(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2013년09월17일
(11) 등록번호 10-1308781
(24) 등록일자 2013년09월09일

(51) 국제특허분류(Int. Cl.)
G06F 9/445 (2006.01) *G06F 9/45* (2006.01)
G06F 9/455 (2006.01)
(21) 출원번호 10-2009-7009185
(22) 출원일자(국제) 2007년10월01일
심사청구일자 2011년01월31일
(85) 번역문제출일자 2009년05월01일
(65) 공개번호 10-2009-0095556
(43) 공개일자 2009년09월09일
(86) 국제출원번호 PCT/GB2007/050600
(87) 국제공개번호 WO 2008/041028
국제공개일자 2008년04월10일
(30) 우선권주장
0619389.0 2006년10월02일 영국(GB)
60/854,054 2006년10월24일 미국(US)
(56) 선행기술조사문헌
US20030079215 A1*
US20030088860 A1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
인터내셔널 비지네스 머신즈 코포레이션
미국 10504 뉴욕주 아몬크 뉴오차드 로드
(72) 발명자
브라운 알렉산더 바라클러프
영국 셰필드 요크셔 에스10 2피엘 블룸힐 뉴볼드 래인 125
(74) 대리인
허정훈

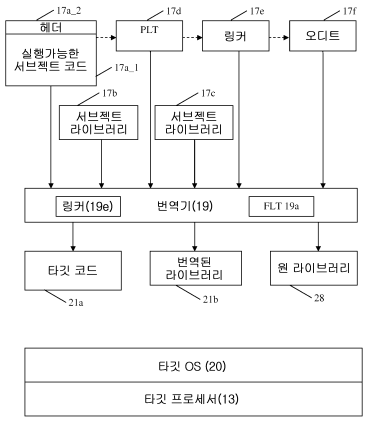
전체 청구항 수 : 총 25 항 심사관 : 지정훈

(54) 발명의 명칭 **프로그램 코드 변환과 관련된 링크된 함수 호출을 동적으로 처리하는 방법 및 장치**

(57) 요 약

서브젝트 제어 흐름을 진행 연결 테이블과 같은 중간 제어 구조로 패스하고, 그리하여 서브젝트 코드의 번동안 상기 서브젝트 함수 호출과 연관된 링크 정보를 동적인 이진 번역기내의 타겟 코드로 수정하기 위한 서브젝트 링커 코드로, 패스하는 동적으로 링크된 서브젝트 함수 호출을 처리하는 기술이 제공된다. 상기 서브젝트 프로세서상의 실행용 서브젝트 코드는 번역기에 의해 수신되고, 상기 타겟 프로세서상의 실행용 대응되는 타겟 코드는 생성된다. 상기 번역기는 상기 서브젝트 코드에 의해 호출된 각 함수에 대한 위치를 제공하는 엔트리를 포함하는 함수 연결 테이블을 생성하며, 그리하여, 코드는, 상기 중간 제어 구조에 대응되는 타겟 코드를 생성하지 않으면서, 번역기에 의해 서브젝트 함수가 상기 함수를 실행하기 위한 코드와 연관되는 코드가 생성될 수 있다.

대 표 도 - 도3



특허청구의 범위

청구항 1

타겟 프로세서; 및

서브젝트 프로세서에서의 실행용 서브젝트 코드를 수신하여 상기 타겟 프로세서에서의 실행용 타겟 코드를 생성하는 번역기;를 포함하고,

상기 번역기는,

FLT(function linkage table); 및

로직(logic);을 포함하고,

상기 로직은,

상기 서브젝트 코드의 동적으로 링크된 서브젝트 함수 호출(function call)과 연관된 PLT(procedure linked subject)의 변경을 검출하고, 상기 동적으로 링크된 서브젝트 함수 호출을 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 코드와 연관시키고, 상기 변경의 검출에 응답하여 상기 FLT에 엔트리를 추가하며,

상기 엔트리는,

상기 동적으로 링크된 서브젝트 함수 호출에 대응하는 식별자; 및 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 상기 코드의 위치(location)을 포함하는,

컴퓨터 장치.

청구항 2

제1항에 있어서,

상기 번역기는 상기 수신된 상기 서브젝트 코드 내의 상기 동적으로 링크된 서브젝트 함수 호출을 식별하고, 상기 서브젝트 코드의 제1 번역상에서 상기 동적으로 링크된 서브젝트 함수 호출에 대한 연결 정보(linking information)를 수집하고, 상기 서브젝트 코드의 다음 번역들(subsequent translations)에서 수집된 상기 연결 정보를 이용하는,

컴퓨터 장치.

청구항 3

제2항에 있어서,

상기 번역기는 상기 다음 번역들에서 상기 수집된 연결 정보를 이용하여 상기 서브젝트 코드 내의 상기 동적으로 링크된 서브젝트 함수 호출로부터 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 상기 코드로 제어 흐름(control flow)을 이동(transfer)시키는,

컴퓨터 장치.

청구항 4

제1항에 있어서,

상기 번역기는 동적으로 링크된 함수 호출에 대한 정보를 상기 동적으로 링크된 함수 호출에 관한 식별자의 형태로 상기 번역기에 접근가능한 FLT에 저장하고, 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 코드의 위치를 상기 FLT에 저장하는,

컴퓨터 장치.

청구항 5

제4항에 있어서,

상기 번역기는 상기 동적으로 링크된 함수 호출에 대응하는 엔트리에 대한 FLT를 체크하고, 상기 FLT에 저장된 정보를 이용하여 상기 동적으로 링크된 함수 호출을 상기 동적으로 링크된 함수 호출을 실행하기 위한 코드와 연관시키기 위해 상기 FLT가 상기 동적으로 링크된 함수 호출에 대응하는 엔트리를 포함하는지 여부를 체크하는,

컴퓨터 장치.

청구항 6

제2항에 있어서,

상기 로직은 서브젝트 링커 활동을 검출하기 위한 로직; 및 서브젝트 코드 명령어 시퀀스를 서브젝트 명령어 캐시 플러시 이벤트(subject instruction cache flush events)의 시퀀스 형태로 검출하기 위한 로직; 중 하나를 포함하는 것인,

컴퓨터 장치.

청구항 7

제1항에 있어서,

상기 번역기는 상기 동적으로 링크된 서브젝트 함수 호출과 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 코드 간의 다이렉트 링크(direct link)를 확립(establish)하는,

컴퓨터 장치.

청구항 8

제4항에 있어서,

상기 번역기는, 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 네이티브 코드(native code)의 위치; 상기 동적으로 링크된 서브젝트 함수 호출를 실행하기 위한 이전에 번역된 타겟 코드의 위치; 및 상기 번역기에 알려져 있고 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위해 타겟 코드가 생성될 수 있는 최적화된 중간 표현(optimised intermediate representation)의 일부(portions)의 위치; 중 하나 이상을 상기 FLT에 저장하는 로직을 추가적으로 포함하는,

컴퓨터 장치.

청구항 9

제1항에 있어서,

상기 번역기는 엄격하게 정의된 함수(strictly defined functions)와 관련된 동적으로 링크된 함수 호출을 식별하고, 상기 엄격하게 정의된 함수와 관련된 동적으로 링크된 함수 호출을 상기 엄격하게 정의된 함수와 관련된 동적으로 링크된 함수 호출을 실행하기 위한 코드로 대체하는,

컴퓨터 장치.

청구항 10

제4항에 있어서,

상기 번역기는 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 서브젝트 코드; 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 네이티브 코드; 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위해 이전에 번역된 타겟 코드; 상기 번역기에 알려져 있고 상기 동적으로 링크된 함수 호출을 실행하기 위해 타겟 코드가 생성될 수 있는 최적화된 중간 표현 영역(optimised intermediate representation)의 일부(portions); 중 하나의 위치와 관련된 역참조된(de-referenced) 변수를 상기 FLT에 저장하는,

컴퓨터 장치.

청구항 11

제1항에 있어서,

상기 번역기는, 서브젝트 제어 흐름을 상기 서브젝트 코드의 PLT(Procedure Linkage Table)를 경유하여 서브젝트 링커 코드로 전달(pass)하는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 수신하고, 상기 서브젝트 코드내에서 상기 동적으로 링크된 서브젝트 함수 호출을 식별하고, 상기 PLT에 대응되는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 서브젝트 함수 호출을 대응되는 함수를 실행하기 위한 코드와 연관시키는,

컴퓨터 장치.

청구항 12

제1항에 있어서,

상기 번역기는, 서브젝트 제어 흐름을 자체적으로 변경한 서브젝트 코드(self modifying subject code)의 영역을 경유하여 서브젝트 링커 코드로 전달(pass)하는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 수신하고, 상기 서브젝트 코드 내의 상기 동적으로 링크된 함수 호출을 식별하고, 자체적으로 변경한 서브젝트 코드에 대응되는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 서브젝트 함수 호출을 대응되는 함수를 실행하기 위한 코드와 연관시키는,

컴퓨터 장치.

청구항 13

서브젝트 프로세서에서의 실행용 서브젝트 코드를 번역기에서 수신하는 단계 - 수신된 상기 서브젝트 코드는 동적으로 링크된 서브젝트 함수 호출을 포함함-;

상기 서브젝트 코드에 연관된 PLT(procedure linkage table)에 대한 변경을 검출하는 단계;

상기 동적으로 링크된 서브젝트 함수 호출을, 대응하는 함수를 실행하기 위한 코드와 연관시키는 단계;

상기 변경의 검출에 응답하여, 상기 번역기에 연관된 FLT(function linkage table)에 엔트리를 추가하는 단계;를 포함하고,

상기 엔트리는,

상기 동적으로 링크된 서브젝트 함수 호출에 대응하는 식별자; 및

상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 코드의 위치;를 포함하는 방법.

