내장형 시스템을 위한 점진적 원격 링킹 로더

이 원영*, 우 덕균*, 표 창우**, 임 채덕***
*홍익대학교 대학원 전자계산학과
**홍익대학교 컴퓨터공학과
***한국전자통신연구원

Incremental Remote Linking Loader for Embedded Systems

Wonyoung Lee*, Dukkyun Wu*, Changwoo Pyo**, Chaedeok Lim***
*Dept. of Computer Science, Graduate School, Hongik University
**Dept. of Computer Engineering, Hongik University
***Electronics and Telecommunications Research Institute

요 약

원격 링킹 로더는 크로스 컴파일러에 의하여 컴파일된 목적 코드를 원격 타겟 시스템에 전송하여 실행할 수 있게 한다. 이 링킹 로더는 내장형 프로세서를 탑재한 시스템에서 사용되는 소프트웨어 개발 환경에 널리 써 왔다[1, 2]. 본 연구의 점진적 원격 링킹 로더는 목적 모듈을 호스트로부터 타겟으로 로딩하는 원격 로딩과 모듈이 로딩될 때 이미 로딩된 모듈들과 통합 링킹하는 점진적 링킹의 역할을 수행한다 특히, 점진적 링킹은 여러 모듈들이 로딩/언로딩될 때 사용자가 로딩/언로딩 순서를 고려해야 하는 번거로움을 해결하였다. 본 논문에서는 원격 로딩과 점진적 링킹을 가능하게 하는 로딩/언로딩 알고리즘을 제시하였다. 그리고 이 알고리즘은 [8]에서 개발된 타겟 서버 미들웨어 시스템에서 구현하였다. 본 논문에서 제시하는 점진적 원격 링킹 로더는 내장형 시스템의 개발 환경 사용에 보다 많은 사용자 편의성을 줄 것으로 기대된다.

1. 서론

내장형 시스템은 일반적으로 자체적인 프로그램 개발이 부적절한 시스템 환경을 갖는다. 따라서, 내장형 시스템 응용 프로그램 개발 환경은 호스트 컴퓨터 시스템이 내장형 시스템 즉, 타겟 시스템을 지원하도록 구성된다. 호스트는 타겟을 위해 프로그램 제작에 필요한 여러 작업을 지원해 주는 역할을 한다. 호스트는 내장형 시스템을 위한 프로그래밍 환경을 제공하고 타겟과 연결시켜주는 미들웨어를 통해 타겟과 연결된다 즉, 타겟 관리자 또는 타겟 서버라고 하는 이 미들웨어는 타겟과 관련된 심볼 테이블의 관리, 타겟의 메모리 관리, 목적 모듈의 로딩 및 언로딩을 수행하여 타겟을 관리한다[1, 2, 8].

타겟은 목적 모듈을 실행시키고 제어하는 환경을 호스트로부터 제공받아 동작한다 호스트와 타겟은 네트워크를 통해 연결되며 제어되고 모니터링 된다. 타겟은 호스트의 크로스 컴파일러를 통해 컴파일된 목적 모듈을 전송 받아 타겟 상에서 실행시킨다[2]. 호스트는 타겟 연결 미들웨어의 링킹 로더를 통해 목적 모듈을 타겟으로 전송하고, 타겟 메모리에 목적 모듈을 로딩, 언로딩 한다. 링킹 로더는 타겟에 로딩될 목적 모듈을 분석하고 목적 모듈을 구성하는 섹션별로 내용을 읽어들인다. 그리고 각 섹션들을 세그먼트 단위로 묶어 타겟 메모리로 보내는 작업을 수행한다.

