내장형 시스템을 위한 점진적이고 목표 재설정 가능한 링커

(Incremental and Retargetable Linker for Embedded System)

우 덕 균 † 한 경 숙 †† 표 창 우 ††† 김 흥 남 ††††

(Deok-Kyun Wu) (Kyungsook Han) (Changwoo Pyo) (Heung-Nam Kim)

요 약 호스트-타겟으로 연결되는 내장형 시스템 개발 환경에서 호스트의 링커는 크로스 컴파일된 목적 화일을 타겟의 모듈들과 링킹하고 타겟으로 다운로딩한다. 본 연구에서는 이와 같은 링커를 목적 화일 형식에 종속적인 모듈과 독립적인 모듈로 세분화하였다. 종속적인 모듈은 목적 화일로부터 화일 형식에 독립적인 링킹 정보를 추출하고, 독립적인 모듈은 이 링킹 정보로부터 실제적인 링킹을 담당한다. 이와 같은 세분화는 내장형 시스템 개발 환경에서 타겟 시스템에 대한 이식성을 높일 수 있다. 또한, 본 연구의 링커는 로딩되는 목적 화일 뿐만 아니라 이미 로딩된 타겟 모듈들에 대해서도 재배치를 적용하는 점진적 원격 링킹을 수행한다. 링커의 점진적 원격 링킹은 모듈 단위로 타겟으로 링킹할 수 있기 때문에 모듈들을 통합하여 타겟으로 링킹하는 방식 보다 링킹 시간을 단축할 수 있다. SPEC95 정수형 벤치마크 프로그램들에 대한 실험 결과 평균 64.90%의 감소율을 보였다. 또한, 링커의 점진적 원격 링킹은 사용자가 목적 화일들의 링킹 순서를 고려하지 않고 임의의 순서로 링킹할 수 있는 편의성을 제공할 수 있다. 현재, 본 연구의 링커는 상용화 준비 중인 내장형 응용 개발 환경 ESTO의[1] 내부 모듈로 개발되었다.

Abstract In a development environment for embedded system with a connection between host and target system, the linker of host system links the cross-compiled object file and modules of target system and downloads the linked object file to the target system. In this research, we separate this linker into the module dependent on object file format and the module independent on object file format. The dependent module gets the linking information independent on file format from the object file, and the independent module actually does the linking process with this linking information. This separation can improve the portability of development environment for a target system. Also, our linker does the incremental remote linking that applies relocation not only to the object file to be loaded but also to target's modules to have been loaded. This incremental remote linking can reduce a linking time than linking by the united modules because of linking by module. The result of measuring linking time for SPEC95 integer benchmarks shows an average of reduction rates of 64.90%. Also, incremental remote linking can improve the comfortability of users who develop programs because users do not consider a downloading order of linking object files. Currently, we developed this linker in the embedded application development environment ESTO[1] to be prepared for a commercial product.

† 정 회 원 : 한국전자통신연구원 인터넷정보가전연구부 연구원
dkwu@etri.re.kr
†† 비 회 원 : 홍익대학교 전자계산학과
khan@wow.hongik.ac.kr
††† 종신회원 : 홍익대학교 컴퓨터공학과 교수
pyo@wow.hongik.ac.kr
†††† 비 회 원 : 한국전자통신연구원 인터넷정보가전연구부 연구원
hnkim@etri.re.kr
논문접수 : 2000년 7월 21일
심사완료 : 2001년 2월 27일

1. 서 론

내장형 시스템은 일반적으로 자체적인 프로그램 개발이 부적절한 시스템 환경을 갖는다. 따라서, 내장형 시스템 응용 프로그램 개발 환경은 호스트 시스템이 내장형 시스템 즉, 타겟 시스템을 지원하도록 구성된다. 호스트는 타겟을 위해 프로그램 개발에 필요한 여러 작업을 지원해 주는 역할을 한다. 호스트는 내장형 시스템을 위한 프로그래밍, 디버깅 등의 개발 환경을 제공하고, 호스트의 미들웨어를 통해 타겟과 연결된다. 타겟 관리자(target manager) 또는 타겟 서버(target server)라고 하는 이 미들웨어는 타겟과 관련된 심볼 테이블 관리, 타겟 메모리 관리, 목적 화일의 링킹, 다운로딩 및 언로딩을 수행하여 타겟을 관리한다[2].

타겟은 목적 화일을 실행시키고 제어하는 환경을 호스트로부터 제공받아 동작한다. 호스트와 타겟은 네트워크를 통하여 연결되며 제어되고 모니터링 된다. 타겟은 호스트의 크로스 컴파일된 목적 화일을 전송 받아 타겟 상에서 실행시킨다. 호스트는 타겟 연결 미들웨어의 링커를 통하여 목적 화일을 타겟으로 다운로딩한다. 링커는 목적 화일을 타겟으로 다운로딩하기 전에, 이미 타겟에 로딩된 모듈과의 링킹을 수행한다.

호스트의 목적 화일을 타겟으로 링킹, 다운로딩하는 본 연구의 링커는 목적 화일 형식에 종속적인 부분과 독립적인 부분으로 세분화하여 설계하였다. 종속적인 부분은 목적 화일을 읽어 화일 형식에 독립적이고 링킹에 필요한 최소한의 링킹 정보를 추출하는 기능을 담당한다. 독립적인 부분은 추출된 링킹 정보로부터 실제적인 링킹 기능을 담당한다. 타겟 시스템이 변경되면 링커의 독립적인 부분은 변경되지 않고, 종속적인 부분만 변경되므로 링커의 개발 시간을 단축시킬 수 있으며, 궁극적으로 전체 개발 환경의 타겟 시스템에 대한 이식성을 높일 수 있다.

링커가 여러 목적 화일들을 다운로딩시켜 실행시켜야 하는 경우, 각 화일들 상호간에 참조 관계가 있을 때 순서에 따라 다운로딩이 되지 않으면 정확한 결과를 얻을 수 없었다[2]. 따라서, 정확한 수행 결과를 얻기 위해서는 사용자가 반드시 목적 화일들의 다운로딩 순서를 지키거나, 호스트에서 목적 화일들을 하나로 통합하여, 통합된 목적 화일을 타겟으로 다운로딩해야 했다. 사용자가 모듈들의 다운로딩 순서를 고려해야 한다는 것은 사용자에게 불편함을 주고, 여러 목적 화일들을 하나의 화일로 통합하여 다운로딩하는 것은 프로그램 개발에서의 링킹 시간을 증가시킨다. 이러한 문제점을 해결하기 위하여 본 연구에서는 점진적 원격 링킹(incremental remote linking)을 수행하는 링커를 개발하였다. 점진적 원격 링킹은 호스트에서 타겟 모듈들을 링킹하는 원격 링킹과 목적 화일들이 하나씩 로딩되면서 링킹이 이루어지는 점진적 링킹을 말한다.