청구항 14

제13항에 있어서,

상기 서브젝트 코드의 제1 번역에서, 상기 동적으로 링크된 서브젝트 함수 호출에 대한 연결(linking) 정보는 상기 수신된 서브젝트 코드의 다음 번역들에서 사용되기 위해 수집되는 방법.

청구항 15

제14항에 있어서,

상기 다음 번역들에서, 상기 제1 번역에서 수집된 상기 연결 정보는 상기 동적으로 링크된 서브젝트 함수 호출로부터 상기 대응하는 함수를 실행하기 위한 코드로 제어 흐름(control flow)을 이동시키는데 사용되는 방법.

청구항 16

제13항에 있어서,

상기 동적으로 링크된 서브젝트 함수 호출에 대한 정보를 FLT(function linkage table)에 저장하고, 상기 동적으로 링크된 함수 호출에 대응하는 엔트리에 대해 상기 FLT를 체크하고, 상기 FLT가 상기 동적으로 링크된 서브젝트 함수 호출에 대응하는 엔트리를 포함하는지 여부를 체크하고, 상기 FLT에 저장된 상기 정보를 이용하여 상기 동적으로 링크된 서브젝트 함수 호출을 상기 대응하는 함수를 실행하기 위한 코드와 연관시키는 단계를 포함

하는 방법.

청구항 17

제14항에 있어서,

상기 동적으로 링크된 서브젝트 함수 호출에 대한 연결 정보의 수집은, 서브젝트 링커 활동(subject linker activity)을 모니터하는 단계를 포함하는 방법.

청구항 18

제14항에 있어서,

상기 동적으로 링크된 서브젝트 함수 호출에 대한 연결 정보의 수집은 서브젝트 명령어 캐시 플러시 이벤트(subject instruction cache flush events)의 시퀀스 형태로 서브젝트 코드 명령어 시퀀스를 검출하는 단계를 포함하는 방법.

청구항 19

제13항에 있어서,

상기 동적으로 링크된 서브젝트 함수 호출과 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 코드 간의 다이렉트 링크(direct link)를 타겟 코드 내에 확립하는 단계를 포함하는 방법.

청구항 20

제16항에 있어서,

상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 네이티브 코드(native code)의 위치;

상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위해 이전에 번역된 타겟 코드의 위치;

상기 번역기에 알려져 있고 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위해 타겟 코드가 생성될 수 있는 최적화된 중간 표현(optimised intermediate representation)의 일부(portions)의 위치;

상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 서브젝트 코드, 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 네이티브 코드, 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위해 이전에 번역된 타겟 코드, 및 상기 번역기에 알려져 있고 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위해 타겟 코드가 생성될 수 있는 최적화된 중간 표현(optimised intermediate representation)의 일부(portions) 중 하나의 위치와 관련된 역참조된(de-referenced) 변수;를 포함하는 그룹으로부터 선택된 하나 이상을 상기 FLT에 저장하는 단계를 추가적으로 포함하는 방법.

청구항 21

제14항에 있어서,

엄격하게 정의된 함수(strictly defined functions)와 관련된 동적으로 링크된 함수 호출을 식별하고, 상기 엄격하게 정의된 함수와 관련된 동적으로 링크된 함수 호출을 상기 엄격하게 정의된 함수와 관련된 동적으로 링크된 함수를 실행하기 위한 코드로 대체하는 단계를 포함하는 방법.

청구항 22

제14항에 있어서,

서브젝트 제어 흐름을 상기 서브젝트 코드의 PLT(Procedure Linkage Table)를 경유하여 서브젝트 링커 코드로 전달(pass)하는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 수신하는 단계;

상기 서브젝트 코드에서 상기 동적으로 링크된 서브젝트 함수 호출을 식별하는 단계; 및

상기 PLT에 대응하는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 서브젝트 함수 호출을 대응하는 함수를 수행하기 위한 코드와 연관시키는 단계;를 포함하는 방법.

청구항 23

제14항에 있어서,

서브젝트 제어 흐름을 자체적으로 변경한 서브젝트 코드(self modifying subject code)의 영역을 경유하여 서브젝트 링커 코드로 전달(pass)하는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 수신하는 단계;

상기 서브젝트 코드에서 상기 동적으로 링크된 서브젝트 함수 호출을 식별하는 단계; 및

상기 자체적으로 변경한 서브젝트 코드에 대응되는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 서브젝트 함수 호출을 상기 대응되는 함수를 실행하기 위한 코드와 연관시키는 단계;를 추가적으로 포함하는 방법.

청구항 24

제13항에 기재된 방법의 각 단계를 수행하기 위해 컴퓨터에 의해 실행가능한 명령을 기록한 컴퓨터 판독가능한 저장 매체.

청구항 25

서브젝트 프로세서에서의 실행용 서브젝트 코드를 수신하여 타겟 프로세서에서의 실행용 타겟 코드를 생성하는 번역기 장치에 있어서,

프로세서;

상기 프로세서에 결합된 컴퓨터 판독가능한 저장 매체; 및

서브젝트 프로세서에서의 실행용 서브젝트 코드를 수신하고 -수신된 상기 서브젝트 코드는 동적으로 링크된 서브젝트 함수 호출을 포함함-, 상기 서브젝트 코드에 연관된 PLT(procedure linakge table)에 대한 변경을 검출하고, 상기 동적으로 링크된 서브젝트 함수 호출을 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 코드와 연관시키고, 상기 번역기에 연관된 FLT(function linkage table)에 엔트리를 추가 -상기 엔트리는 상기 동적으로 링크된 서브젝트 함수 호출에 대응하는 식별자 및 상기 동적으로 링크된 서브젝트 함수 호출을 실행하기 위한 코드의 위치(location)를 포함함- 하도록, 상기 컴퓨터 판독가능한 저장 매체에 저장되고 상기 프로세서 상에서 실행되는 로직(logic);을 포함하는,

번역기 장치.

명 세 서

기 술 분 야

[0001] 본 발명은 컴퓨터 및 컴퓨터 소프트웨어 분야와 일반적으로 관련이 있으며, 특히, 예를 들어, 동적으로 링크된 함수 호출을 포함하는 프로그램 코드를 변환하는 코드 번역기, 에뮬레이터, 가속기에서 유용한 프로그램 코드 변환 방법 및 기기에 관한 것이다.

배 경 기 술

[0002] 엠비디드 및 넌-엠비디드 CPU 시장 전반에 있어서, 그것들이 투명하게 관련 소프트웨어에 접근할 수 있다면, 대부분의 소프트웨어에 존재하고, 더 나은 비용/수행을 제공할 수 있는 무수히 많은 유능한 프로세서로의 "번역(translated)" 또는 실행을 위해 "가속(accelerated)"화 될 수 있는 대부분의 ISAs(Instruction Set Architectures)는 유익하다는 것을 누구나 발견한다. 또한, 제시간에 ISA와 연결되고, 그리고 실행 면에서 또는 시장범위에서 진화할 수 없고, “종합적인 CPU(synthetic CPU)” 코-아키텍처(co-architecture)로부터 유익한 대부분의 CPU 아키텍처를 누구나 발견한다.

[0003] 자주, 제2 타입의 프로세서("타겟" 프로세서)상에서 제1 타입 컴퓨터 프로세서("서브젝트" 프로세서)를 위해 쓰인 프로그램 코드를 동작시키는 것을 바란다. 여기서, 에뮬레이터 또는 번역기는 프로그램 코드 변환을 수행하기 위해 사용되며, 그리하여 서브젝트 프로그램은 타겟 프로세서상에서 동작할 수 있다. PCT 출원 WO00/22521은 본 발명의 예에서 적용될 수 있는, 그러한 가속, 번역 및 코-아키텍처 성능을 용이하게 하는 프로그램 코드 변환 방법 및 기기를 개시한다.

[0004] 번역될 서브젝트 프로그램은, 라이브러리의 일부는 독점적일 수 있으며, 라이브러리의 일부는 서브젝트 OS(operating system)의 일부로서 제공되는("시스템 라이브러(system libraries)"), 많은 수의 서브젝트 라이브러리 및 서브젝트 어플리케이션을 포함하면서, 일반적으로 서브젝트 코드의 멀티플 유닛으로 구성된다. 서브젝트 프로그램이 동작할 때, 제어 흐름은, 라이브러리에 대한 함수 호출이 만들어지는 것과 같이, 서브젝트 코드의 다른 유닛들 간에 패스된다.

[0005] 예를 들어 선 마이크로시스템스 회사(Sun Microsystems Inc.)로부터의 솔라리스(Solaris)와 같은, 어떤 OS(operating system)에서, 함수 호출을 함수를 수행하는 라이브러리 코드로 링크하는 과정은 동작 시간에 수행될 수 있으며, 이 과정은 동적인 링크로 알려져 있다. 동적인 링크는 동적인 링커 코드에 의해 실행되며, PLT(Procedure Linkage Table)로 알려진 중간 제어 구조의 사용을 포함한다.

[0006] PLT는 컴파일된 프로그램의 일부이며, 프로그램에 의해 요청된 각 라이브러리 함수에 대한 링크 정보를 포함하는 엔트리(entry)을 포함한다. 라이브러리 함수에 대한 제1 호출이 프로그램의 전형적인 동작에서 만났을 때, 제어 흐름은 그 함수와 관련된 PLT 엔트리(entry)로 점프한다. 그 함수에 대한 PLT 엔트리는 동적인 링커 코드를 끌어냄으로써 이 단계에서 연결 프로세스(linking process)를 제어한다. 동적인 링커 코드는 질문내 함수에 대한 링크 정보가 업데이트되도록 한다. 링크 정보를 업데이트함으로써, 동적인 링커 코드는 그 함수를 위한 PLT 엔트리에서 그 함수를 수행하기 위한 라이브러리 코드로 링크를 만든다. 이 방법으로 확립된 링크는 프로그램 동작의 나머지 동안 전형적으로 지속된다.