링킹 로더는 여러 목적 모듈들을 로딩시켜 실행시켜야 하는 경우, 각 모듈들 상호간에 참조 관계가 있을 때 순서에 따라 로딩이 되지 않으면 정확한 실행 결과를 얻을 수 없다. 따라서, 정확한 수행 결과를 얻기 위해서는 사용자가 반드시 모듈들의 로딩 순서를 지켜야 하기 때문에 불편했다. 따라서, 본 연구는 목적 모듈들이 로딩 순서에 상관없이 로딩되어도 사용자가 아닌 링킹 로더가 이를 해결할 수 있는 방법에 대한 연구를 진행하였다. 이 방법을 점진적 링킹이라 한다[4]. 점진적 원격 링킹 로딩은 다른 시스템으로부터 모듈의 로딩이 이뤄지는 원격 로딩과 모듈들이 하나씩 로딩됨에 따라 로딩된 다른 모듈들과 링킹이 이뤄지는 점진적 링킹 방법이다. 각 목적 모듈들이 로딩 순서와 다르게 링킹 로더에 의해 로딩되어도 정확한 실행 결과를 얻을 수 있도록 하기 위한 알고리즘을 제시하였고, 구현하였다.

2절에서는 원격 링킹과 점진 링킹 개념을 보이고, 점진적 원격 링킹 로더에 대한 세부적 기능에 대해서 나타냈다. 3절에서는 로딩 순서와는 다르게 모듈이 로딩되어도 정확한 수행이 가능한 링킹 로더를 위한 데이터 구조와 알고리즘을 보였다. 4절에서는 점진적 원격 링킹 로더에 대한 구현 방법을 제시하였다. 마지막으로 5절에서는 결론 및 향후 연구 방향을 제시하였다.

2. 점진적 원격 링킹 로더

링킹 로더는 타겟에 로딩될 목적 모듈을 분석하여 필요한 정보를 저장한다. 분석된 목적 모듈의 심볼들을 처리하며, 재배치 정보와 로딩될 타겟 메모리 주소를 이용, 재배치를 수행한다. 목적 모듈은 링킹 로더에 의해 타겟 메모리로 로딩/언로딩된다.

기존의 링킹 로더는[3] 서로 참조하는 모듈들간의 순서를 지키지 않고 목적 모듈을 로딩시킬 경우에 정확한 프로그램 실행 결과를

본 논문은 1999년도 한국전자통신연구원의 "사용자 개발 도구 연구" 사업의 지원 결과이다

얻을 수 없었다. 이 문제를 해결하기 위한 전제 조건은 심볼에 대한 타겟 주소가 결정될 때까지 심볼의 재배치를 연기시켜야 한다. 이를 위해 각 모듈내에서 정의된 재배치 정보를 주소가 결정될 때 참조하기 위해 그 정보를 모듈의 심볼이 유지해야 한다. 링킹 로더는 다음과 같은 단계로 목적모듈을 로딩시킨다.

2..1 목적 모듈 분석

목적 모듈에 대한 분석 과정을 통해 목적 모듈에 대한 정보를 얻는다. 목적 모듈에 대한 정보들은 모듈이 언로딩될 때까지 유지된다. 이 정보는 모듈을 타겟에 로딩하는 과정에서 필요에 따라 사용된다. 목적 모듈 정보는 목적 모듈의 파일 헤더와 섹션 헤더를 해석함으로써 목적 모듈내의 각 섹션들의 위치를 찾게 되고 그 섹션들의 내용을 읽어옴으로써 얻는다. 목적 모듈이 갖는 내용은 목적 모듈의 섹션들에 대한 정보, 정의되는 심볼과 사용되는 심볼에 대한 정보, 목적 모듈의 텍스트 또는 데이터 섹션에서 참조하는 심볼과 관련된 재배치 정보 등이다.

2.2 타겟 메모리 할당

목적 모듈에서 실제 타겟에 로딩되는 내용이 담긴 섹션들은 실행 코드가 담긴 텍스트 섹션, 데이터 값이 저장되는 데이터 섹션, bss 섹션이 있다. 로더는 이와 같은 섹션들을 각각의 세그먼트로 합친다 세그먼트들이 로딩될 타겟 메모리내의 위치는 세그먼트들의 크기가 정해지면 결정되고, 로더에서 타겟 메모리 관리자를 호출해 사용 가능한 타겟 메모리를 할당받는다.