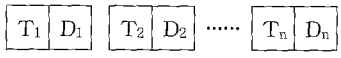
목적 화일 형식에 종속적, 독립적으로 세분화되고 점진적 원격 링킹을 수행하는 본 논문의 링커는 실제 상용화 준비에 있는 ESTO 개발 환경의[1] 내부 모듈로 구현되었다. 현재, ESTO 개발 환경은 StrongARM-110 프로세서와[3] QPlus 실시간 운영체제를[4] 탑재한 타겟 시스템을 지원하고 있으며, 크로스 컴파일러는 gcc를 사용한다. 본 논문의 링커는 gcc에서 생성되는 COFF[5] 또는 ELF[6] 형식의 목적 화일들에 대해서 링킹, 다운로딩을 수행한다.

본 논문의 구성은 다음과 같다. 2절에서는 본 논문의 링커와 다른 링커들을 비교하였다. 3절에서는 내장형 시스템 개발 환경과 그 환경에서 호스트-타겟을 연결 시켜주는 미들웨어인 타겟 관리자를 소개하였다. 4절에서는 타겟 관리자의 핵심 모듈로서 목적 화일 형식에 종속적, 독립적으로 세분화되어 점진적 원격 링킹을 수행하는 링커를 설명하였다. 5절에서는 링커의 실제 구현 내용을 소개하였고, 점진적 원격 링킹의 개발 시간 단축에 대한 실험 결과를 제시하였다. 마지막으로 6절에서 결론 및 향후 연구 방향을 제시하였다.

2. 관련 연구

호스트-타겟으로 연결되는 내장형 응용 프로그램 개발 환경인 Tornado의[7] 링커는 내장형 응용 프로그램 개발 환경의 링커인 점에서 본 연구와 유사하다. 그러나, Tornado의 링커는 점진적 원격 링킹을 지원하지 않는다. 모듈이 타겟으로 링킹, 다운로딩될 때, 참조 심볼의 주소 결정(resolution)은 현재 다운로딩되는 모듈에 대해서만 적용된다. 이것은 심볼이 정의되는 모듈이 심볼이 참조되는 모듈보다 먼저 다운로딩되어야 하는 제약 조건을 가진다. 그러나, 모듈들이 교차-참조(cross-reference)되는 경우에는 모듈들을 한 덩어리로 통합하여 통합된 모듈을 타겟으로 링킹, 다운로딩해야 한다. 이것은 호스트에서의 불필요한 링킹과 호스트-타겟의 통신 과부하로 인한 링킹, 다운로딩 시간을 증가시킨다. 본 연구의 링커는 현재 다운로딩되는 모듈에서 정의되는 심볼을 참조하는 다른 모듈의 심볼에 대하여 주소 결정(resolution)을 적용한다. 본 연구의 이와 같은 점진적 원격 링킹은 모듈의 다운로딩 순서에 대한 제약도 없고, 교차-참조의 경우에도 모듈 단위로 링킹, 다운로딩할 수 있기 때문에 링킹과 호스트-타겟의 통신 과부

(a) Quong의 링커



(b) 본 연구 링커

그림 1 Quong의 링커와 본 연구 링커의 모듈 배치(T_i : 모듈 i의 텍스트 세그먼트, D_i : 모듈 i의 데이타 세그먼트)

하를 줄여 링킹, 다운로딩 시간을 줄인다.

Tornado의 링커는 타겟/형식(타겟 프로세서 또는 목적 화일 형식)에 종속적이다. 그러나, 본 연구의 링커는 타겟/형식에 독립적인 부분과 종속적인 부분으로 분리된다. 링커가 새로운 타겟 시스템을 지원할 때, 본 연구의 링커는 타겟/형식에 종속적인 부분만 변경되면 되나, Tornado의 링커는 링커 전체가 변경되어야 한다. 즉, 본 연구의 링커는 Tornado의 링커보다 새로운 타겟 시스템에 대한 이식성을 증가시켰다.

Simon은 호스트와 타겟이 동일한 단일 시스템 개발 환경의 링커와 내장형 시스템 개발 환경의 링커에 대한 차이점을 제시하였다[8]. 단일 시스템 개발 환경의 링커는 모듈들을 링킹하여 실행 화일을 생성하고, 실행 화일의 메모리 로딩과 실행은 로더(loader)가 담당한다. 내장형 시스템 개발 환경의 링커는 모듈들을 링킹하여 타겟 메모리로 다운로딩한다. 내장형 시스템 개발 환경의 링커는 단일 시스템 개발 환경의 로더의 메모리 로딩 기능도 담당한다. 또한, 내장형 시스템 개발 환경의 링커는 이미 타겟으로 다운로딩된 모듈에 대한 타겟 메모리 주소 정보도 유지하고 있는 점이 단일 시스템 개발 환경의 링커와 다르다.

Quong은 단일 시스템 개발 환경의 링커에서 점진적 링킹을 통한 링킹 시간 감소 효과를 제시하였다[9]. Quong의 링커는 응용 프로그램의 모듈들을 완전히 링킹하여 하나의 실행 파일로 만들며, 모듈의 각 세그먼트는 그림 1(a)와 같이 배치된다. 모듈이 수정되어 다시 링킹될 때, 수정된 모듈을 기존 모듈이 위치했던 자리에 배치하고, 수정된 모듈과 수정된 모듈을 참조하는 다른 모듈들에 대하여 재배치를 적용한다. 수정된 모듈이 기존 모듈의 공간에 들어갈 수 없을 때, 모듈 전체에 대하여 다시 링킹한다. 모듈이 수정되어 다시 링킹될 때, 다른 모듈의 수정을 최소화하기 위하여 그림 1(a)와 같이 각 세그먼트에 대하여 여유 공간을 두어 할당하였다. 본 연구의 링커는 실행 화일 이미지를 만드는 것이 아니라, 응용 프로그램의 모듈들을 모듈 단위로 링킹, 다운로딩하여, 타겟 메모리에 그림 1(b)와 같이 배치한다. 그리고, 응용 프로그램의 모듈 일부만 타겟으로 링킹, 다운로딩되어 테스팅될 수 있다. 모듈이 수정되는 경우에는, 수정된 모듈에 대하여 재배치를 적용하고, 타겟으로 다운로딩한다. 그리고, 호스트에서 수정된 모듈을 참조하는 다른 타겟 모듈들에 대하여 재배치를 적용한다.