[0007] 그 때 동적인 링커 코드는 함수가 실행될 수 있도록 라이브러리내 코드로 제어 흐름을 패스한다.

[0008] 프로그램에 의한 함수의 다음 주문은 PLT 엔트리 전에 처럼 제어 흐름을 패스한다. PLT 엔트리가 함수를 수행하는 라이브러리 코드에 대한 링크와 함께 업데이트되었기 때문에, 지금 PLT는 제어 흐름이 거기에서 이 다음 주문내의 라이브러리로 직접 패스하도록 한다. 이 다음 주문은 PLT 엔트리의 추가적인 업데이트를 요구하지도 않고, 동적인 링커의 추가적인 주문을 요구하지도 않는다.

[0009] 어떤 OS에서, 함수를 PLT 엔트리와 함수를 실행하기 위한 코드를 포함하는 라이브러리간의 링크를 생성하는 동작은, 글로벌 옵셋 테이블과 같은, PLT와 관련된 데이터를 수정하는 효과를 갖는다. 그때, 수정된 글로벌 옵셋 테이블은 PLT 엔트리내의 코드에 의해 동작시간(run-time)에서 판독될 수 있고, 링크 정보를 줄 수 있다. 예를 들어, 솔라리스와 같은, 어떤 다른 OS에서, 어떤 함수를 위한 PLT 엔트리와 그 함수를 포함하는 라이브러리간의 링크를 생성하는 동작은 PLT 엔트리 자체를 형성하는 실행가능한 코드를 수정하는 대안적인 효과를 갖는다.

[0010] 위에서 언급한 바와 같이, 중간 제어 구조로서 PLT 사용, 특히, 함수가 호출되는 제1 시간에서 PLT를 포함하는 코드를 수정함으로써 PLT내의 링크 정보에 대한 수정은, 동적인 링크 메커니즘의 일부로서 PLT를 적용하는 서브젝트 코드에 대한 프로그램 코드 변환을 복잡하게 만든다.

[0011] 수정된 서브젝트 코드가 이미 번역된 타겟 코드와 대응될 수 있기 때문에, 동작시간에 코드 수정은 동적인 번역기에 문제를 일으킨다. 서브젝트 코드의 그러한 수정이 발생할 때, 수정된 서브젝트 코드의 모든 타겟 코드 번역은 확인되어야 하고 스테일(stale)로서 버려져야 한다. 그리하여, 번역기는 수정된 특정 서브젝트 코드 어드레스에 대응되는 모든 타겟 코드 시퀀스(즉, 번역)을 확인할 수 있어야 한다.

[0012] 동적인 번역기에서, 주어진 서브젝트 어드레스에 대응되는 타겟 코드를 발견하고 삭제하는 것은 어려우며, 때때로 가능하지 않을 수 있다. 일부 상황에서, 번역이 표현하는 서브젝트 어드레스의 범위와 더 이상 정확하게 관련이 없는 번역을 야기하는 번역 동안, 최적화가 적용될 수 있다. 이러한 상황에서, 서브젝트 프로그램이 어떤 서브젝트 어드레스에서 그 자체의 코드를 수정한다면, 번역기는 무효화될 각각의 번역된 타겟 코드가 어떤 것인지 확인하는 방법을 갖지 않는다. 게다가, 멀티-쓰레디드(multi-threaded) 환경에서 번역된 타겟 코드에 대한 안전한 삭제는 보다 더 큰 문제를 야기할 수 있다.

[0013] 코드 수정을 취급하는 기술은 PCT 출원 WO 05/008487에서 설명되었다. 그러한 기술이 유용하다 할지라도, 발명자는 동작시간에 PLT에서 발생한 코드 수정의 높은 밀집은 PCT WO 05/008487에 묘사된 바와 같은 기술이 PLT 업데이트를 취급하는 방법을 비효율적으로 만들 수 있다는 것을 지금 확인하였다. 그러한 기술은 무효가 아닌 번역된 타겟 코드가 실행되었다고 확신할 수 있고, 그러나, 이러한 기술을 이용하여 PLT 업데이트를 취급할 때 제어 흐름의 관리는 프로세스 및 메모리 리소스 관점에서 값비싸다는 것이 발견되었다.

[0014] 무엇보다, 발명자는 솔라리스 OS에서 동적인 링커 ld.so에 의해 사용된 PLT 업데이트 방법이, 몇몇 다른 OS 내의 동적인 링커와 비교하여, 전형적으로 동적인 링크 프로그램을 실행할 필요가 있는 많은 부분을 현저하게 증

가시킨다는 것을 지금 확인하였다.

[0015] 취급 방법을 이용한 컴퓨터 기기와 함께, 서브젝트 프로그램 코드의 번역과 관련하여 동적으로 링크된 함수 호출을 취급하는 방법이 제공된다. 바람직한 실시예는, 중간 제어 구조에 대응되는 타겟 코드를 생성하지 않고, 링크가 함수 연결 테이블을 이용하여 서브젝트 코드내의 함수 호출로부터 서브젝트 코드 함수에 대응되는 코드로 확립될 수 있도록 한다. 서브젝트 코드를 타겟 코드로 변환하도록 동작하는 번역기를 포함하는 컴퓨터 기기는 서브젝트 프로그램내의 동적으로 링크된 함수 호출이 확인될 수 있도록, 그리고 함수를 실행하기 위한 코드와 직접적인 관련을 갖는 타겟 코드가 생성될 수 있도록 하며, 상기 관련은 번역기에 의해 함수 연결 테이블로 수집되는 정보에 기초한다. 동적으로 링크된 함수 호출을 취급하는 기술의 바람직한 실시예는 서브젝트 코드내 동적으로 링크된 함수 호출이 함수를 실행하기 위한 코드와 편리하게 연관될 수 있도록 한다. 바람직한 실시예는, 코드 수정을 취급하는 PCT WO05/008487로부터 알려진 것들과 같은 기술과 관련된 전반적인 메모리 및 처리를 감소시킬 수 있다.

[0016] 그리하여, 예를 들어, 프로그램 코드 변환을 하는 동안, 본 발명은 컴퓨터 시스템의 성능을 향상시킬 수 있다.

발명의 상세한 설명

[0017] 본 발명에 따라 첨부된 청구항에서 처럼 장치 및 방법이 제공된다. 본 발명의 바람직한 특징은 이하의 종속항 및 설명으로부터 명확해질 것이다.

[0018] 일 관점에서, 타겟 프로세서; 및 서브젝트 프로세상에서 실행용 서브젝트 코드를 수신하고, 상기 타겟 프로세서상에서 실행용 타겟 코드를 생성하는 번역기;를 포함하고, 상기 번역기는, (a) 상기 서브젝트 코드내 중간 제어 구조를 경유하여 서브젝트 제어 흐름을 서브젝트 링커 코드로 패스하고, 그리하여 상기 함수를 실행하기 위한 서브젝트 코드와 연관시키는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 수신하고, 그리고, (b) 상기 수신된 서브젝트 코드내에 상기 동적으로 링크된 함수 호출을 확인하고, 상기 중간 제어 구조에 대응되는 타겟 코드를 생성하지 않으면서, 상기 대응되는 함수를 실행하기 위한 코드와 상기 동적으로 링크된 함수 호출을 연관시키는; 컴퓨터 기기가 제공된다.

[0019] 상기 번역기 기기는 상기 수신된 서브젝트 코드내의 동적으로 링크된 서브젝트 함수 호출을 확인하고, 상기 수신된 서브젝트 코드의 제1 번역상에서 상기 동적으로 링크된 함수 호출에 대한 연결(linking) 정보를 수집하며, 상기 수신된 서브젝트 코드의 다음 번역에서 상기 수집된 정보를 이용하는 것이 바람직하다.

[0020] 상기 컴퓨터 기기는 상기 번역기 기기가 상기 확인된 함수 호출로부터 제어 흐름을 상기 대응되는 함수를 실행하기 위한 상기 코드로 이동시키기 위해 다음 번역내의 상기 수집된 정보를 이용하는 것이 바람직하다.

[0021] 상기 컴퓨터 기기는, 상기 번역기가 서브젝트 제어 흐름이 상기 서브젝트 코드의 PLT(Procedure Linkage Table)를 경유하여 서브젝트 링커 코드로 패스하는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 수신할 때, 이것들은, 상기 번역기에 의해, 상기 수신된 서브젝트 코드에서 확인되고, 그리고, 상기 PLT에 대응되는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 함수 호출을 상기 대응되는 함수 실행하기 위한 코드와 연관되는 것이 바람직하다.

[0022] 다른 관점에서, 서브젝트 프로세서상에서, 제어 흐름을 상기 서브젝트 코드내 중간 제어 구조를 경유하여 서브젝트 링커 코드로 패스하고, 그리하여 상기 함수를 실행하기 위한 서브젝트 코드와 연관시키는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 실행을 위해 수신하고; 상기 수신된 서브젝트 코드내의 상기 동적으로 링크된 함수 호출을 확인하며; 및 상기 중간 제어 구조와 대응되는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 함수 호출을 상기 대응되는 함수를 실행하기 위한 코드와 연관시키는 상기 타겟 프로세서상의 실행용 타겟 코드를 생성하는, 단계를 포함하는, 타겟 프로세서상에서 실행된 컴퓨터 코드 변환 방법이 제공된다.