2.3 심볼 처리

목적 모듈에서 사용하는 심볼에 대한 정보는 목적 모듈의 심볼 테이블 섹션을 통해 얻는다. 목적 모듈에서 정의되고, 사용되는 심볼들은 링킹 로더를 통해 전역 심볼 테이블에 등록된다. 점진적 원격 링킹 로딩 방법으로 프로그램이 타겟에서 제대로 동작하도록 하기 위해서는 두 개의 심볼 테이블을 관리한다. 하나는 로딩된 모듈내에서 정의된 심볼들이 유지되는 *정의 심볼 테이블*이고, 또 하나는 모듈에서 사용하지만 로딩되지 않은 외부 모듈에서 정의된 심볼들을 유지하는 *무정의 심볼 테이블*이다. 정의 심볼 테이블의 심볼들은 타겟 메모리의 심볼에 대한 주소값이 결정된 것들이다. 무정의 심볼 테이블에 등록되는 심볼들은 아직 결정되지 않은 것이다. 즉, 다른 모듈에서 정의된 심볼의 주소가 결정될 때까지 대기하는 심볼들이다.

무정의 심볼 테이블에 등록된 심볼들은 새로 로딩되는 모듈에서 정의되어 있다면 무정의 심볼 테이블에서 삭제되고 정의 심볼 테이블에 등록된다. 각 심볼들은 특정 모듈에서 정의된 재배치 참조 정보를 가지고 있으며, 이 재배치 참조 정보들은 심볼의 타겟 메모리 주소가 결정되는 시점에서 연기되었던 재배치가 수행되기 위해 필요한 정보로 사용된다.

2.4 재배치 처리

재배치는 로딩될 재배치 모듈에서 심볼들이 가리키는 상대 주소를 수정하는 작업이다[5]. 재배치가 적용되기 전에 이미 로딩될 각 세그먼트들의 타겟 메모리 주소와 심볼들의 주소가 결정된다. 재배치 모듈의 텍스트 섹션과 데이터 섹션의 일부가 타겟 메모리로 로딩되기 전에 호스트에서 재배치가 수행된다 재배치 정보는 재배치가 적용되는 섹션 주소, 재배치와 관련되는 심볼 테이블 인덱스, 재배치 유형으로 이루어져 하나의 엔트리를 이룬다. 목적 모듈의 재배치 섹션은 이 엔트리들로 구성되었다. 재배치가 적용되는 심볼은 재배치 엔트리의 목적 모듈 심볼 테이블 인덱스가 가리키는 심볼이며, 로딩될 타겟 메모리 주소가 결정된 심볼이다. 재배치 유형에 따라 정확한 재배치 값이 결정되면, 해당하는 텍스트, 또는 데이터 세그먼트에 기록된다. 만일 로딩될 타겟 메모리 주소가 결정되지 않은 심볼

```
MT : 모듈 테이블
DefST : 정의 심볼 테이블
UndefST : 무정의 심볼 테이블
ST(m) : 목적 모듈 m 의 심볼 테이블
Rri(m, s) : 모듈 m에서 사용하는 심볼 s 의 재배치 참조 정보

loading (m : 목적 모듈)
begin
  m의 텍스트, 데이터, bss 세그먼트의 크기를 분석한다
  m의 새 세그먼트들의 타겟 메모리 공간을 할당한다.
  while s ∈ ST(m) do
    if s가 정의 심볼 then
      if s ∈ UndefST then
          UndefST := UndefST - {s}
          DefST = DefST ∪ {s}
          s의 재배치 참조 정보를 이용하여 재배치를 수행한다
      else
          DefST := DefST ∪ {s}
      fi
    else
      if s ∉ UndefST then
          UndefST := UndefST ∪ {s}
      fi
  od
  m의 재배치 정보를 이용하여 m의 재배치를 수행한다
  m의 재배치와 관련된 심볼들에 재배치 정보를 더한다.
  재배치가 수행된 m의 데이터, bss세그먼트를 타겟 메모리에 기록한다.
  MT := MT ∪ {m}
end

unloading(m : 목적 모듈)
begin
  m이 차지하는 타겟 메모리 공간을 해제시킨다.
  while s ∈ ST(m) do
    if s ∈ DefST then
          UndefST := UndefST ∪ {s}
          DefST := DefST - {s}
    else
          if Rri(s, m) ∈ {}
             UndefST := UndefST - {s}
          else
             Rri(m,s) 삭제
          fi
    fi
  od
  m에 유지되는 재배치 참조 정보를 모듈과 심볼로부터 삭제한다
  MT := MT - {m}
end
```

그림 1 로딩/언로딩 알고리즘

일 경우에는 심볼에 대한 재배치 참조 정보가 작성되고, 심볼의 타겟 메모리 주소값이 결정될 때까지 해당 심볼이 재배치 참조 정보를 유지하도록 한다.