GNU 개발 환경의[10] 링커는 단일 호스트 시스템에서 재목적성을 갖는 링커이다. GNU 개발 환경의 링커는 타겟/형식에 종속적인 부분과 독립적인 부분으로 분리되어 설계되었다. 종속적인 부분은 BFD(Binary File Descriptor) 라이브러리로[11] 구축되었다. 그러나 BFD 라이브러리는 링커에 국한된 라이브러리가 아니라 GNU 개발 환경의 범용적인 라이브러리로서 본 연구에서 유지하는 링킹 정보보다 더 많은 정보를 유지한다. 이와 같은 정보 처리에 대한 과부하는 결국 링킹 시간을 증가시킬 수 있다.

Ung은 목적 화일 형식의 명세로부터 자동으로 로더를 생성하는 재목적 로더(retargetable loader)에 대하여 연구하였다[12]. 이 연구는 본 연구의 재목적 링커보다는 발전된 형태이지만, 링커가 아닌 로더에 대한 연구이다. 그리고 내장형 시스템 개발 환경이 아닌 단일 시스템 개발 환경 기반인 점이 본 연구와 다르다.

3. 내장형 응용 개발 환경

내장형 시스템의 응용 프로그램 개발은 그림 2와 같이 내장형 시스템보다 성능이 우수한 호스트 시스템에서 수행된다. 호스트 시스템이 목표로 하는 내장형 시스템을 타겟 시스템이라 부른다. 호스트 시스템과 타겟 시스템은 네트워크를 통하여 연결되며, 호스트 시스템은 타겟 시스템을 관리하는 타겟 관리자(target manager)와 프로그램 개발에 필요한 크로스 컴파일러, 디버거, 쉘, 자원 모니터 등의 개발 도구들을 포함한다. 개발 도구들은 타겟 관리자를 통하여 타겟 시스템과 연결된다.

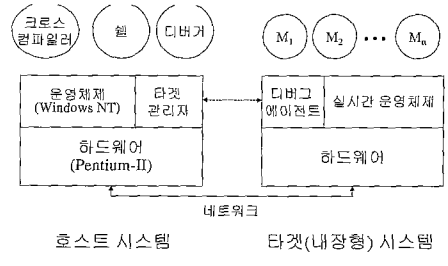


그림 2 내장형 응용 개발 환경

타겟 시스템은 호스트의 타겟 관리자와 연결되는 디버그 에이전트, 실시간 운영체제, 응용 프로그램 모듈들로 구성된다. 타겟 시스템의 독립 응용 프로그램인 디버그 에이전트는 호스트의 타겟 관리자와 연결하여, 호스트의 개발 도구들 또는 타겟 관리자에게 프로그램 개발에 필요한 타겟 서비스를 제공한다[13].

호스트 시스템의 타겟 관리자는 그림 3과 같이 도구들과 연결되는 전단부, 타겟과 연결되는 후단부, 그리고 내부 모듈들로 구성된다. 내부 모듈들은 모듈 로딩에서

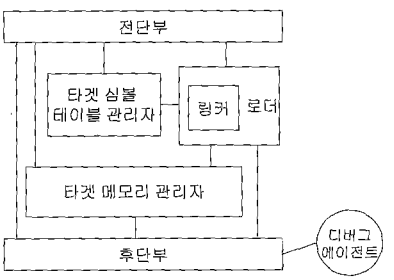


그림 3 타겟 관리자

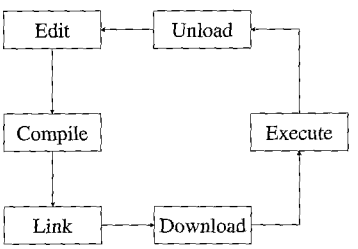
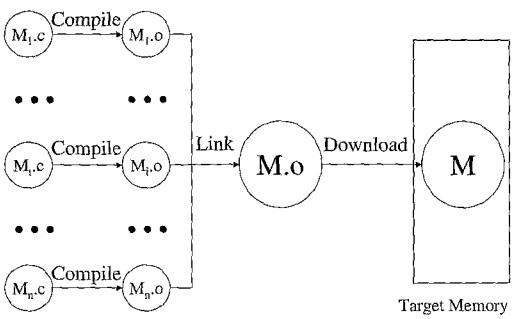


그림 4 내장형 응용 개발 주기

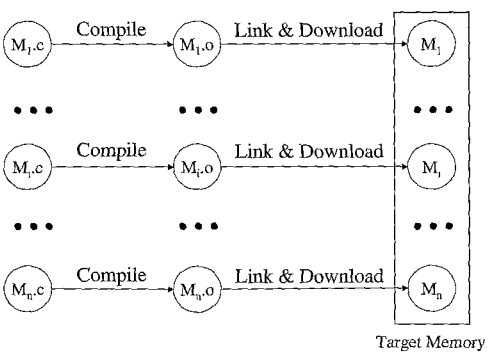
사용할 타겟 메모리를 관리하는 타겟 메모리 관리자(target memory manager), 로딩된 모듈에서 정의 또는 사용되는 심볼을 관리하는 타겟 심볼 테이블 관리자(target symbol table manager), 호스트에서 크로스 컴파일된 목적 화일을 타겟으로 다운로딩 또는 타겟에 다운로딩된 모듈들을 언로딩하는 로더(loader)로 구성된다. 타겟 관리자의 로더는 링커를 포함하며, 링커는 호스트에서 크로스 컴파일된 목적 화일을 분석하여 타겟에 다운로딩된 모듈들과의 링킹을 수행한 후, 타겟에 다운로딩하는 역할을 담당한다.

그림 2와 같은 내장형 응용 개발 환경에서 응용 프로그램의 개발은 그림 4와 같은 개발 주기를 갖는다. 개발자는 호스트에서 소스 프로그램을 편집(edit)하고, 편집된 소스 화일을 크로스 컴파일하여 목적 화일을 만든다. 목적 화일은 타겟 관리자의 링커를 통하여 호스트에서 링킹된 후, 타겟으로 다운로딩된다. 타겟의 다운로딩된 모듈은 타겟에서 실행되어 테스트된다. 테스트 결과 오류가 발생한 경우에는 다운로딩된 모듈을 타겟으로부터 언로딩하고, 소스 프로그램을 수정하여 다시 컴파일한다. 이와 같은 주기로 반복적으로 수행되며, 최종적으로 응용 프로그램이 완성된다.