[0023] 여전히 다른 관점에서, 서브젝트 프로세서상에서, 제어 흐름을 상기 서브젝트 코드내 중간 제어 구조를 경유하여 서브젝트 링커 코드로 패스하고, 그리하여 상기 함수를 실행하기 위한 서브젝트 코드와 연관시키는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 실행을 위해 수신하고; 상기 수신된 서브젝트 코드내의 상기 동적으로 링크된 함수 호출을 확인하며; 및 상기 중간 제어 구조와 대응되는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 함수 호출을 상기 대응되는 함수를 실행하기 위한 코드와 연관시키는 상기 타겟 프로세서상의 실행용 타겟 코드를 생성하는; 단계를 포함하는 타겟 프로세서상에서 실행된 프로그램 코드 변환 방법을 실행하기 위해 컴퓨터에 의해 수행가능한 명령을 거기에 기록한 컴퓨터-판독가능한 매체가 제

공된다.

[0024] 여전히 다른 관점에서, 서브젝트 프로세서상에서 실행용 서브젝트 코드를 수신하고 상기 타겟 프로세서 상에서 실행용 타겟 코드를 생성하는 번역기 기기가 제공되며, 여기서, 상기 번역기 기기는, 서브젝트 제어 흐름을 상기 서브젝트 코드내 중간 제어 구조를 경유하여 서브젝트 링커 코드로 패스하고, 그리하여 상기 함수를 실행하기 위한 서브젝트 코드와 연관시키는 하나 이상의 동적으로 링크된 서브젝트 함수 호출을 포함하는 서브젝트 코드를 서브젝트 프로세서상에서 실행을 위해 수신하고; 상기 수신된 서브젝트 코드내에서 상기 동적으로 링크된 함수 호출을 확인하며; 및 상기 중간 제어 구조에 대응되는 타겟 코드를 생성하지 않으면서, 상기 동적으로 링크된 함수 호출이 상기 대응되는 함수를 실행하기 위한 코드와 연관되는 상기 타겟 프로세서상에서 실행용 타겟 코드를 생성한다.

[0025] 상기한 것은 본 발명의 실시예의 다양한 관점에 대한 요약이다. 이것은 계속되고 그리고 그렇지 않은 본 발명의 상세한 설명을 보다 빠르게 받아들일 수 있도록 당업자를 도와주는 지침서로서 제공되고 여기 첨부된 청구항의 범위를 한정하기 위한 임의의 방법으로 의도되지 않는다.

실 시 예

[0031] 다음의 설명은 당업자가 본 발명을 만들고 이용하고 이들 발명을 행하는 발명자들에 의해 시도된 최선의 형태들을 설명할 수 있도록 제공된다. 그러나 본 발명의 일반적인 원리들이 특히 개선된 프로그램 코드 방법 및 기기를 제공하기 위해 여기서 정의되었기 때문에, 다양한 변형 예들은 당업자에게 용이하게 명백한 채로 있을 것이다.

[0032] 도 1을 참조하면, 서브젝트 프로그램(17)은 서브젝트 프로세서(3)를 갖는 서브젝트 컴퓨팅 플랫폼(1)상에서 실행하는 경향이 있다. 그러나 타겟 컴퓨팅 플랫폼(10)은, 프로그램 코드 변환을 수행하는 번역기부(19)를 통해, 서브젝트 프로그램(17)을 대신 실행하는데 이용된다. 번역기부(19)는 서브젝트 코드(17)에서 타겟 코드(21)로 코드 변환을 실행하며, 그리하여 타겟 코드(21)는 타겟 컴퓨팅 플랫폼(10)상에서 실행될 수 있다.

[0033] 당업자에게 익숙한 바와 같이, 서브젝트 프로세서(3)는 서브젝트 레지스터 세트(5)를 갖는다. 서브젝트 메모리(8)는, 특히, 서브젝트 코드(17) 및 서브젝트 OS(2)를 보유한다. 유사하게, 도 1에서 예시적인 타겟 컴퓨팅 플랫폼(10)은 복수개의 타겟 레지스터(15)를 갖는 타겟 프로세서, 타겟 OS(20)를 포함하는 복수 개의 동작 요소를 저장하기 위한 메모리(18), 서브젝트 코드(17), 번역기 코드(19), 및 번역된 타겟 코드(21)를 포함한다. 타겟 컴퓨팅 플랫폼(10)은 일반적으로 마이크로 프로세서-기반 컴퓨터 또는 다른 적합한 컴퓨터이다.

[0034] 일 실시예에서, 번역기 코드(19)는, 최적화와 함께 또는 최적화없이, 서브젝트 ISA(subject instruction set architecture)의 서브젝트 코드를 다른 ISA의 번역된 타겟 코드로 번역하는 에뮬레이터이다. 다른 실시예에서, 번역기(19)는, 프로그램 코드 최적화를 수행함으로써, 서브젝트 코드를 타겟 코드, 각각의 동일한 ISA로 변환하는 가속기와 같은 기능을 한다.

[0035] 적합하게는, 번역기 코드(19)는 번역기를 실행하는 컴파일된(compiled) 버전의 소스 코드이고, 타겟 프로세서(13)상의 OS(20)와 결합하여 동작한다. 도 1에 설명된 구조는 단지 예시적인 것이고, 예를 들어, 본 발명의 실시예에 따른 방법 및 프로세스는 OS(20) 내 또는 밑에 있는 코드에서 실행될 수 있다는 것은 인정되어야 할 것이다. 서브젝트 코드(17), 번역기 코드(19), OS(20), 및 메모리(18)의 저장 메커니즘은, 당업자에게 알려진 바와 같이, 보다 다양한 타입일 수 있다.

[0036] 도 1에 따른 기기에서, 프로그램 코드 변환은, 동작시간(run-time)에, 타겟 코드(21)가 동작하는 동안, 타겟 아키텍처상에서 실행되도록 동적으로 수행된다. 즉, 번역기(19)는 번역된 타겟 코드(21)를 즉시 처리하도록 동작한다. 번역기(19)를 통해 서브젝트 프로그램(17)을 동작하는 것은 인터리브된(interleaved) 방법에서 실행되는 두 개의 다른 타입의 코드 즉, 번역기 코드(19); 및 타겟 코드(21)를 포함한다. 그리하여, 타겟 코드(21)는, 동작 시간 전반에 걸쳐, 번역되고 저장된 프로그램의 서브젝트 코드(17)에 기초하여, 번역기 코드(19)에 의해 생성된다.

[0037] 일 실시예에서, 타겟 프로세서(13)상의 타겟 코드(21)로 서브젝트 프로그램(17)을 실제로 실행하는 동안, 번역기부(19)는 서브젝트 프로세서(3) 및 특히 서브젝트 레지스터(5)와 같은 서브젝트 아키텍처(1)의 관련 영역을 에뮬레이트한다. 바람직한 실시예에서, 적어도 하나의 글로벌 레지스터 스토어(27)(서브젝트 레지스터 뱅크(27) 또는 요약 레지스터 뱅크(27)로도 언급된다.)이 제공된다. 멀티프로세서 환경에서, 임의로 하나 이상의 요약 레지스터 뱅크(27)는 서브젝트 프로세서의 아키텍처에 따라 제공된다. 서브젝트 상태의 표현은 번역기(19) 및 타

겟 코드(21)의 요소에 의해 제공된다. 즉, 번역기(19)는 변수들 및/또는 오브젝트들과 같은 다양한 명시적인 프로그래밍 언어 장치에서 서브젝트 상태를 저장한다. 비교적으로, 번역된 타겟 코드(21)는 타겟 레지스터(15) 및 메모리 로케이션(18)에서 함축적으로 서브젝트 프로세서 상태를 제공하고, 이것은 타겟 코드(21)의 타겟 명령에 의해 조작된다. 예를 들어, 글로벌 레지스터 스토어(27)의 로우-레벨(low-level) 표현은 단순히 할당된 메모리의 영역이다. 그러나 번역기(19)의 소스 코드에서, 글로벌 레지스터 스토어(27)는 보다 높은 레벨(higher level)에서 접근되고 조작될 수 있는 데이터 어레이 또는 오브젝트이다. "기본 블록"이라는 용어는 당업자에게 익숙할 것이다. 기본 블록은 정확하게 하나의 입력 포인트 및 정확하게 하나의 출력 포인트를 갖는 코드 섹션이다. 이 이유 때문에, 기본 블록은 제어 흐름의 유익한 기본 유닛이다. 적합하게는, 번역기(19)는 서브젝트 코드(17)를 복수 개의 기본 블록으로 분류하고, 여기서 각 기본 블록은 단일 입력 포인트에서 첫번째 명령과 단일 출력 포인트에서 마지막 명령간의 (점프, 호출 또는 브랜치(branch) 명령 같은) 연속적인 명령 세트이다. 번역기(19)는 단지 이러한 기본 블록 중 하나(블록 모드)를 선택하거나 기본 블록의 그룹(그룹 블록 모드)을 선택할 수 있다. 적합하게는, 그룹 블록은 단일 유닛으로 함께 취급되는 두 개 이상의 기본 블록을 포함한다. 무엇보다, 번역기는 서브젝트 코드의 동일한 기본 블록 그러나 다른 입력 조건에 속하는 아이소-블럭(iso-blocks)을 형성할 수 있다.

[0038] 바람직한 실시예에서, IR(Intermediate Representation) 트리는 원 서브젝트 프로그램(17)으로부터 타겟 코드(21)를 생성하는 과정의 일부로서, 서브젝트 명령 시퀀스에 기초하여 생성된다. IR 트리는 서브젝트 프로그램에 의해 실행된 동작 및 계산된 식의 요약 표현이다. 나중에, 타겟 코드(21)는 IR 트리에 기초하여 생성된다. IR 노드의 집합은 실질적으로 DAGs(directed acyclic graphs)이나 구어체로 "트리(trees)"로 언급된다.

