하여 제공하면, 응용프로그램 개발자는 정의된 인터페이스를 통해 운영체제를 호출할 수 있다. WIPI 런타임 엔진의 일부인 WIPI 라이브러리를 구현할 때 이러한 기법을 사용할 수 있다.

반면 응용프로그램의 수행을 시작하는 방법은 운영체제에서 응용프로그램의 main 함수를 호출하는 것이다.[1] 로더 또는 동적 링커는 적재한 응용프로그램의 코드가 위치한 주소를 알고 있으므로, 역시 함수 포인터를 이용하는 방법으로 응용프로그램의 함수를 호출할 수 있다. 진입점의 주소는 ELF 헤더에 정의되어 있다.

이동단말기가 ARM 기반의 프로세서를 탑재하고 있다면, 함수 포인터를 이용하는 경우에 BX 인스트럭션이 인스트럭션 모드를 변경하는 기능[9]을 가진 것에 유의하여 분기하려는 주소의 최하위비트(LSB)를 적절히 설정해 주어야 한다. 함수를 호출하는 방식으로 응용프로그램 수행을 시작했으므로, 응용프로그램 수행이 끝나면 함수를 호출한 지점으로 되돌아오며, main 함수의 반환 값도 얻을 수 있다.

7. 결론

본 연구에서 구현한 응용프로그램 로더와 동적 링커는 휴대폰과 같은 제한된 이동단말기 환경에서도 새로운 응용프로그램을 전송 받아 수행하는 것을 가능하게 해 준다. 특히 차세대 무선 응용 서비스의 기본 구조가 될 WIPI 환경은 이러한 방식의 응용프로그램 수행 방식을 전제하고 있다. 따라서 응용프로그램 로더와 동적 링커는 WIPI 런타임 엔진을 구성하는 기본적인 요소가 될 것이다. 본 연구에서는 이 원천 기술의 확보하여 관련 산업체와 공유할 것을 목적으로 응용프로그램 로더와 동적 링커를 시험적으로 구현하였다. 향후 본 연구의 결과가, WIPI 런타임 엔진 구현을 비롯하여 각종 이동단말기에 응용프로그램 수행 환경을 구현하는데, 중요한 참고 자료로 활용될 수 있을 것으로 기대한다.

참고문헌

[1] 특집 기사, *무선인터넷 플랫폼, WIPI로 날다*, 월간 마이크로 소프트웨어 2002년 10월호

[2] 한국무선인터넷표준화포럼, *모바일 표준 플랫폼 규격 v1.1*

[3] 한국무선인터넷표준화포럼, *모바일 표준 플랫폼 규격 v1.2*

[4] TIS Committee, *Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification Version 1.2*

[5] ARM, *ARM ELF Specification Issue A-08*

[6] ARM, *ARM ELF Specification Issue B-02*

[7] *Newlib*, http://sources.redhat.com/newlib/

[8] John R. Levine, *Linkers and Loaders*, Morgan Kaufmann Publishers, 2000년

[9] David Seal, *ARM Architecture Reference Manual*, Addison-Wesley, 2000년

1) 수행하려는 응용프로그램은 이동 코드를 모두 적재한 상태이다.