응용 프로그램은 일반적으로 여러 개의 소스 화일로 구성되므로, 내장형 응용 개발 주기에서 여러 목적 파일



(a) 통합 모듈 링킹



(b) 단위 모듈 링킹

그림 5 모듈들의 링크/다운로드

의 타겟으로의 링킹과 다운로딩은 그림 5와 같은 두 경우로 살펴볼 수 있다. 그림 5(a)의 통합 모듈 링킹은 호스트에서 여러 목적 화일을 하나의 목적 화일로 링킹하여 타겟으로 링킹, 다운로딩하는 경우를 말한다. 그림 5(b)의 단위 모듈 링킹은 호스트의 여러 목적 화일을 모듈 단위로 타겟으로 링킹, 다운로딩하는 경우를 말한다.

통합 모듈 링킹은 단일 호스트 시스템의 개발 환경에

서 사용되는 방식이다. 그러나 호스트-타겟으로 연결되는 개발 환경에서 이와 같은 링킹 방식은 모듈들이 통합되어 다운로딩되기 때문에, 통신 과부하를 유발하여, 단위 모듈 링킹보다 다운로딩 시간을 증가시킨다. 이와 같은 문제점을 해결하는 단위 모듈 링킹은 모듈 단위로 다운로딩 되기 때문에 링킹 수와 통신 과부하를 줄여 다운로딩 시간을 감소시킨다. 그러나 이와 같은 링킹 방식에서는 개발자가 모듈들의 상호 참조 관계를 고려하여 모듈들의 링킹, 다운로딩 순서를 고려해야 했다[2].

응용 프로그램 개발 중에 모듈들의 다운로딩 순서를 고려해야 하는 것은 개발자에게 프로그램 개발의 불편함을 가져온다. 본 연구의 링커는 모듈들을 원격지에서 점진적으로 링킹함으로써, 모듈들의 다운로딩 순서 고려에 대한 불편함을 해소 시켰다.

4. 링커

4.1 목표 재설정 가능한 링킹

호스트-타겟으로 연결되는 내장형 시스템 개발 환경에 포함되는 링커는 호스트에서 크로스 컴파일된 목적 화일을 타겟의 모듈들과 링킹, 다운로딩하는 기능을 담당한다. 내장형 시스템 개발 환경에서의 링커는 목적 화일 형식에 종속되므로, 타겟 시스템 또는 크로스 컴파일러가 변경되면 그에 따라 링커도 변경되어야 한다. 이와 같은 링커 변경을 최소화하기 위하여 본 연구에서는 링커를 목적 화일 형식에 종속적인 부분과 독립적인 부분으로 구분하였다.

그림 6은 목적 화일 형식에 종속적인 리더(reader) 모듈과 독립적인 링커 모듈로 구성되는 본 연구의 링커를 나타낸다. COFF 리더는 COFF 목적 파일 형식에[5] 종속적인 리더 모듈을 나타내고, ELF 리더는 ELF 목적 파일 형식에[6] 종속적인 리더 모듈을 말한다. *리더 모듈은 목적 화일 형식에 종속되어* 목적 화일을 분석하여 목적 화일 형식에 독립적인 링킹 정보를 링커 모듈에게 제공한다. 링킹 정보를 바탕으로 링커 모듈은 목적 화일 형식에 독립적으로 링킹, 다운로딩을 수행한다.

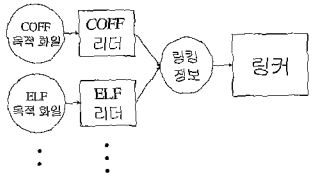


그림 6 링커 구성도

표 1 목적 화일 형식에 독립적인 링킹 정보

(a) 섹션 정보	(b) 심볼 정보	(c) 재배치 정보
	심볼 타입	재배치 타입
섹션 타입	심볼 이름	재배치 적용 섹션
섹션 위치	정의 섹션	재배치 위치
섹션 크기	심볼 위치	재배치 심볼

링킹 정보는 표 1과 같이 섹션 정보, 심볼 정보, 재배치 정보로 구성된다. 섹션 정보는 타겟 메모리 영역을 차지하는 텍스트, 데이타, bss 섹션들의 정보를 나타낸다. 섹션 타입은 텍스트, 데이타, bss 섹션들을 구분해주고, 섹션 위치는 목적 화일 내에서 해당 섹션의 옵셋 값을 말하고, 섹션 크기는 섹션의 크기 정보를 말한다. 목적 화일의 텍스트, 데이타 섹션은 섹션 위치와 크기를 바탕으로 호스트에서 타겟으로 다운로딩되고, 초기화되지 않은 데이타 값을 갖는 bss 섹션은 크기 정보를 바탕으로 타겟 메모리 영역만 할당된다.

심볼 정보는 목적 화일에서 정의되는 정의 심볼과 외부에서 정의되고 목적 파일 내에서 참조되는 미정의 심볼에 대한 정보를 유지한다. 심볼 타입은 정의 심볼, 미정의 심볼을 구분하여 주고, 정의 섹션은 그 심볼이 정의되는 섹션을 가리키며, 섹션 정보의 인덱스 값을 갖는다. 미정의 심볼의 경우는 0 값을 갖는다. 심볼 위치는 심볼이 정의되는 섹션 내 옵셋 값을 말한다. 미정의 심볼의 경우는 0 값을 갖는다.

재배치 정보는 텍스트, 데이타 섹션에 대하여 재배치를 적용할 때 사용되는 정보를 유지한다. 재배치 타입은 재배치가 적용되는 규칙, 예를 들면 재배치 계산 규칙, 재배치 적용 비트 수 등의 정보를 내포하는 정수 값을 갖는다. 재배치 적용 섹션은 재배치가 적용되는 섹션을 가리키며, 섹션 정보의 인덱스 값을 갖는다. 재배치 위치는 재배치가 적용되는 부분의 섹션내 옵셋 값을 말한다. 재배치 심볼은 재배치와 관련된 심볼을 가리키며, 심볼 정보의 인덱스 값을 갖는다.