3. 로딩 및 언로딩 알고리즘

3.1 자료구조

점진적 원격 링킹 로딩을 위해서는 다음과 같은 자료 구조가 필요하다.

- 세그먼트 : 목적 모듈을 구성하는 섹션들 중, 텍스트, 데이터, bss 섹션들의 크기와 시작 주소에 대한 정보를 갖는다.
- 모듈 : 로딩되는 목적 모듈에 대한 정보를 포함한다. 모듈의 세

```
M1                M2                M3
int a;            int b;            int c;
m1()              m2()              m3()
{                 {                 {
 a = 10;           b = 10;           c = 10;
}                  b = a + c;        c = a - c;
                  }                 }
```

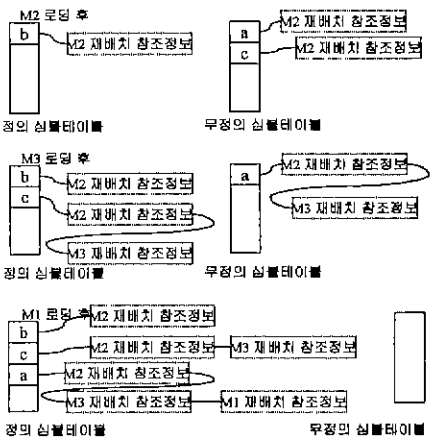

그림 2 로딩된 목적 모듈 M1, M2, M3 과 심볼 처리 과정

그먼트들의 유형과 크기, 타겟 메모리에 로딩될 시작 주소값.
- 심볼 : 목적 모듈에서 사용되는 심볼에 대한 것으로 심볼의 이름, 심볼 유형, 모듈내 심볼 테이블 섹션에서의 위치 정보, 심볼이 속한 세그먼트에 대한 정보를 갖는다.
- 모듈 테이블 : 로딩되는 모듈들로 구성된다.
- 심볼 테이블 : 로딩된 모듈들과 관련된 심볼들로 구성된다. 심볼의 주소 결정 여부에 따라 정의 심볼 테이블, 무정의 심볼 테이블로 구분된다
- 재배치 정보 : 모듈이 타겟으로 로딩될 때 필요한 재배치 정보를 갖는다. 재배치 유형, 심볼 테이블 인덱스, 참조할 가상 주소값으로 하나의 재배치 엔트리로 구성되어 있다. 재배치 엔트리가 모여 파일의 재배치 섹션을 구성한다.

또, 점진적 링킹을 위해 재배치를 심볼 주소 결정까지 보류하기 위해서 다음의 자료 구조가 추가되어야 한다.
- 재배치 참조 정보 : 재배치를 심볼 주소가 결정될 때까지 유지하기 위한 것이다. 재배치가 적용되는 심볼의 포인터, 재배치 유형, 재배치가 이뤄질 타겟의 주소, 언로딩을 위해 재배치가 적용되는 심볼을 사용하는 모듈 ID로 구성된다.

이 재배치 참조 정보는 재배치 대상이 되는 각각의 심볼과 모듈 정보에 연결되어 심볼의 주소가 결정될 때까지 재배치를 보류시킨다.

3.2 로딩/언로딩 알고리즘

[그림 1]은 점진적 원격 링킹 로딩을 위한 로딩/언로딩 알고리즘을 나타냈다 [그림 1]의 윗부분은 로딩/언로딩 알고리즘 모두에 사용되는 전역 변수와 함수들이다.