[0039] 당업자가 인정하는 바와 같이, 하나의 실시예에서, 번역기(19)는 C++과 같은 오브젝트-지향 프로그래밍 언어를 이용하여 수행된다. 예를 들어, IR 노드는 C++ 오브젝트로서 수행되고, 다른 노드에 대한 참조는 그 다른 노드에 대응하는 C++ 오브젝트에 대한 C++ 참조로서 수행된다. 그리하여 IR 트리는 각각에 대한 다양한 참조를 포함하는, IR 노드 오브젝트의 집합으로서 수행된다.

[0040] 무엇보다, 언급하고 있는 실시예에서, IR 생성은, 서브젝트 프로그램(17)이 동작하는 서브젝트 아키텍처의 특정 특징에 대응되는 요약 레지스터 정의 세트를 사용한다. 예를 들어, 서브젝트 아키텍처상의 각 물리적인 레지스터에 대한 고유한 요약 레지스터 정의(예를 들어, 도 1의 서브젝트 레지스터(5))가 있다. 그리하여, 번역기내의 요약 레지스터 정의는 IR 노드 오브젝트(예를 들어, IR 트리)에 대한 참조를 포함하는 C++ 오브젝트로서 수행될 수 있다. 요약 레지스터 정의의 세트에 의해 언급되는 모든 IR 트리의 집합은 워킹 IR 포레스트("포레스트(forest)"왜냐하면 그것은 다중 요약 레지스터 루트(root)를 포함하고, 그것의 각각은 IR 트리를 언급하기 때문이다.)로서 언급될 수 있다. 적합하게는, 이 IR 트리 및 다른 프로세스는 번역기(19)의 일부를 형성한다.

[0041] 도 2는 타겟 컴퓨팅 플랫폼(10)상에서 동작할 때 번역기(19)를 보다 상세하게 설명한다. 위에서 언급한 바와 같이, 번역기(19)의 프론트 엔드(front end)는 (일반적으로 각각은 서브젝트 코드의 하나의 기본 블록을 포함하는) 복수 개의 서브젝트 코드 블록(171a, 171b, 171c)을 제공하기 위해 서브젝트 프로그램(17)의 현재 필요한 섹션을 디코딩하는 디코더부(191)를 포함하고, 그리고 각 서브젝트 블록과 관련된 디코더 정보(172) 및 거기에 포함되고 번역기(19)의 나중 동작을 도와줄 서브젝트 명령을 제공할 수 있다. 어떤 실시예에서, 번역기(19)의 코어(core)내의 IR부(192)는 디코딩된 서브젝트 명령으로부터 IR(intermediate representation)를 생산하고, 그리고 최적화는 IR과 관련하여 적절하게 실행된다. 번역기(19)의 백엔드(back end)의 일부로서 인코더(193)는 타겟 프로세서(13)에 의해 실행될 수 있는 타겟 코드(21)를 생성한다(플랜트한다(plant)). 이 간단한 예에서, 세 개의 타겟 코드 블록(211a-211c)은 서브젝트 플랫폼(1)상에 서브젝트 코드블록(171a-171c)을 실행하는 것과 같은 타겟 플랫폼(10)상에서 일을 실행하기 위해 생성된다. 또한, 인코더(193)는, 타겟 코드가 동작할 환경을 설정하고 적절하게 번역기(19)로 제어신호를 다시 패스하는 것과 같은 기능을 실행하는 일부 또는 모든 타겟 코드 블록(211a-211c)에 대한 헤드 코드(header code) 및/또는 푸터 코드(footer code)(212)를 생성할 수 잇다.

[0042] 도 3은 본 발명의 실시예에 의해 적용되는 기기를 설명하는 보다 상세한 도식도이다. 도 3의 설명적인 예에서, 번역기(19)는 x86 번역에 대한 SPARC를 실행한다.

[0043] 서브젝트 코드(17)는 타겟 코드(21a)로 번역될 서브젝트 실행가능한 파일(17a)을 포함한다. 실행가능한 서브젝트(17a)는 독점적인 라이브러리 및/또는 시스템 라이브러리를 포함하는 다수의 서브젝트 라이브러로부터 함수를 교대로 참조하고 이용할 수 있다. 두 개의 예시적인 라이브러리(17b, 17c)가 설명된다.

[0044] 실행가능한 서브젝트(17a)는 서브젝트 프로세서의 OS와 양립가능한 실행가능한 파일 포함에 따라 구성된다. 전

형적으로, 실행가능한 서브젝트(17a)는 코드의 메인 바디(17a_1) 및 헤더(17a_2)를 포함한다. 헤더(17a_2)는, 예를 들어, 실행가능한 서브젝트(17a)가 동작될 때 동적인 연결을 실행하기 위해 사용되는 정보 및 실행가능한 서브젝트를 파싱하는데 유용한 정보와 같은, 코드의 메인바디(17a_1)에 관한 정보를 제공한다. "헤더"로 언급된다 하더라도, 헤더(17a_2)는 실행가능한 서브젝트(17a)의 시작으로부터 전체적으로 또는 부분적으로 멀리 존재하는 것이 바람직하다.

[0045] 도 3의 설명적인 예에서, 실행가능한 서브젝트(17a)는 ELF(Executable and Linking Format)에 따라 구성된다. ELF 표준이 널리 이용되고, 이 구조 파일의 헤더(17a_1)는 실행가능한 서브젝트(17a)와 연관된 PLT(Procedure Linkage Table)(17a_1) 정보를 포함한다.

[0046] 서브젝트 코드(17)는 서브젝트 프로세서(3)상에서 원래 동작하고 있을 때, 함수 라이브러리(17b, 17c)에 대한 서브젝트 실행가능한 파일(17a)내 호출은 PLT(17d)의 사용 및 서브젝트 링커 코드(17e)에 의해 관리된다. 이 점에서, 서브젝트 링커 코드(17e)는, 서브젝트 실행가능한 파일(17a)내 동적으로 링크된 함수를 취급하기 위한 중간 제어 구조로서 PLT(17d)를 사용한다.

[0047] 서브젝트 함수 호출은 제어 흐름을 서브젝트 링커 코드(17e)로 패스하며, 이것은 서브젝트 함수 호출과 연관된 PLT(17d)내의 링크 정보를 수정하고 그리하여 서브젝트 함수 호출이 함수를 실행하기 위한 서브젝트 코드에 링크된다.

[0048] 서브젝트 코드가 서브젝트 프로세서상에서 원래 동작할 때 PLT가 서브젝트 실행가능한 파일(17a)내 함수 호출을 서브젝트 라이브러리(17b)내 함수로 링크하기 위해 사용되는 과정은 이 문서의 도입부분에서 설명되었고, 당업자가 알 것이다. 솔라리스 OS내의 PLT 및 동작시간 연결의 사용에 대한 보다 많은 정보는

[0049] http://docs.sun.com/app/docs/doc/817-1984/6mhm7pllb

[0050] 에서 이용가능한다.

[0051] 또한 도 3에는, 이하에서 링크 오디터(link auditor)로 언급되는, 링크 오디팅 함수(link auditing functions)을 실행하는 서브젝트 코드가 도시된다. 솔라리스 OS내에서, 링크 오디터(17f)는 OS에 의해 제공되는 연결 오디트 인터페이스(link audit interface)와 양립가능한 코드를 이용함으로써 링커 활동을 모니터할 수 있다. 솔라리스 링크 오디트 인터페이스는 rtld-audit로 알려져 있다. rtld-audit에 관한 보다 많은 정보는

[0052] http://docs.sun.com/app/docs/doc/817-1984/6mhn7p124

[0053] 에서 이용가능한다.

[0054] 인터페이스와 일치하는 링크 오디터가 실행가능한 부분으로 로드(load)된다면, 여기서 포함된 오디트 루틴은 링커 실행의 다양한 단계에서 서브젝트 링커 코드에 의해 자동으로 호출된다. rtld-audit 인터페이스를 사용하는 것은, 링크 오디터가 실행가능한 서브젝트 코드(17a), 및 서브젝트 라이브러리(17b 및 17c)와 같은 로드된 오브젝트에 관한 정보에 접근할 수 있게 한다. 무엇보다, 링크 오디터는, 어플리케이션 및 라이브러리간의 정보 이동과 관련된 다른 정보 뿐만 아니라, 그러한 로드된 오브젝트간에 만들어진 연관에 대한 정보에 접근할 수 있다. 보다 많은 정보는

[0055] http://docs.sun.com/app/docs/doc/806-0641/6j9vuqujm

[0056] 에서 이용가능한다.

[0057] 또한, 링크 오디터는 링커에 의해 실행되는 PLT 업데이트 동안 함수 호출 및 그 리턴 값의 개별적인 주문을 모니터할 수 있다.

[0058] 서브젝트 프로세서상에서 원래 실행된다면, 서브젝트 라이브러리(17b)내 함수에 대한 호출은 서브젝트 제어 흐름을 PLT(17d)로, 그리고나서 서브젝트 링커 코드(17e)로 패스할 것이다. 서브젝트 링커 코드(17e)는 서브젝트 함수와 연관된 PLT(17d)내의 존재하는 링크 정보를 수정할 것이고, 그리하여 서브젝트 함수 위한 PLT 엔트리에 대한 다음 호출을 서브젝트 라이브러리 함수(17b)내 존재하는 함수를 실행하기 위한 관련된 서브젝트 코드로 링크한다.

[0059] ***함수 연결테이블(Function Linkage Table) 생성-링커 활동을 감시하기 위해 링크 오디터 사용***

[0060] 실행가능한 서브젝트(17a)에 대응되는 타겟 코드(21)를 생성하기 전에, 번역기(19)는 실행가능한 서브젝트(17a)의 헤더(17a_1)를 참조함으로써 그리고/또는 실행가능한 서브젝트(17a)의 하나 이상의 스캔을 실행함으로써

실행가능한 서브젝트(17a) 정보를 수집한다. 번역기(19)는 실행가능한 서브젝트(17a)와 연관된 PLT(17d)를 확인하기 위해 수집된 정보를 사용한다.