그림 7은 두 C 프로그램의 링킹을 보여준다. add.c 화일에는 두 수를 더하는 add 함수와 그 결과 값이 저장되는 전역 변수 sum이 정의되고, calc.c 화일에는 add 함수와 sum 변수를 참조하여 10과 20의 합을 구하는 calc 함수가 정의된다. add.c로부터 크로스 컴파일된 add.o 목적 화일은 add 함수로 구성되는 텍스트 섹션과 sum 변수로 구성되는 데이타 섹션을 포함하며, calc.c로부터 크로스 컴파일된 calc.o 목적 화일은 calc 함수로 구성되는 텍스트 섹션을 포함한다. add 함수는 add.o

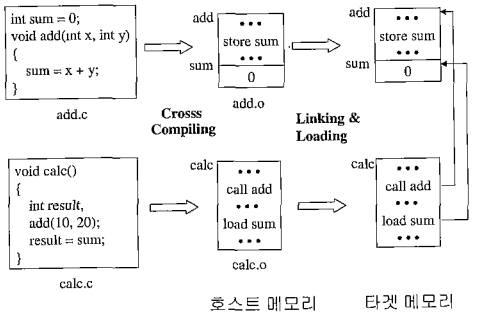


그림 7 링킹 예

목적 화일에서 정의되는 sum 변수를 참조하며, calc 함수는 add.o 화일에서 정의되는 add 함수와 sum 변수를 참조한다. 두 목적 화일의 텍스트, 데이타 섹션들은 타겟 메모리로 다운로딩되고, add 함수, sum 변수의 참조에 대한 재배치가 이루어진다. 보다 자세한 링킹 알고리즘은 다음 절에서 소개된다.

본 연구의 링커는 두 목적 화일 add.o, calc.o를 링킹하기 위하여 리더 모듈은 링킹에 필요한 링킹 정보를 분석한다. 목적 화일이 COFF 형식인 경우에는 COFF 리더가 연결되어 링킹 정보가 분석되고, ELF 형식인 경우에는 ELF 리더가 연결되어 링킹 정보가 분석된다. 분석된 링킹 정보는 COFF, ELF 형식에 관계없이 단일한 형식의 정보를 포함한다. 이와 같이 분석된 링킹 정보는 표 2에서 제시된다. 각 표들의 두 번째 행은 각 링킹 정보들의 인덱스를 나타낸다. 섹션 정보에서 섹션 타입은 텍스트, 데이타, bss로 구분되고, 심볼 정보에서 심볼 타입은 정의, 미정의로 구분된다. 재배치 정보에서 재배치 타입 ABS32와 PC24는 ARM 프로세서를 목표로 하는 ELF 목적 화일 형식에서 사용되는 재배치 타입의 한 예이다[14]. 재배치 타입 ABS32는 재배치 적용될 부분의 32비트를 심볼 주소 값으로 대체하는 것을 말하며, PC24는 재배치 적용될 부분 명령의 32비트 부분에서 24비트 부분을 심볼 주소를 PC 주소의 상대 값으로 계산하여 대체하는 것을 의미한다.

4.2 점진적 원격 링킹

본 연구의 링커는 호스트에서 크로스 컴파일된 목적 화일들을 각각 하나씩 호스트에서 타겟으로 점진적으로 링킹, 다운로딩한다. 예를 들면, 그림 7에서와 같이, 두 목적 화일 add.o, calc.o는 호스트에서 하나의 목적 화일로 합쳐져 타겟으로 링킹되지 않고, 각각 독립적으로 타겟으로 링킹된다. 또한, 목적 화일들의 링킹은 임의의 순서로 진행될 수 있다. 이와 같이 링킹은 원격지에 있는 모듈들과 점진적으로 진행되므로, 본 논문에서는 이와 같은 링킹 방식을 점진적 원격 링킹(*incremental remote linking*)이라 부른다.

목적 화일의 점진적 원격 링킹 과정은 그림 8에서 제시된다. T_i는 이미 타겟으로 다운로딩된 i 번째 모듈, 또는 타겟으로 링킹될 호스트의 i 번째 목적 화일의 텍스트 섹션을 나타낸다.

D_i는 i 번째 데이타 섹션을 말하고, B_i는 i 번째 bss 섹션을 말한다. 그림 8의 좌측 원은 호스트에서 크로스 컴파일되어 타겟으로 링킹되는 n 번째 목적 화일을 나타내고, 상단 직사각형은 그 목적 화일로부터 분석된 링

표 2 링킹 정보 예

(a) 섹션 정보

	add.o		calc.o
인덱스	0	1	0
섹션 타입	텍스트	데이타	텍스트
섹션 위치	52	108	52
섹션 크기	56	4	52

(b) 심볼 정보

	add.o		calc.o		
인덱스	0	1	0	1	2
심볼 타입	정의	정의	정의	미정의	미정의
심볼 이름	"sum"	"add"	"calc"	"add"	"sum"
정의 섹션	1	0	0	0	0
심볼 위치	0	0	0	0	0

(c) 재배치 정보

	add.o	calc.o	
인덱스	0	0	1
재배치 타입	ABS32	PC24	ABS32
재배치 적용 섹션	1	0	0
재배치 위치	8	24	36
재배치 심볼	0	1	2

킹 정보를 말한다. 하단 직사각형은 이미 다운로딩된 모듈들에서 정의 또는 참조되는 심볼들이 유지되는 타겟 심볼 테이블을 나타낸다. 타겟 심볼 테이블의 두 원은 임의의 심볼을 나타내고, Dsym은 정의 심볼을 말하고, Usym은 미정의 심볼을 말한다. 우측 직사각형은 타겟 메모리를 나타내며, 현재 n-1개의 모듈들이 타겟 메모

리에 위치하고 점선으로 표시된 직사각형은 n 번째 모듈이 저장될 공간을 나타낸다. 각 원들과 직사각형들 사이의 화살표는 링킹이 진행되는 방향을 나타내고, 그 위의 숫자는 링킹이 진행되는 순서를 나타낸다.

n 번째 목적 화일에 대한 타겟으로의 링킹, 다운로딩 과정은 다음과 같다.