[그림 2]는 제시한 로딩 알고리즘이 제대로 동작함을 보여주는 예이다. M1, M2, M3는 목적 모듈들을 나타내고, M2, M3 ,M1의 순서대로 로딩이 된다. 순서에 따라 각 모듈의 로딩이 끝난 후 정의 심볼 테이블과 무정의 심볼 테이블에 등록되는 심볼들을 나타낸다 무정의 심볼 테이블에서 삭제되어 정의 심볼 테이블에 등록되는 동안 재배치 참조 정보를 이용, 재배치가 이루어진다

4. 구현

[그림 1]의 알고리즘을 사용하는 링킹 로더는 내장형 시스템 프로그램 개발 환경 중 호스트-타겟 연결 미들웨어의 내부 모듈에[8] 연결된다. 이 미들웨어는 심볼 테이블 관리 루틴, 타겟 메모리 관리 루틴, 링킹 로더 루틴으로 구성된다. 점진적 원격 링킹 로더 루틴을 이 미들웨어에 연결시킴으로써 ARM 프로세서가[6] 장착된 타겟시스템에 COFF 형식[7] 목적 모듈을 링킹 로딩하는 기능을 수행할 수 있다. 이 미들웨어의 심볼 테이블 관리자는 정의 심볼 테이블과 무정의 심볼 테이블의 두개의 심볼 테이블을 관리한다. 심볼들은 모듈들의 로딩/언로딩에 의하여 두개의 심볼 테이블에 등록되거나 삭제된다. 타겟에 로딩되는 COFF 목적 모듈은 호스트의 크로스 컴파일러를 통해 생성된 것이다. 로딩 순서에 구애받지 않는 원격 점진 연결 로딩을 적용하기 위해서, 위의 알고리즘에 따라 COFF 형식의 목적 모듈 분석 루틴, 목적 모듈의 심볼을 심볼 테이블에 등록하는 루틴, 재배치하는 루틴, 타겟에 전송하는 루틴을 링킹 로더의 메인 루틴이 호출하도록 구현하였다.

5. 결론

일반적인 내장형 시스템 응용 프로그램 개발 환경은 호스트-타겟 연결 미들웨어가 타겟 메모리를 관리하고 타겟에 로딩 작업을 수행한다. 이와 같은 개발 환경에서의 타겟을 위한 링킹 로더는 목적 모듈을 타겟 메모리에 로딩시킬 경우, 하나씩 목적 모듈을 로딩시켜야 하며 사용자가 목적 모듈들간의 참조 관계를 고려하여 로딩 순서를 결정해야 한다. 그러나, 사용자가 모듈간의 참조 관계를 제대로 알지 못하고 로딩시켜야 할 목적 모듈의 개수가 많을 경우에는 일일이 모듈간의 참조 관계를 고려하여 로딩 순서를 정하기는 어려운 일이다. 본 논문에서 제시한 원격 점진 링킹 로더는 사용자가 로딩시켜야 할 목적 모듈들간의 로딩 순서를 고려해야 하는 문제점을 해결하였다. 이것은 사용자에게 부담을 덜 주는 편리한 내장형 시스템 응용 프로그램 개발 환경 구축에 적용될 수 있다.

참고문헌

[1] Spectra, Mentor Graphics, http://www.mentor.com/embedded.
[2] Tornado, WindRiver Systems, http://www.wrs.com.
[3] Tornado API Guide, WindRiver Systems. 1997.
[4] R. W. Quong and M. A. Linton. "Linking programs incrementally" ACM Transactions on Programming Language and Systems, January, 1991
[5] 이기철, 표창우, "시스템 소프트웨어 이론 및 실제(5장 연결 및 적재)", 생능 출판사, 1993.
[6] D. Jaggar, "ARM Architectural Reference Manual", Prentice Hall. July, 1996.
[7] G. R. Gircys, "Understanding and Using COFF", O'Reilly & Associates, 1998.
[8] 박현수, 한경숙, 우덕균, 표창우, 김홍남, "내장 프로세서 응용 개발을 위한 타겟 연결 미들웨어 구현" 1999년 한국정보과학회 추계학술발표회 제출.