[0061] 번역기(19)가 실행가능한 서브젝트(17a) 정보를 수집한 후, 번역기(19)는 제1 시간동안 실행가능한 서브젝트(17a)를 통해 작업을 한다. 그리고, 그렇게 할 때 실행가능한 서브젝트(17a)내의 동적으로 링크된 함수 호출과 대응되는 함수를 실행하기 위한 코드간의 링크를 확립한다.

[0062] 실행가능한 서브젝트(17a)를 통해 번역기(19)가 작업을 하기 때문에, 서브젝트 코드(17a)에서 만난 동적으로 링크된 함수 호출 각각은 번역기(19)내에 보유된 FLT(function linkage table)(19a)를 거슬러 체크된다. 서브젝트 코드(17a)의 제1 번역에서 기대된 바와 같이, 함수 호출이 FLT(19a)내 어떤한 엔트리도 없는 것과 만나면, 서브젝트 코드(17)의 제어 흐름은 PLT(17d)를 경유하여 링커 코드(17e)에 뒤따른다. 그때 번역기(19)는 함수를 실행하기 위한 코드를 통해 작업을 진행할 수 있다.

[0063] 링커(17e)는 연결 정보를 링크 오디터(17f)로 패스하고, 번역기(19)는 이 정보를 FLT(19a)에 대한 엔트리를 추가하기 위해 사용한다. 특히, 번역기(19)는 실행가능한 서브젝트(17a)에 의해 호출된 각 함수에 대한 식별자 및 그 함수에 대한 위치를 수신하고 저장한다. FLT(19a)내 각 엔트리는 함수 식별자 및 대응되는 함수의 위치를 포함한다. 그러나, 번역기(19)는 링커 코드(17e)에 의한 PLT 영역(17d)의 업데이트에 대한 번역을 포함하는 타겟 코드를 생성하지 않는다.

[0064] 위에서 언급된 예시적인 실시예에서, 번역기(19)는 FLT(19a)를 만드는 rtld-audit 인터페이스와 양립가능한 링크 오디터(17f)를 사용한다. 그러나, 다른 예시적인 실시예는 링커 활동을 확인하고 그리하여 FLT(19a)를 채우고(populate) 유지하기 위해 다른 기술을 적용할 수 있다.

[0065] ***함수 연결 테이블 생성- 링커 활동을 직접 모니터***

[0066] 보다 예시적인 실시예에서, 대안적인 메커니즘은 PLT(17d)에 대한 수정을 검출하고, 실행가능한 서브젝트(17a)에 의해 호출된 동적으로 링크된 서브젝트 함수를 실행을 위한 코드 위치를 확정하기 위해 사용될 수 있다. 그러한 실시예에서, 번역기(19)는 실행가능한 파일 포맷의 지식으로부터 PLT를 인식할 수 있다. PLT 영역의 위치에 관한 정보를 이용하여, 번역기는, 그것이 서브젝트 코드 명령의 특징적인 시퀀스를 검출함으로써 PLT를 수정할 때 링커의 동작을 확인할 수 있다. SPARC로부터 번역의 예에서, 인식된 PLT 영역에 영향을 주는 서브젝트 명령 캐시 플러시 명령(subject instruction cache flush instructions)의 특징적인 시퀀스가 검출될 수 있다.

[0067] 서브젝트 코드내 제어 흐름이 PLT로부터 링커로 패스될 때, 그리고 링커가 응답으로 그 PLT 엔트리를 수정할 때, 서브젝트 코드 명령 캐시 플러시 명령이 실행된다. 플러시의 적절한 시퀀스가 발생할 때, 번역기는 실행가능한 서브젝트에 의해 호출된 함수를 실행을 위한 코드의 어드레스를 결정하기 위해 수정된 PLT 영역을 판독할 수 있다. 이 어드레스는 실행가능한 서브젝트내 동적으로 링크된 함수 호출과 함수를 실행하기 위한 코드간의 연관을 생성하기 위해 사용될 수 있다. 그러나, 번역기는 PLT 영역에 의해 제공되는 중간 제어 구조의 수정에 대응되는 타겟 코드를 생성하지 않는다. 그 전에, 번역기는 함수 식별자 및 FLT내 함수를 수행하기 위한 코드의 어드레스를 저장할 수 있다.

[0068] ***함수 연결 테이블내 정보 이용***

[0069] 전형적인 어플리케이션에서, 번역기(19)는 FLT 엔트리가 만들어진 하나 이상의 동적으로 연결된 함수 호출을 포함하는 서브젝트 코드(17)을 만날 것이다. 예를 들어, 번역기는 서브젝트 코드의 이전에 번역된 영역을 재-번역하기 위해 요청될 수 있다. 이러한 환경에서, 함수 호출은 FLT(19a)에 존재하는 대응되는 엔트리와 만난다. 그때 번역기(19)는 서브젝트 코드내 함수 호출과 함수를 실행하기 위한 대응되는 코드간의 연관을 확립하기 위해 FLT(19a)에 이전에 기록된 정보를 사용할 수 있다. 이 연관은 편리하게 직접적인 링크 형태일 수 있다.

[0070] 특정 함수의 재-번역은 PLT(17d) 및 링커 코드(17e)를 바이패스(bypass)하고, 그리고, 번역기(19)가 동적으로 링크된 함수 호출을 함수를 실행하기 위한 관련된 코드에 효율적으로 연관시키게 한다. 즉, 서브젝트 코드내 함수 호출을 FLT를 이용하여 함수를 실행하기 위한 대응되는 코드와 연관시키는 것은 서브젝트 코드내 중간 제어 구조에 대응되는 타겟 코드를 생성하는 것을 포함하지 않는다. 질문내 함수 호출과 관련된 PLT(17d)에 채우는 링크 정보를 수정하기 위한 서브젝트 링커 코드(17e) 또한, 번역되지 않는다. 재-번역동안 서브젝트 링커 코드(17e)는 서브젝트 코드에서 바이패스되고, 그리하여 번역기(19)에 의해 제공되지 않는다.

[0071] 위에서 설명된 바와 같이, 번역기(19)는 제1 시간동안 서브젝트 코드의 영역을 통해 작업하기 때문에, 그것은 FLT(19a)를 채운다. 번역기(19)는 링커 코드(17e)를 피하고 FLT(19a)내 정보를 이용하기 때문에, 다음 재-번역

은 보다 효율적일 것이다. 재-번역이 FLT(19a)를 채우기 위해 번역기가 작업을 하도록 요구하지 않는다.

[0072] ***간접적인 함수 호출***

[0073] 번역기에 의해 만난 몇몇 동적으로 링크된 함수 호출은 간접적인 함수 호출일 것이다. 간접적인 함수 호출은 변수 값에 의존하는 호출이다. 변수 값은 호출된 위치를 결정하고, 변경할 수 있다. 그리하여, 번역기는 간접적인 함수 호출에 대응되는 타겟 코드를 실행하는 지점에서 요청된 함수를 실행하기 위한 코드의 위치를 단지 결정할 수 있다.

[0074] 도 1과 관련된 위에서 언급한 바와 같이, 번역기가 함수 호출을 만날 때, 그것은 새로운 기본 블록을 확립한다. 번역기는 FLT에 저장된 정보에 거슬러 간접적인 함수 호출에 의해 참조된 서브젝트 어드레스를 체크할 수 있다. 어드레스가 FLT에 이미 존재한다면 번역기는 간접 함수 호출이 현재 참조하고 있는 함수에 대한 호출을 이전에 만났어야 한다. 그때, 번역기는 새롭게 확립된 기본 블록과 동일한 기능성(fuctionality)을 수행하는 것과 같이 그 함수 호출과 관련된 이전에 확립된 기본 블록을 처리할 수 있다. 이것은 번역기가 새롭게 확립된 기본 블록에 대신하여 이전에 확립된 기본 블록을 사용할 수 있도록 함으로써 번역기가 수행하도록 요청되는 작업의 양을 보다 감소시킬 수 있다.

[0075] ***서브젝트 코드내 동적으로 링크된 함수 호출을 서브젝트 코드와 다른 코드와 연관***

[0076] 위에서 설명된 실시예에서, 편리하게는, FLT내 유지된 위치는 대응되는 함수를 실행하기 위한 서브젝트 코드 명령의 어드레스이다. 그러나, 본 발명의 다른 실시예에서, FLT 에서 어드레스가 대응되는 함수를 실행하기 위한 다른 코드에 코드를 지시할 수 있도록 FLT 엔트리는 번역기(19)에 의해 선택될 수 있다. 이 방법으로 FLT를 이용함으로써, 번역기는 그 작업량을 줄일 수 있다. FLT 엔트리는,

[0077] 이전에 번역된 타겟 코드;

[0078] 타겟 OS(20)의 원 라이브러리(28)내 함수;

[0079] 번역기에 알려져 있고 타겟 코드가 편리하게 생성되는 최적화된 IR 영역; 또는,

[0080] 서브젝트 코드의 위치, 또는 위의 임의의 어느 하나를 표현하거나 생성하기 위해 사용될 수 있는 재-참조된 변수; 중 어느 하나에 대한 지점으로 번역기에 의해 선택될 수 있다.

[0081] 위에서 언급된 바와 같이 재-참조된 변수의 사용은 번역기가 함수 호출 및 대응되는 함수를 실행하기 위한 코드 간의 연관에 대한 보다 많은 제어를 실시할 수 있도록 한다. 특정 함수를 실행하기 위한 코드의 위치가 서브젝트 프로그램의 현재 주문 동안 고정되지 않는다면 이것은 바람직할 수 있다.