1. 목적 화일로부터 섹션 정보, 심볼 정보, 재배치 정보 등의 링킹 정보를 분석한다. (그림 8의 (1))
2. 섹션 정보의 섹션 타입과 섹션 크기를 바탕으로 텍스트 섹션 T_n, 데이타 섹션 D_n, bss 섹션 B_n들이 저장될 타겟 메모리 공간을 할당하며, 각 섹션들의 타겟 메모리 주소를 구한다. (그림 8의 (2))

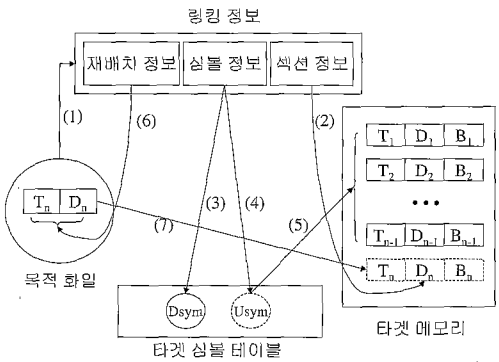


그림 8 점진적 원격 링킹 흐름도

3. 심볼 정보의 각 엔트리들에 대하여,
   (a) 엔트리가 정의 심볼이고 심볼 테이블에 존재하지 않으면, (그림 8의 (3))
   -새로운 심볼을 생성한다.
   -생성된 심볼에 심볼 이름, 타겟 메모리 주소 등을 더한다.
   -심볼을 심볼 테이블에 등록한다.
   (b) 엔트리가 정의 심볼이고 심볼 테이블에 미정의 심볼로 존재하면, (그림 8의 (4), (5))
   -미정의 심볼을 정의 심볼로 변환한다.
   -심볼에 타겟 메모리 주소를 더한다.
   -미정의 심볼이 유지하는 재배치 정보를 이용하여, 타겟의 모듈들에 대하여 재배치를 적용한다.
4. 재배치 정보의 각 엔트리들에 대하여,
   (a) 재배치와 관련된 심볼을 타겟 심볼 테이블에서 가져온다.
   (b) 해당 심볼이 정의 심볼이면 심볼의 타겟 메모리 주소와 엔트리의 재배치 정보를 바탕으로 호스트에서 목적 화일의 텍스트, 데이타 섹션들에 대해서 재배치를 수행한다. (그림 8의 (6))
   (c) 미정의 심볼이면 재배치 정보를 심볼에 더한다.
5. 재배치된 목적 화일의 텍스트, 데이타 섹션을 타겟으로 보낸다. (그림 8의 (7))

그림 7의 링킹 예에서, calc.o 목적 화일이 타겟에 다운로딩된 상태에서 add.o 목적 화일에 대한 점진적 원격 링킹 과정을 살펴보자. 그 내용은 그림 9에서 제시된다.

1. add.o 목적 화일로부터 링킹 정보를 구한다. (그림 9의 (1)) 링킹 정보는 표 2에서 제시된다.

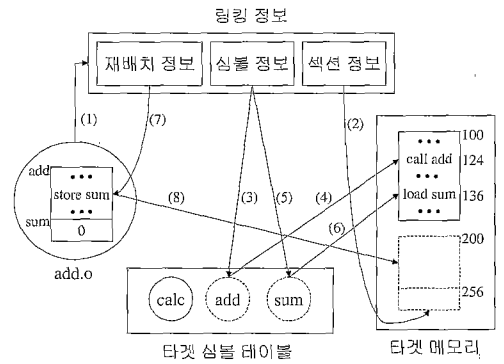


그림 9 add.o 목적 화일의 점진적 원격 링킹

2. add.o 목적 화일의 섹션 정보로부터, 텍스트 섹션 56 바이트와 데이타 섹션 4바이트에 대한 타겟 메모리 공간을 할당한다. 이 때, 각 섹션들의 타겟 메모리 주소 200, 256을 구한다. (그림 9의 (2))
3. 심볼 정보의 각 엔트리 add, sum들에 대하여
   (a) add 심볼에 대하여,
   -심볼 테이블의 미정의 심볼 add를 정의 심볼로 변환한다. (그림 9의 (3))
   -add 심볼의 타겟 메모리 주소 200과 add 심볼의 재배치 정보로부터 124번지의 add 심볼 참조 부분에 대하여 재배치를 적용한다. (그림 9의 (4))
   (b) sum 심볼에 대하여,
   -심볼 테이블의 미정의 심볼 sum을 정의 심볼로 변환한다. (그림 9의 (5))
   -sum 심볼의 타겟 메모리 주소 256과 add 심볼의 재배치 정보로부터 136번지의 sum 심볼 참조 부분에 대하여 재배치를 적용한다. (그림 9의 (6))
4. 링킹 정보의 재배치 정보와 sum 심볼의 타겟 메모리 주소로부터 add.o 목적 화일의 sum 참조 부분에 대하여 재배치를 적용한다. (그림 9의 (7))

5. 재배치된 add.o 목적 화일의 텍스트 섹션과 데이타 섹션을 타겟 메모리에 저장한다.

본 연구의 점진적 원격 링킹은 목적 화일들의 타겟으로의 링킹 순서에 상관 없이 타겟으로 링킹된 모듈들은 같은 실행 이미지를 유지하게 한다. 따라서 응용 프로그램 개발자는 프로그램 개발 과정에서 프로그램을 구성하는 여러 목적 화일들을 타겟으로 링킹할 때, 목적 화일들의 링킹 순서를 고려하지 않아도 된다. 이것은 응용 프로그램 개발자에게 프로그램 개발에 대한 편의성을 제공한다.

프로그램 개발자는 응용 프로그램을 구성하는 여러 목적 화일들을 모듈 단위로 링킹하지 않고, 호스트에서 하나의 이미지로 통합하여 타겟으로 링킹할 수 있다. 그러나, 프로그램 개발 중에 프로그램이 수정되면 수정된 목적 화일 만 타겟으로 링킹하는 것이 아니라, 전체 목적 화일들의 통합된 이미지가 다시 타겟으로 링킹되어야 한다. 이것은 모듈 단위로 타겟으로 링킹하는 것보다 링킹, 다운로딩 시간을 증가시킨다. 궁극적으로 본 연구의 점진적 원격 링킹은 응용 프로그램 개발 시간 측면에서도 보다 효과적일 수 있다. 이것에 대한 내용은 다음 절에서 실험을 통하여 제시된다.

5. 구현 및 실험

5.1 구현

본 논문의 리더와 링커는 ESTO 개발 환경의[1] 타겟 관리자 내부 모듈로 개발되었다. ESTO 개발 환경은 내장형 시스템의 응용 프로그램 개발을 지원하는 개발 환경으로, 호스트 시스템은 Windows NT의 PC 환경이고, 타겟 시스템은 Strong ARM-110 프로세서와[3] Qplus 실시간 운영체제[4] 기반이다. 호스트 시스템에서의 크로스 컴파일러는 gcc를 사용한다. gcc 크로스 컴파일러는 COFF 형식의[5] 목적 화일을 생성하는 버전과 ELF 형식의[6] 목적 화일을 생성하는 버전 두 가지가 지원된다. ESTO의 타겟 관리자는 타겟의 디버거 에이전트와 연결하여 타겟을 관리하고, 호스트의 디버거, 자원 모니터, 쉘 등의 개발 도구들에게 타겟에 대한 서비스를 제공한다.

ESTO의 타겟 관리자는 목적 화일의 다운로딩과 언로딩을 담당하는 로더 모듈이 포함된다.
로더 모듈은 목적 화일의 링킹, 다운로딩을 담당하는 링커와 링킹 정보를 분석하는 리더 모듈을 포함한다. 본 연구에서는 COFF 형식의 목적 화일로부터 링킹 정보를 구하는 COFF 리더와 ELF 형식의 목적 화일로부터 링킹 정보를 구하는 ELF 리더를 개발하였다. 또한, 리더로부터의 목적 화일 형식에 독립적인 링킹 정보로부터 점진적 원격 링킹을 수행하는 링커를 개발하였다.

COFF와 ELF 리더는 동적 링킹 라이브러리(dynamic linking library)로서 링커와 연결된다. 타겟으로 로딩할 목적 화일이 COFF 형식이면 COFF 동적 링킹 라이브러리가 연결되어, COFF 화일을 분석하여 링커에게 링킹 정보를 전달하고, 링커는 링킹을 수행한다. 다음에 ELF 형식의 목적 화일이 로딩되면 ELF 동적 링킹 라이브러리가 연결되어, ELF 화일을 분석하여 링커에게 링킹 정보를 전달한다. 전달 받은 링킹 정보에 의하여 링커는 점진적 원격 링킹을 수행한다. 이와 같이 서로 다른 형식의 목적 화일들이 링킹될 때, 타겟 관리자는 전혀 영향을 받지 않으며 수행된다.

5.2 실험

실험은 점진적 원격 링킹의 효과를 살펴보기 위하여 그림 5의 통합 모듈 링킹과 단위 모듈 링킹에 대하여 컴파일, 링킹 시간을 측정하여 비교하였다. 실험은 ESTO 개발 환경에서 진행되었으며, 타겟 운영체제는 Qplus를 사용하였다. 컴파일 옵션에는 디버깅 옵션 '-g'를 포함하였다. 호스트 시스템은 운영체제가 Windows NT 4.0이고, 프로세서는 Pentium-II 350MHz이며, 램 용량은 128MB이다. 벤치마크 프로그램으로는 SPEC95 정수형 벤치마크 프로그램에서[15] gcc, go, li, m88ksim, vortex, perl 프로그램들을 사용하였다.

본 실험에서는, 그림 10에서와 같이 n개의 C 프로그램으로 구성되는 응용 프로그램에서 하나의 모듈이 수정되어 타겟으로 다시 링킹되는 경우에, 두 경우의 링킹에 대한 컴파일, 링킹 시간을 측정하였다. 링킹 시간은 타겟으로의 다운로딩 시간까지 포함한다. 그림 5(a)의 통합 모듈 링킹에서는 하나의 소스 화일이 수정될 때, 소스 화일의 컴파일 시간, 다른 목적 화일들과의 호스트에서의 링킹 시간, 통합 목적 화일의 타겟으로의 링킹 시간 합이 계산된다. 그림 5(b)의 단위 모듈 링킹에서는 그 소스 화일의 컴파일 시간과 컴파일된 목적 화일의 타겟으로의 링킹 시간의 합이 계산된다. 그림 10은 통합 모듈 링킹에 대한 단위 모듈 링킹의 컴파일, 링킹 시간의 감소율을 나타내며 다음과 같이 계산된다.

$$\frac{\left(\begin{array}{c}\text{통합 모듈 링킹의 컴}\\ \text{파일, 링킹 시간}\end{array}\right)-\left(\begin{array}{c}\text{단위 모듈 링킹의}\\ \text{컴파일, 링킹 시간}\end{array}\right)}{(\text{통합 모듈 링킹의 컴파일, 링킹 시간})}\times 100$$

실험 결과 6개의 벤치마크 프로그램들에 대하여 평균 64.90의 감소율을 보였다. 특히, gcc 프로그램의 경우에는 81.57의 감소율을 보였다. 이와 같은 링킹 시간의 원인을 살펴보기 위하여 통합된 모듈 크기와 단위 모듈

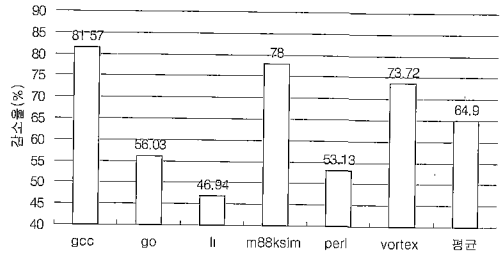


그림 10 단위 모듈 링킹의 컴파일, 링킹 시간 감소율

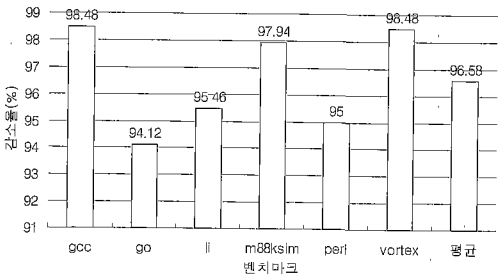


그림 11 단위 모듈 크기 감소율

크기를 비교하였다. 그림 11은 단위 모듈 크기의 감소율을 나타낸다. 이 감소율도 앞의 컴파일, 링킹 시간 감소율과 같은 방법으로 구하였다. 측정 결과 평균 96.58의 감소율을 보였다. 이것은 단위 모듈 링킹이 통합 모듈 링킹보다 컴파일, 링킹 시간이 줄은 이유는 단위 모듈 링킹이 호스트에서의 모듈 통합을 위한 링킹 시간과 타겟으로의 모듈 전송 과부하를 줄였기 때문이다.

6. 결 론

일반적인 내장형 시스템 응용 프로그램 개발 환경은 호스트-타겟 연결 미들웨어가 타겟 메모리를 관리하고 타겟에 목적 화일의 링킹, 다운로딩을 수행한다. 이와 같은 개발 환경에서의 링킹, 다운로딩을 수행하는 링커는 목적 화일을 타겟 메모리에 다운로딩시킬 경우, 하나씩 목적 화일을 다운로딩시켜야 한다. 이때, 사용자는 목적 화일들 간의 참조 관계를 고려하여 다운로딩 순서를 결정하거나, 호스트에서 하나의 목적 화일로 통합하여 다운로딩해야 했다[2]. 그러나, 사용자가 모듈간의 참조 관계를 제대로 알지 못하고, 로딩시켜야 할 목적 화일들의 개수가 많을 경우에는 일일이 화일들 간의 참조 관계를 고려하여 다운로딩 순서를 정하기는 어려운 일이다. 그리고, 하나의 목적 화일로 통합하여 다운로딩하는 경우는 프로그램 개발의 링킹 시간을 증가시킬 수 있다.