[0082] 설명적인 예에서, 번역기(19)는 x86 번역에 대한 SPARC를 실행한다. SPARC 타겟 시스템은 거기에 특정 인수상의 함수를 실행한 결과가 엄격하게 정의되는 하나 이상의 함수를 실행하기 위한 루틴을 포함할 수 있다. 그러한 엄격히 정의된 함수는 "닷(dot)" 함수로 알려져 있으며, 예를 들어, .umul, .smul 등과 같은 수학식의 범위를 포함한다. 이러한 ABI 닷 함수는

[0083] http://www.spare.com/standards/psABI3rd.pdf

[0084] 에 기록되어 있다.

[0085] ABI 닷 함수의 동작이 엄격하게 정의되기 때문에, 번역기는 ABI 닷 함수에 대한 호출로 확인된 함수 호출을 ABI 닷 함수와 같은 동일한 효과를 갖는 간단한 명으로 처리할 수 있다. 이것은, 예를 들어, FLT 엔트리를 확립시키고, 함수 호출을 대응되는 함수를 수행하기 위한 비-서브젝트 코드(non-subject code)의 영역과 연관시킴으로써, 번역기가 PLT를 완전히 바이패스하게 한다. 이 방법으로 PLT를 바이패스하는 것은 번역기의 작업로드를 줄인다. 무엇보다, 이 실시예에서, 번역기는, 함수 호출 자체를 대신하여 관련된 ABI 닷 함수의 효과를 갖는 코드를 추가함으로서 함수 호출을 처리하는 것과 관련된 업무를 피할 수 있다.

[0086] 원 함수 라이브러리와 공지된 일치를 갖는 서브젝트 함수에 대한 호출은 번역기(19), 그리고, 번역기의 작업로드를 줄이기 위해 이용된 원 함수의 일치에 의해 확인될 수 있다. 예를 들어, 서브젝트 라이브러리(17c)내 memcpy에 대한 호출은, FLT(19a)에서 원 라이브러리(28)내에 동일한 원 x86의 위치와 연관될 수 있다. 이것은 memcpy 함수의 서브젝트(SPARC) 버전을 번역하는 코스트를 제거할 수 있다. 게다가, memcpy 함수의 원 (x86) 버젼은 원 하드웨어 복잡함에 적용될 수 있으며, 그 하드웨어에 대한 가장 효율적인 방법으로 함수의 바람직한 효과를 얻을 수 있다.

[0087] 도 4는 동적으로 링크된 함수 호출을 처리하는 방법의 예시적인 실시예를 설명하는 개략적인 흐름도이다. 서브젝트 코드는 PLT 영역을 확인하기 위해 수신되고, 스캔된다(S101). 동적으로 링크된 함수 호출은 서브젝트 코드에서 확인되고(S102), 이것은 PLT에서 서브젝트 함수 호출과 연관된 링크 정보를 수정하기 위한 서브젝트 링커 코드로 서브젝트 제어 흐름을 패스하여, 서브젝트 함수 호출이 함수를 수행하기 위한 서브젝트 코드로 링크된다. FLT는 확인된 함수 호출에 대응되는 엔트리을 위해 체크되고, 그리고 FLT가 관련 엔트리를 포함한다면 방법은 타겟 코드가 생성되는 단계, 그 단계에서는 함수 호출이 함수를 실행하기 위한 코드와 관련된 단계 107로 진행한다. 단계 107에서 확립된 연관은 FLT에 저장된 바와 같은 위치 및 함수 식별자에 기초한다. 생성된 타겟 코드는 PLT에 의해 제공된 중간 제어 구조에 대응되는 타겟 코드를 포함하지 않는다.

[0088] 단계 103에서 FLT가 확인된 함수 호출에 대한 관련 엔트리를 포함하지 않는다고 결정되면, 링크 오디터는 함수 식별자 및 위치를 획득하기 위해 동작할 수 있고(S104), 및/또는 함수 식별자 및 위치는 서브젝트 코드 지시 캐시 플러시를 모니터함으로써 획득될 수 있다(S104). 그리하여, 단계 104 및/또는 105에서 획득된 정보는 FLT로 입력된다(단계 106). 그리하여 단계 107은, 단계 104 및/또는 105에서 획득된 정보에 기초하여, 실행된다.

[0089] 위 상세히 설명된 예시적인 실시예에서, 서브젝트 링커 코드는 번역기에 접근가능한 서브젝트 코드의 바디에 존재한다. 그러나, 다른 실시예에서 서브젝트 코드에서 링크를 실행하기 위한 링커 기능성은, 도 3에 도시된 링커 코드(19e)와 같이, 타겟 프로세서상의 특별히 실행을 위해 쓰여진 타겟 코드를 이용하여 번역기에 의해 제공된다. 무엇보다, 예시적인 실시예가 실행가능한 서브젝트와 연관된 PLT에 집중하였다 하더라도, 서브젝트 라이브러리(17b, 17c)는 또한 그 자신의 PLT 영역의 형태로 중간 제어 구조를 포함할 것이다. 여기서 언급된 방법 및 기기는 이 PLT 영역, 또는 중간 제어 구조를 포함하고 대응되는 문제를 야기하는 코드의 다른 동급의 영역에 동일하게 적용할 수 있다.

[0090] 여기서 언급된 기술을 이용함으로써, 번역기(19)는, PLT 또는 그와 유사한 것과 같은 중간 제어 구조내의 간접적인 연결 정보의 업데이트와 대응되는 번역을 생성하기 위해, 서브젝트 코드의 제어 흐름을 따를 필요는 없으며, 그리하여 동작 시간에 수정된 코드의 높은 밀집을 포함하는 서브젝트 코드의 영역을 번역하기 위한 처리 및 메모리 비용을 피할 수 있다.

[0091] 추가적으로, 효율적인 메커니즘은 번역기가 간접적인 연결 정보의 업데이트를 포함하는 서브젝트 코드를 실행하는 환경을 어드레스하기 위해 설명되었다.

[0092] 특별히, 발명자는 프로그램 코드 변환을 실행하기 위한 컴퓨터 시스템에서 유용한 방법 및 유닛을 개발하여 왔다. 그러한 방법 및 유닛은 서브젝트 프로그램 코드의 타겟 코드로의 동적인 이진 번역을 제공하는 동작시간 번역기를 가진 것과 같은 컴퓨터 시스템 셋업과 연결하는데 특별히 유용하다.

[0093] 본 발명은 또한 여기서 정의된 임의의 방법을 실행하는 번역기 기기에까지 확장된다. 또한, 본 발명은 여기서 정의된 임의의 방법을 실행하기 위해 컴퓨터에 의해 수행할 수 있는 명령을 거기에 기록한 컴퓨터-판독가능한 저장매체에까지 확장된다.

[0094] 본 발명의 적어도 몇몇 실시예는 단독으로 전용 하드웨어를 이용하여 구성될 수 있고, 여기서 사용되는 '모듈' 또는 '유닛'과 같은 용어는, 어떤 일을 수행하는, FPGA(Field Programmable Gate Array) 또는 ASIC(Application Specific Integrated Circuit)와 같은, 하드웨어 디바이스를 포함할 수도 있지만, 이에 한정되지 않는다. 대안적으로, 본 발명의 요소는 어드레스가능한 저장 매체에 있을 수도 있도록 구성될 수 있지만, 하나 이상의 프로세서상에서 실행하도록 구성될 수 있다. 그리하여, 본 발명의 기능적 요소는, 예를 들어, 소프트웨어 요소, 오브젝트-지향 소프트웨어 요소, 클래스 요소 및 태스크 요소, 프로세스, 기능, 특징, 절차, 서브루틴, 프로그램 코드의 세그먼트, 드라이버, 펌웨어, 마이크로코드, 회로, 데이터, 데이터 베이스, 데이터 구조, 테이블, 어레이 및 변수들과 같은 요소를 포함한다. 무엇보다, 바람직한 실시예가 이하 언급된 요소, 모듈 및 유닛에 대한 참조와 함께 설명되었다 할지라도, 그러한 기능적인 요소는 더 작은 요소로 결합되고, 추가적인 요소로 분리될 수 있다.

[0095] 본 발명의 기기 및 방법에 대한 다른 특징은 위 실시예 각각에서 분리되어 설명되었다. 그러나, 여기서 설명된 각 실시예에 분리된 특징이 여기서 설명된 다른 실시예에 결합될 수 있다는 것은 발명자의 완전한 의도이다.

[0096] 당업자는 언급된 바람직한 실시예의 다양한 적용 및 수정은 본 발명의 범위 및 정신으로부터 분리됨이 없이 구성될 수 있다는 것을 인정할 것이다. 그리하여, 첨주된 청구항의 범위내에서, 본 발명은 여기서 상세하게 언급된 바와 다르게 실행될 수 있다는 것을 이해하여야 할 것이다.

[0097] 몇몇 바람직한 실시예가 도시되고 설명되었다 하더라도, 첨부된 청구항에 정의된 바와 같이, 다양한 변경 및 수정이 본 발명의 범위를 벗어남이 없이 만들어 질 수 있다는 것을 당업자에 의해 인정되어야 할 것이다.

[0098] 주의해야 할 점은 이 출원과 관련하여 이 명세서와 동시에 또는 이전에 제출되고 이 출원과 관련하여 이 명세서와 함께 공개되고 이 명세서와 함께 공개 열람에 오픈된 모든 서류들 및 문서들에 있고, 모든 이와 같은 서류들 및 문서들의 내용들은 참조로 본원에 포함된다.

[0099] 이 명세서(첨부 청구항들, 요약 및 도면들을 포함)에 개시된 모든 특징들 및/또는 그렇게 개시된 임의의 방법 또는 공정들의 모든 단계들은 이와 같은 특징들 및/또는 단계들의 적어도 일부가 상호 배타적인 조합들을 제외하고, 어떠한 조합으로도 조합될 수 있다.