본 논문에서 제시한 점진적 원격 링킹을 수행하는 링커는 사용자가 로딩시켜야 할 목적 화일들 간의 로딩 순서를 고려해야 하는 문제점을 해결하였다. 이것은 사용자에게 부담을 덜 주는 편리한 내장형 시스템 응용 프로그램 개발 환경 구축에 적용될 수 있다. 또한, 하나의 목적 화일로 통합하여 다운로딩하는 것 보다 링킹 시간을 단축시킬 수 있다. 링킹 시간 단축 효과는 실험을 통하여 살펴보았다. SPEC95 정수형 벤치마크 프로그램들에[15] 대하여 실험 결과, 점진적 원격 링킹에 의한 방법이 목적 화일들을 통합하여 하나의 화일로 다운로딩하는 것 보다 평균 64.90 정도의 링킹 시간을 감소시킬 수 있었다.

또한, 본 논문에서는 링커를 목적 화일 형식에 종속적인 리더 모듈과 독립적인 링커 모듈로 세분화하였다. 리더 모듈은 목적 화일을 분석하여 형식에 독립적인 링킹 정보를 추출하고, 이 링킹 정보로부터 링커 모듈은 점진적 원격 링킹을 수행한다. 이와 같은 링커의 세분화는 타겟 시스템이 변경되어 목적 화일 형식이 변경되는 경우에, 링커 모듈을 포함한 개발 환경 전체는 변하지 않고 리더 모듈만 변경되는 장점을 갖는다. 즉, 내장형 시스템 개발 환경에서 타겟 시스템에 대한 이식성을 높일 수 있다.

본 연구의 링커는 실제 상용화 단계에 있는 ESTO 개발 환경의 내부 모듈로 구현되었다. StrongARM-110 프로세서의 COFF, ELF 목적 화일 형식을 지원하는 두 리더 모듈을 개발하였고, 리더가 분석한 링킹 정보로부터 점진적 원격 링킹을 수행하는 링커를 개발하였다.

자동 링커 생성기는 본 연구의 링커가 갖는 재목적성을 개선할 수 있다. 타겟 프로세서 또는 목적 화일 형식 명세로부터 자동으로 링커를 생성하는 것은 자동 링커 생성기는 GNU 개발 환경의 BFD 라이브러리를[11] 사용하여 구현될 수 있다. 내장형 시스템을 위한 자동 링커 생성기는 향후 연구될 내용이다.

참 고 문 헌

[1] 김흥남, "사용자 개발 도구", 정보가전용 실시간 OS 컨퍼런스 자료집, pp. 178-196, 1999년 11월.

[2] WindRiver Systems, "Tornado user's guide," http://www.wrs.com.

[3] D. Jaggar, *"ARM Architectural Reference Manual,"* Prentice Hall, 1996.

[4] 김태근, "실시간 OS 커널 개발 연구", 정보가전용 실시간 OS 컨퍼런스 자료집, pp.70-94, 1999년 11월.

[5] G. R. Gircys., *"Understanding and Using COFF,"* O'Reilly Associates, 1988.

[6] Tool Interface Standards(TIS) Committee, "Exe-

cutable and linking format (ELF) specification (version 1.2)," Available by ftp://ftp.x86.org/manuals/tools/elf.pdf.
[7] WindRiver Systems, "Tornado," http://www.wrs.com.
[8] D. E. Simon, *"An Embedded Software Primer,"* pp. 261-281, Addison-Wesley, 1999.
[9] R. W. Quong and M. A. Linton, "Linking programs incrementally," *ACM Transactions on Programming Languages and Systems*, Vol.13, No.1, pp. 1-20, January 1991.
[10] J. Arceneaux, M. Tiemann, and D. V. Henkel-Wallace, "The portability of GNU software," *Proceedings of the Spring 1992 EurOpen USENIX Workshop*, pp 89-103, 1992.
[11] S. Chamberlain, "libbfd-the binary file descriptor library," April 1991.
[12] D. Ung and C. Cifuentes, "SRL-a simple retargetable loader," *Proceedings of the Australian Software Engineering Conference*, pp.60-69, October 1997.
[13] 공기석, 손승우, 임채덕, 김홍남, "내장형 실시간 소프트웨어의 원격디버깅을 위한 디버그에이전트의 설계 및 구현", 1999년 가을 정보과학회 학술발표논문집 (III), pp. 407-409, 1999년 10월.
[14] Development Systems Business Unit Engineering Software Group, "ARM ELF," 1999.
[15] SPEC(Standard Performance Evaluation Corporation), *"SPEC95 Documentation,"* 1995.

우 덕 균

1993년 홍익대학교 컴퓨터공학과(학사). 1995년 홍익대학교 전자계산학과(석사). 2001년 홍익대학교 전자계산학과(박사). 2001년 1월 ~ 현재 한국전자통신연구원 인터넷정보가전연구부 선임연구원. 관심분야는 최적화 컴파일러, 내장형 시스템 응용 개발 환경

한 경 숙

1993년 홍익대학교 컴퓨터공학과(학사). 1995년 홍익대학교 전자계산학과(석사). 1995년 1월 ~ 1997년 2월 삼성전자 근무. 1997년 ~ 현재 홍익대학교 전자계산학과 박사과정. 관심분야는 최적화 컴파일러, 내장형 시스템 응용 개발 환경

표 창 우

1980년 서울대학교 전자공학과(학사). 1982년 서울대학교 컴퓨터공학과(석사). 1989년 Univ. of Illinois at Urbana Champaign 전산학과(박사). 1990년 1월 ~ 1991년 3월 Research Fellow, US Army Construction Engineering Research. 1991년 4월 ~ 현재 홍익대학교 컴퓨터공학과 부교수. 관심분야는 최적화 컴파일러, 내장형 시스템 응용 개발 환경

김 홍 남

1980년 서울대학교 전자공학과(학사). 1989년 미국 Ball State University 전산학(석사). 1996년 미국 Pennsylvania State University 전산학(박사). 1983년 ~ 현재 한국전자통신연구원 인터넷정보가전연구부 내장형S/W연구팀장. 관심분야는 Real-time Operating System, Video Compression Algorithm, Distributed Multimedia System