[0100] 이 명세서(첨부 청구항들, 요약 및 도면들을 포함)에 개시된 각각의 특징은 달리 명확하게 설명되지 않는다면, 동일, 등가 또는 유사 목적을 달성하는 다른 특징들에 의해 대체될 수 있다. 따라서, 달리 명확하게 설명되지 않는다면, 개시된 각 특징은 등가 또는 유사 특징들만의 일반 시리즈의 일 예이다.

[0101] 본 발명은 상기 실시예(들)의 상세들로 제한되지 않는다. 본 발명은 이 명세서(첨부 청구항들, 요약 및 도면들을 포함)에 개시된 특징들의 어느 신규한 하나 또는 임의의 신규의 조합, 또는 그렇게 개시된 임의의 방법 또는 공정의 어느 신규의 하나 또는 어느 신규의 조합으로 확대된다.

도면의 간단한 설명

[0026] 수반되는 도면은 설명의 일부와 결합되고 구성되며, 현재의 바람직한 수행을 설명하면, 이하처럼 설명된다.

[0027] 도 1은 본 발명의 일시예가 애플리케이션을 발견하는 경우의 기기를 설명하는 블록도이다.

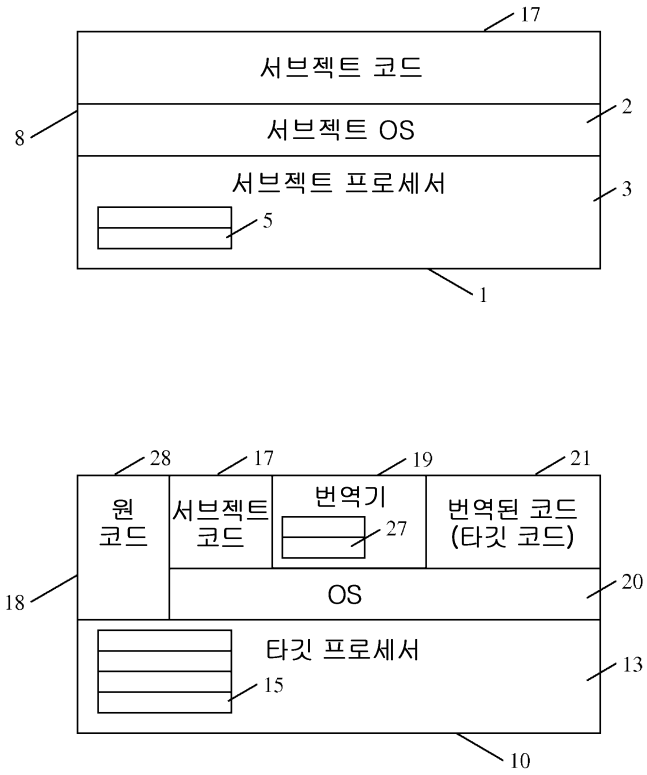
[0028] 도 2는 본 발명의 실시예에 의해 적용되는 바와 같이 번역기 부의 도식도이다.

[0029] 도 3은 본 발명의 실시예에 의해 적용되는 기기를 설명하는 블록도이다. 그리고,

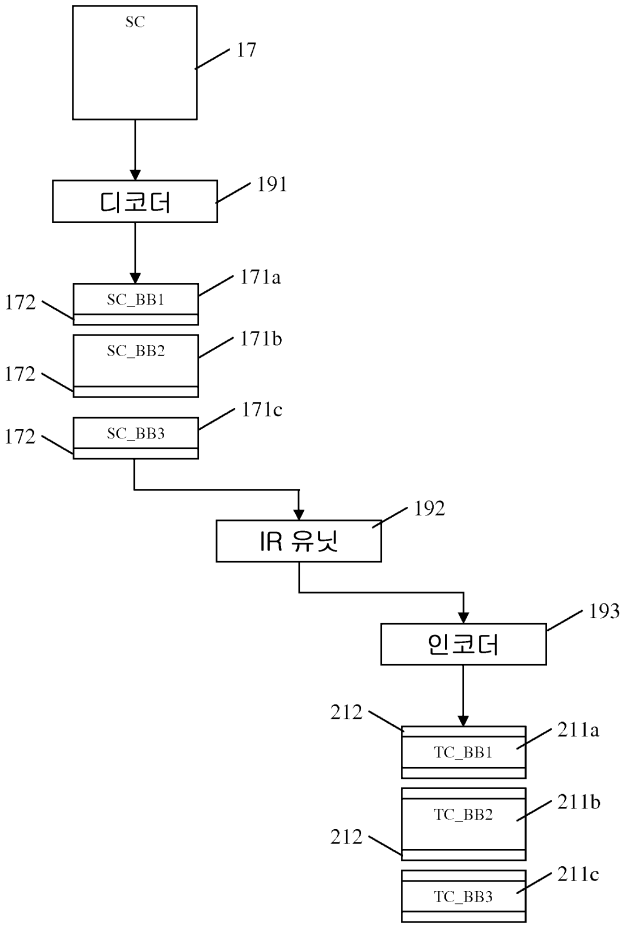
[0030] 도 4는 예시적인 함수 호출을 취급하는 방법을 설명하는 개략적인 흐름도이다.

도면

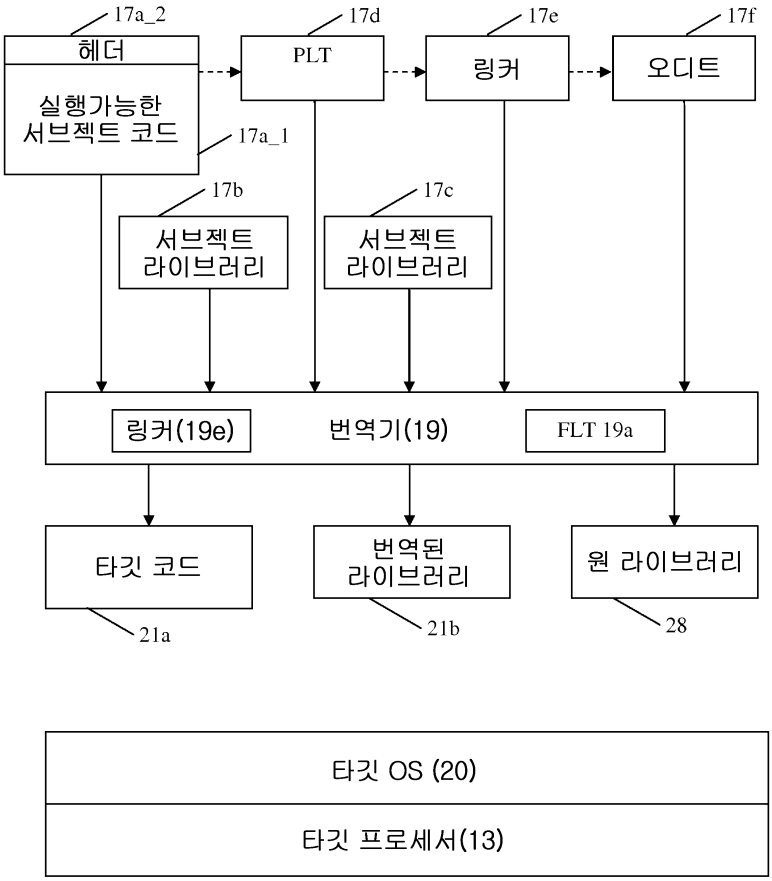
도면1


17
서브젝트 코드
8
2
서브젝트 OS
서브젝트 프로세서
3
5
1
28
17
19
21
원
코드
서브젝트
코드
번역기
27
번역된 코드
(타깃 코드)
18
OS
20
타깃 프로세서
13
15
10

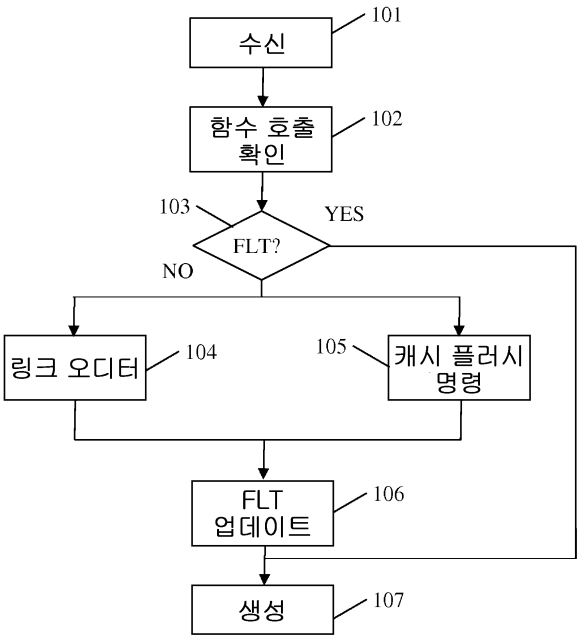
도면2


SC
17
디코더
191
SC_BB1
171a
172
SC_BB2
171b
172
SC_BB3
171c
172
IR 유닛
192
인코더
193
212
TC_BB1
211a
TC_BB2
211b
212
TC_BB3
211c

도면3


17a_2
헤더
실행가능한
서브젝트 코드
17a_1
17d
PLT
17e
링커
17f
오디트
17b
서브젝트
라이브러리
17c
서브젝트
라이브러리
링커(19e)
번역기(19)
FLT 19a
타깃 코드
21a
번역된
라이브러리
21b
원 라이브러리
28
타깃 OS (20)
타깃 프로세서(13)

도면4


수신
101
함수 호출
확인
102
103
FLT?
YES
NO
링크 오디터
104
105
캐시 플러시
명령
FLT
업데이트
106
생성
107